Свободные программы и системы в школе

Версия 1.0.1 от 01 декабря 2003 г.

Максим Отставнов <maksim@otstavnov.com>

Курс лекций, включенных в брошюру, знакомит читателя с популярными свободными программами и системами, полезными при преподавании информатики в средней школе. В обзор вошли основы открытых операционных систем, сведения о пакете «офисных» программ OpenOffice.org, коммуникационном пакете Mozilla, графическом редакторе GIMP, современных графических средах GNOME и KDE и других программах.
Для преподавателей информатики и методистов, а также для студентов и аспирантов соответствующих специальностей.

Использован текст лекций, публиковавшихся автором в приложении к газете «Первое сентября» «Информатика» (http://inf.1september.ru) в 2002-3 г., материалы брошюры «Прикладные свободные программы в школе» (М.: 2003 г.), а также фрагменты статей, ранее публиковавшихся в журналах «Компьютерра» и «Домашний компьютер».

Оглавление

Введение. Зачем программам быть свободными?

На сегодня в школе, как и в некоторых других сегментах рынка, преобладают «альтернативные», нестандартные операционные системы и платформы (такие, как «МС-ДОС», «Майкрософт Уиндоуз», «Майкрософт Уиндоуз НТ», «МакОС» версий до X). Пользование ими описывается в учебниках, на них ориентируются авторы отечественных учебных программ.

В течение долгого времени «цена вопроса» просто не становилась поводом для сколько-либо серьезного анализа — ни для кого не секрет, что доля контрафактного ПО в российских школах очень высока, а общественное мнение, увы, пока склонно считать «легальность» программного обеспечения вопросом скорее отвлеченно-академическим, нежели насущно-практическим, а уж там, где дело касается зарубежных правообладателей, — относиться к нему не как к «пиратскому», а как к «трофейному» (что, впрочем, также имеет свои основания).

Имеются очень, на наш взгляд, веские причины, чтобы постепенно отказаться от этой недоброй традиции и вернуться в русло, более соответствующее магистральным линиям развития информационных и коммуникационных технологий:

- нестандартные системы ненадежны и небезопасны. С распространением в школах компьютеров и особенно сетей (включая доступ к региональным и глобальным сетям) ущерб от вирусов, несанкционированного доступа к информации и т.п. станет заметной статьей издержек информатизации;
- нестандартные системы дороги. Практически для всех стандартных технологий имеются свободные реализации (или, по крайней мере, конкурентный рынок реализаций), в то время, как ожидать появления свободной реализации, допустим, интерфейса той же «Майкрософт Уиндоуз» в обозримом будущем не приходится, и цены будут оставаться монопольно завышенными. Мы не сторонники «экономии на детях», но, право же, выделяемые на информатизацию школы деньги можно расходовать гораздо более разумно — от их вложения в перспективные разработки до повышения окладов учителей и обслуживающего персонала;
- нестандартные системы ограничивают выбор оборудования и зачастую предъявляют завышенные требования к его параметрам;
- поставщик нестандартных систем и его партнеры получают неосновательное преимущество в других сегментах рынка. Если целенаправленно подменять обучение в школе общим принципам и стандартным технологиям изучением конкретных программ,

через некоторое время специфические навыки пользования ими распространятся в обществе настолько, что поставщикам конкурирующих технологий и решений пробиться на рынок будет совсем нелегко;
- использование несвободного ПО отрезает учащихся (и их наставников!) от современных технологических решений.

Могут ли сегодня свободные реализации стандартных технологий качественно обеспечить учебный процесс в школе? Вопрос неоднозначный, и мы видим ответ на него двояким:

- с технической точки зрения — безусловно. Имеющийся пул СПО с избытком перекрывает потребности любого разумного учебного курса по информатике, причем большинство программ способно работать на массовом и недорогом оборудовании, распространенном в школах (ПК архитектур IBM PC и «Эппл Макинтош», терминальные классы от «Сан Майкросистемз» на процессорах «УльтраСПАРК» и т.п.), включая весьма «пожилые» модели. Некоторые программы требуют определенных усилий по локализации (переводу элементов интерфейса и документации), однако эти затраты на порядок меньше, чем стоимость «легализации» несвободных альтернатив;
- с организационной — все зависит от того, как поставить дело.

Мы не беремся предсказывать масштабы и темп освоения свободного ПО российской школой, но заметим, что любые попытки волевого «насаждения» тех или иных решений или конкретных систем «сверху», на наш взгляд, к успеху не приведут.

Гораздо разумнее со вниманием отнестись к тому интересу, который уже проявляется учителями к свободному ПО (и наоборот, авторов и поставщиков свободных программ — к учебному процессу) и поддержать процесс его освоения в том темпе, который окажется «естественным».

В начале 2002-3 учебного года к автору обратилась группа методистов с предложением описать десяток наиболее применимых в школьной учебной практике программ. Полагая, что такой рассказ интересен и более широкой аудитории, автор договорился с редакцией «Информатики» (приложения к газете «Первое сентября»), о «цикле лекций» на страницах этого издания[1], текст которых использован в этой книге, так же как и текст «лекций» следующего цикла «Введение в открытые ОС» (2003-4 уч. г.), публикуемого там же[2].

[1] Первые десять «лекций» упомянутого цикла также публиковались отдельной брошюрой [21].

[2] Пользуясь случаем, мы рады поблагодарить редакцию «Информатики» за воз-

0.1 Право и экономика ПО

Поскольку существует изрядная путаница с терминологией, имеет смысл привести определения далее используемых терминов. Это особенно уместно, когда мы говорим о программах в школе, ведь свободное ПО пока очень слабо проникло в эту важнейшую сферу применения компьютеров.

Свободными называются программы, автор (или иной обладатель имущественных авторских прав) которых опубликовал (обнародовал) их в сопровождении так называемой «свободной лицензии», или, если следовать терминологии отечественного законодательства, публичного авторского договора, передающего приобретателю права: 0) *пользоваться программой* для любых целей (в рамках российского законодательства это тавтология, так как у правообладателя нет правомочия ограничивать цели, в которых собственник экземпляра программы может ее применять) и на неограниченном количестве компьютеров или мест в сети; 1) беспрепятственно *получать доступ к ее исходным кодам*; 2) изготовлять (производить) неограниченное количество дополнительных ее экземпляров, как для собственного пользования, так и для *распространения* или сдачи в прокат/аренду на тех же условиях, возмездно или безвозмездно (по своему выбору); 3) *модифицировать* ее как для собственного пользования, так и для распространения на тех же условиях.

Конкретная «лицензия» (условия конкретного договора) могут предоставлять приобретателю дополнительные правомочия, безусловно или на определенных условиях, и это не делает программу несвободной. *Несвободной* является программа, распространяемая на условиях, ограничивающих вышеперечисленные права приобретателя.

Свободные программы (free software) не следует, как это часто делают, путать со «свободно распространяемыми» (shareware, иногда почему-то называемыми у нас «условно-бесплатными») или «бесплатно лицензируемыми» (freeware).

Также следует иметь в виду, что, хотя термин «программы с открытыми исходниками» (open source software) часто используется как синоним «свободных программ», им иногда злоупотребляют.

Свободное ПО не следует путать и с «открытым» (open systems, open software): «открытость» относится к соблюдению стандартов на интерфейсы, и только, а свобода — к условиям лицензирования и модели раз-

можность обратиться к широкой педагогической аудитории и, конечно же, всем читателям за отзывы, замечания и предложения, которые мы постарались учесть при подготовке этой книги.

работки.

И наконец, не следует путать «коммерческое» с несвободным, а «некоммерческое» — со свободным. Значительная часть (вероятно, большая) свободного кода разрабатывается в коммерческих рамках. В то же время, существует большое количество некоммерческого несвободного кода (freeware).

Узел терминологических тонкостей и концептуальных сложностей можно разрубить, введя такое определение: *свободные программы — это программы, все услуги по разработке, модификации, сопровождению и поддержке которых продаются на свободном рынке*.

Если, например, свободный дистрибутив какой-либо версии «ГНУ/Линукс», включающий, помимо операционной системы и нескольких операционных сред, большое количество прикладных программ, стоит от нескольких десятков до нескольких тысяч рублей (в то время, как «набрать» даже небольшую часть этой функциональности программами несвободными может обойтись и в десять, и в сто раз дороже[3]), причина этому не в «бесплатности» чего-либо, а в конкурентности рынка.

Мы избавим читателя от обсуждения анатомии рынка (как формируются цены, кто, за что и каким образом вознаграждается), отослав интересующихся к соответствующей литературе [2, 3]. Важно то, что этот рынок успешно развивается уже в течении четверти века, есть примеры исполнения им задач, немыслимых для отдельных корпораций (самый яркий из них — разворачивание в девяностых на основе свободных программ и систем всемирного сообщества сетей Интернет), и, наконец, то, что после некоторых колебаний, большая часть лидеров компьютерной отрасли сегодня практически однозначно высказывается в поддержку свободного ПО.

0.2 Применимость СПО при реализации «Обязательного минимума...»

Информатика, будучи относительно молодым предметом в школьной практике (особенно в сравнении с учебными предметами, чей «возраст» превышает две тысячи лет), с неизбежностью вызывает споры среди теоретиков педагогики и практикующих учителей о своем содержании.

[3] Например, типичная цена одно-двухдискового дистрибутива российской сборки, включающего софт, достаточный для обеспечения школьной программы по информатике, не превышает трехсот рублей, в то время, как комплект из «Майкрософт Уиндоуз» и «Майкрософт Офис», покрывающий лишь часть программы, со всеми скидками обойдется не дешевле трех тысяч *на каждое рабочее место*.

Спорят и о том, прагматическая (знакомство с миром компьютерных технологий) или теоретическая (основы компьютерных наук) ориентация должна превалировать в школе, и о том, стоит ли программирование делать частью общего школьного образования, и даже о том, должна ли информатика развиваться в виде отдельного предмета. Периодически этот спор выплескивается за рамки профессионального педагогического сообщества на страницы печати, что также вполне понятно.

Мы попытаемся в данном случае уклониться от участия в этом споре и не заявлять свою позицию. Возможно, это не вполне соответствует традициям русского интеллигентского общества, в котором не принято, не решив ряда «последних вопросов» (в том числе, кто виноват, что делать и, главное, чему и как учить), предпринимать какие-либо действия. Но рамки у этой книги вполне прагматические, и можно только надеяться, что данные в таких рамках ответы окажутся в достаточной степени инвариантными для ответов на вопросы более глубокие (и, охотно согласимся, в конечном итоге более важные), и что-то полезное для себя в изложенном материале найдут партизаны самых разных точек зрения по этим глубоким и важным вопросам.

Упомянутые прагматические рамки заключаются в следующем. Если для освоения грамотности и счета школьнику неплохо бы овладеть ручкой и карандашом и понять, чем одна от другого отличается (вне зависимости от того, выделяется ли чистописание в отдельный учебный предмет, как это было принято ранее, или же, как сейчас, освоение инструментов письма идет на тех же уроках, где ученик знакомится с буквами и их чтением), то уж наличие в «Обязательном минимуме содержания образования по информатике» [1] списка приложений компьютера («информационных технологий»), с которыми стоит познакомить школьников, протестов у нас не вызывает.

Среди них:

- создание и редактирование текстов (в широком смысле),
- создание и редактирование растровой графики,
- создание и редактирование векторной графики,
- работа с электронными таблицами,
- работа с базами данных,
- работа с электронной почтой и новостными конференциями,
- обмен файлами в сетях,
- работа с мультимедиа (только уровень Б),
- работа с распределенными гипермедиа (WWW) в сетях.

Кроме того, другие пункты «Минимума» (не отнесенные авторами к «информационным технологиям») также с неизбежностью подразумевают

знакомство с определенными приложениями компьютера (если, конечно, автор конкретного курса не предпочел чисто теоретический режим знакомства с материалом, соответствующим тому или иному пункту).

Например, для того, чтобы предметно показать учащимся разницу в различных способах представления (кодирования) информации (п. 2 «Обязательного минимума...»), неплохо иметь под рукой соответствующую программу просмотра. Знакомство с языками программирования (п. 5) предполагает и знакомство с интерпретатором или компилятором и другими компонентами инструментальной среды и т.д. и т.п. Ну и, разумеется, чтобы учащийся мог работать со всеми этими программами, ему необходимы базовые навыки обращения с операционной системой/средой (вне зависимости от того, присутствует ли такая тема отдельной строкой в учебном плане).

Поскольку, как упоминалось выше, СПО развивалось в направлении от инструментальных программ к системным и, далее, к прикладным, сегодняшнее положение дел, при котором все перечисленные приложения имеют свободную реализацию, сложилось не сразу. Однако уже в течение нескольких лет основная масса свободных прикладных программ находится на уровне, достаточном для их применения в школьном учебном процессе, и в педагогическом сообществе постепенно накапливается опыт их использования.

В России, насколько нам известно, наиболее последователен в этом плане опыт Центра компьютерных технологий Московского государственного индустриального университета, сеть которого используется, в том числе, и для практических занятий учащихся московских школ[4].

0.3 Логика и последовательность освоения СПО

Логика и последовательность изложения материала в этом курсе существенно отличается от логики, в которой написано большинство книг, посвященных СПО.

Чаще всего авторы исходят из того, что последовательность внедрения

[4] В ноябре 2001 года, во время рабочей встречи «Свободное программное обеспечение», проведенной Высшей школой экономики и издательским домом «Компьютерра» в рамках разворачивания федеральной целевой программы «Электронная Россия» [2], было достигнуто принципиальное соглашение о том, чтобы учебные материалы, разработанные в ЦКТ МГИУ и касающиеся применения СПО в учебном процессе, были опубликованы также под свободной лицензией. И коллеги из МГИУ выполнили свое обещание, благодаря чему на их сайте можно найти учебник «Практическая информатика» [4] (также опубликован и выпущен в продажу в виде традиционного двухтомного издания).

программ и пакетов, относящихся к различным категориям, будет соответствовать последовательности построения информационной системы (например, школьной сети). Это вполне оправдано в случае разработки и реализации такой системы «с нуля», однако может вызвать существенные сложности в ситуациях, когда система уже существует, развернута на базе несвободного (и, как следствие, почти всегда нестандартного) ПО, нужна поддержка ее роста и развития, и администрация учебного заведения обращается к СПО именно как к средству обеспечения такого развития.

«Сплошная миграция», при которой администрации необходимо внедрять (а всем пользователям, включая учителей и учащихся, осваивать) и системное, и прикладное (а в случаях, если проводятся какие-то свои разработки, и инструментальное) ПО одновременно, — объективно очень сложный и болезненный процесс. Если говорить об учебном заведении, то он сложен вдвойне, поскольку слишком неодинаков уровень пользователей различных категорий. Если говорить о школах, где единственный преподаватель информатики — сам себе и учитель, и методист, и системный администратор, и лаборант (а не большинство ли наших школ таковы?), миграция по этому сценарию требует, без преувеличения, героических усилий, ожидать которых в массовом порядке не вполне разумно.

Однако поскольку значительная часть свободного прикладного программного обеспечения портирована (перенесена), в том числе, и в несвободное окружение (включая получившие широкое распространение в наших школах операционные среды «Майкрософт Уиндоуз» и «МакОС»), миграцию можно облегчить, проведя ее в два этапа (которым в учебном заведении могут соответствовать два академических года): сначала прикладная часть, а затем — системная.

В настоящей небольшой книге мы ограничились кратким введением в открытые операционные системы (глава 1) и обзором наиболее универсальных свободных *прикладных* платформ (пользовательская графика, глава 2) и пакетов (коммуникационный пакет «Мозилла», глава 3, «офисный» пакет «ОткрытыйОфис», глава 4, пакет растровой графики «ГИМП», глава 5).

Глава 1. Краткое введение в открытые ОС

Традиционно существовало два жанра введений в ОС: введение в архитектуру ОС с точки зрения программиста, и введение в пользование ОС с точки зрения оператора. Как правило, такие книги освещают также элементы администрирования (чаще — какой-то конкретной ОС).

В последнее десятилетие появился специфический третий жанр. Он предполагает «полное погружение» студента в материал, и в нем параллельно вводятся сведения, необходимые для оператора и для администратора. Появление этого жанра связано с массовым введением в эксплуатацию открытых ОС на весьма капризном «персонально-компьютерном» оборудовании и с тем, что во все возрастающем количестве новые пользователи открытых ОС осваивали их самостоятельно.

В последние годы ситуация опять изменилась: сегодня доступно множество дистрибутивов свободных ОС, позволяющих в большинстве случаев получить по крайне мере базовую стандартную функциональность «из коробки», и продается достаточное количество компьютеров «персонального» и «домашнего» класса с предустановленной ОС «ГНУ/Линукс». Поэтому новичок может совершенно спокойно освоить сначала основы пользования системой, а затем, при необходимости, ее администрирование.

Предлагаемый в настоящем разделе курс подпадает под определение второго из перечисленных жанров, а его отличительными особенностями являются:

- компактность (в случае необходимости в жертву приносились «технические» аспекты в пользу «гуманитарных», т.е. сведениям, необходимым для понимания литературы и общения на заданную тему, отдавалось предпочтение перед сведениями, необходимыми для осуществления тех или иных действий),
- ориентация на современную версию стандарта ОС (ИСО/МЭК 9945:2001), а не особенности той или иной ОС.

Существенной сложностью при изучении открытых систем остается неполнота переводов встроенной документации, поэтому приложены (раздел 1.10) справочные страницы обсуждаемых команд (полностью или в части, соответствующей учебному материалу).

Для воспроизведения примеров и другой практической работы потребуется доступ к терминалу открытой ОС. Идеальным вариантом является доступ к администрируемой и поддерживаемой в порядке системе. Если такой возможности нет, можно установить на имеющийся компьютер

ОС «ГНУ/Линукс» или «БСД», руководствуясь документацией к дистрибутиву. Пользователи «Макинтошей» с установленной «МакОС X» могут получить доступ к терминалу ОС в соответствии с документацией последней.

Наименее проблемной будет установка из дистрибутива «Дебиан ГНУ/Линукс», особых сложностей также не должно быть с «АЛЬТ Линукс», «АСПЛинукс», «РедХэт Линукс», «Линукс-Мандрейк» или «ФриБСД». Любой из этих дистрибутивов позволяет осуществить установку на компьютеры самой популярной архитектуры IA-32 («IBM PC-совместимые»). Еще лучше, если у студента есть знакомый, администрирующий любую открытую ОС, который сможет помочь с установкой; тогда дистрибутив должен порекомендовать он.

При установке следует обратить внимание на правильные национальные установки (кириллица с кодовой таблицей «KOI8-R», клавиша переключения кодовых подстраниц) и на то, чтобы после загрузки *не* осуществлялся автоматический запуск графической оконной системы «Икс» («X Window System»). Последняя в нашем курсе вообще не нужна.

Для нормальной работы с оболочкой и утилитами должно хватить возможностей процессора Intel 386-DX66 или аналога, 16 МБ ОЗУ, 100 МБ дискового пространства, хотя некоторые из перечисленных дистрибутивов могут требовать расширенного набора команд «Пентиум» и содержать программу установки, более требовательную к ресурсам (особенно к ОЗУ).

Современные загрузчики позволяют осуществлять попеременную загрузку на одном компьютере более одной ОС (в том числе альтернативных), так что если тот же компьютер эксплуатируется и под «МС-ДОС» («Майкрософт Уиндоуз»), это не будет помехой (прочитайте внимательно разделы документации по установке и не забудьте осуществить резервное копирование данных перед переразбиением дисков на разделы, если таковое потребуется).

В крайнем случае можно запустить оболочку и команды под «Майкрософт Уиндоуз», пользуясь такими пакетами, как «Cygwin», «Unix Services for Windows» или «UWIN», что, впрочем, не рекомендуется.

1.1 Операционные системы

Существуют две группы определений операционных систем: «совокупность программ, управляющих оборудованием» и «совокупность программ, управляющих другими программами». Обе они имеют свой точный технический смысл, который, однако, становится ясен только при

более детальном рассмотрении вопроса о том, зачем вообще нужны операционные системы.

Есть приложения вычислительной техники, для которых ОС излишни. Например, встроенные микрокомпьютеры содержатся сегодня во многих бытовых приборах, автомобилях (иногда по десятку в каждом) и т.п. Зачастую такой компьютер постоянно исполняет лишь одну программу, запускающуюся по включении. И простые игровые приставки — также представляющие собой специализированные микрокомпьютеры — могут обходиться без ОС, запуская по включении программу, записанную на вставленном в устройство «катридже» или компакт-диске.

(Многие встроенные компьютеры и даже некоторые игровые приставки на самом деле работают под управлением своих ОС. Мы не будем подробно останавливаться на этих классах вычислительных систем. Тем не менее, ответ на вопрос, почему это так, к концу этого раздела должен быть в общих чертах понятен).

Операционные системы, в свою очередь, нужны, если:

- вычислительная система используется для различных задач, причем программы, исполняющие эти задачи, нуждаются в обмене данными. Из этого следует необходимость в универсальном механизме хранения данных; в подавляющем большинстве случаев ОС отвечает на нее реализацией *файловой системы*. Современные ОС, кроме того, предоставляют возможность непосредственно «связать» вывод одной программы со вводом другой, минуя относительно медленные дисковые операции;

- различные программы нуждаются в выполнении одних и тех же рутинных действий. Например, простой ввод символа с клавиатуры и отображение его на экране требуют исполнения сотен машинных команд, а дисковая операция — тысяч. Чтобы не программировать их каждый раз заново, ОС предоставляют *системные библиотеки* часто используемых подпрограмм (функций);

- между программами и пользователями системы необходимо *распределять полномочия*, чтобы пользователи могли защищать свои данные от чужого взора, а возможная ошибка в программе не вызывала тотальных неприятностей;

- необходима возможность имитации «одновременного» исполнения нескольких программ на одном компьютере, осуществляемой с помощью приема, известного как «разделение времени». При этом специальный компонент, называемый *планировщиком*, «нарезает» процессорное время на короткие отрезки и предоставляет их поочередно раз-

личным исполняющимся программам (*процессам*);

- наконец, оператор должен иметь возможность как или иначе управлять процессами выполнения отдельных программ. Для этого служат *операционные среды*, одна из которых — оболочка и набор стандартных утилит — является частью ОС (прочие, такие, как графическая операционная среда, образуют независимые от ОС прикладные платформы).

Таким образом, современные универсальные ОС можно охарактеризовать как 1) использующие файловые системы (с универсальным механизмом доступа к данным), 2) многопользовательские (с разделением полномочий), 3) многозадачные (с разделением времени).

Многозадачность и распределение полномочий требуют определенной иерархии привилегий компонентов самой ОС. В составе ОС различают три группы компонентов:

- *ядро*, содержащее планировщик, драйверы устройств, непосредственно управляющие оборудованием, сетевую подсистему, файловую систему;
- *системные библиотеки* и
- *оболочку с утилитами*.

Большинство программ, как системных (входящих в ОС), так и прикладных, исполняются в непривилегированном (пользовательском) режиме и получают доступ к оборудованию (и, при необходимости, к другим ядерным ресурсам, а также ресурсам иных программ) только посредством системных вызовов. Ядро исполняется в привилегированном режиме: именно в этом смысле говорят, что ОС (точнее, ее ядро) управляет оборудованием.

Текущая редакция стандарта на ОС содержит определения около тысячи системных вызовов (часть из которых должна реализоваться только в определенных классах систем; например, в системах «реального времени») и около двухсот команд оболочки и утилит ОС. Стандарт определяет лишь функции вызовов и команд, и не содержит указаний относительно способов их реализации.

Стандарт, кроме этого, определяет способ адресации файлов в системе, локализацию (установки, касающиеся национально-специфических моментов, таких, как язык сообщений или формат даты и времени), совместимый набор символов, синтаксис регулярных выражений, структуру каталогов в файловой системе, формат командной строки и некоторые другие аспекты поведения ОС.

В определении состава ОС значение имеет критерий *операциональной целостности* (замкнутости): система должна позволять полноценно использовать (включая модификацию) свои компоненты. Поэтому в полный состав ОС включается и набор инструментальных средств (от текстовых редакторов до компиляторов, отладчиков и компоновщиков). Операциональной замкнутостью обладают системы, удовлетворяющие «разработческому» профилю в терминах стандарта.

Краткая история открытых ОС

К концу шестидесятых годов XX в. операционным системам как классу программного обеспечения шел уже второй десяток. Были разработаны больше сотни различных ОС для разных компьютеров, из них полтора десятка находились в «боевой» эксплуатации. На рубеже шестидесятых и семидесятых в одном из исследовательских подразделений американской телекоммуникационной монополии «Эй-Ти-энд-Ти» была выполнена разработка, ставшая важнейшей вехой в истории ОС: система «Юникс».

Задуманная и реализованная Кеном Томсоном при участии нескольких коллег, она вобрала в себя многие черты более ранних ОС, но обладала целым рядом свойств, отличающих ее от большинства предшественниц:

- компонентная архитектура: принцип «одна программа — одна функция» плюс мощные средства связывания различных программ для решения возникающих задач;
- минимизация ядра (кода, выполняющегося в привилегированном режиме процессора) и количества системных вызовов;
- независимость от аппаратной архитектуры и реализация на языке высокого уровня (язык программирования C стал «побочным продуктом» разработки «Юникс»).

«Юникс», благодаря своему удобству прежде всего в качестве инструментальной среды (среды разработки), была тепло принята сначала в университетах, а затем и в отрасли, получившей прототип единой ОС, которая могла использоваться на самых разных вычислительных системах и, более того, быстро и с минимальными усилиями перенесена на вновь разработанную аппаратную архитектуру.

Одним из центров развития «Юникс» стал Университет Калифорнии в Беркли, там было создано множество средств, дополняющих систему и развивающих ее концепцию. В конце концов, в Беркли создали свой вариант ОС той же архитектуры, получивший название «БСД».

Задачу разработать независимую (от авторских прав «Эй-Ти-энд-Ти») ре-

ализацию той же архитектуры поставил и Ричард Столлмен, основатель проекта «ГНУ» (характерно, что аббревиатура расшифровывается как GNU's Not Unix, т.е. «ГНУ — это не "Юникс"»). В ходе разворачивания проекта (1980-90е гг.) было создано множество утилит и инструментальных средств, которые сегодня активно используются в «БСД» (входя в систему) и «Юникс» (как правило, распространяемые в качестве дополнений), а также являющихся основой операционных систем на основе ядра «Линукс», разработка которого была запущена и возглавляется с начала девяностых Линусом Торвальдсом.

Таким образом, на сегодня существует три семейства *открытых операционных систем*, концептуально происходящих от «Юникс», но реализованных независимо:

- основанные на «Эй-Ти-энд-Ти Юникс» (в разнообразных фирменных вариантах, таких как AIX (компания IBM), «Солярис» (компания Sun Microsystems) и т.п.),
- «БСД» (в него входят FreeBSD, OpenBSD, NetBSD, а также Darwin, являющаяся основой MacOS X),
- «ГНУ/Линукс» (в различных вариантах, или дистрибутивах, таких как Debian GNU/Linux, RedHat Linux, Linux-Mandrake и пр.)

Системы, содержащие код, изначально написанный в AT&T, несвободны[5], а «БСД» и «ГНУ/Линукс» разрабатываются под свободными лицензиями.

Благодаря конкурентности реализаций архитектура открытых ОС стала вначале фактическим отраслевым стандартом, а затем обрела статус и стандарта юридического (свежая версия принята Международной организацией стандартизации (ISO) в 2001 г.).

Стандартизация ОС означает возможность безболезненной замены самой ОС или оборудования при развитии вычислительной системы или сети и дешевого переноса прикладного программного обеспечения (строгое следование стандарту предполагает полную совместимость программ на уровне исходного текста; из-за профилирования стандарта и его развития некоторые изменения иногда все же необходимы, но перенос программы между открытыми системами на порядки дешевле, чем между альтернативными), а также преемственности опыта пользователей.

[5] В 2002 г. «Белл Лабз», подразделение «Эй-Ти-энд-Ти», занимающееся исследованиями и разработками, с какой-то целью релицензировало свободно одну из старых версию «Юникс», System 7, имеющую сегодня лишь историческое значение.

Самым заметным эффектом существования этого стандарта стало эффективное разворачивание сетей Интернет в девяностых годах.

Вытеснение открытыми ОС альтернативных архитектур — медленный и сложный процесс. Хотя открытые системы сегодня существуют для вычислительных систем практически всех типов — от встроенных и карманных компьютеров до суперсерверов и мэйнфреймов — доля их в разных сегментах рынка неодинакова. Открытые ОС уверенно доминируют в серверном сегменте (особенно в сетевых приложениях), но распространены на ПК и рабочих станциях нижнего уровня пока менее широко, чем альтернативные.

Роль свободных операционных систем

В последние 5-7 лет рост пользовательской базы открытых ОС в основном происходит за счет распространения их свободных вариантов — «БСД» и «ГНУ/Линукс» — причем темп задает сейчас «ГНУ/Линукс».

Спецификой рынка свободных ОС, особенно, основанных на ядре «Линукс», является их существование в виде «популяции» параллельно развивающихся вариантов, называемых «дистрибутивами» (от англ. distributive kit — «распространяемый комплект»). Обычно дистрибутив включает в себя, помимо системных программ, большое количество прикладных программ и пакетов.

Несвободные открытые операционные системы

Свое значение сохраняют и несвободные открытые операционные системы, такие как «Солярис», «AIX», «Тру64 Юникс». Как правило, они применяются в сочетании с соответствующими аппаратными платформами, сопровождаются и поддерживаются производителями последних. Большинство свободных прикладных программ и пакетов перенесены или легко переносятся на такие ОС.

Альтернативные операционные системы

Большинство альтернативных (нестандартных) операционных систем вытеснены сегодня в ниши и не претендуют на универсальность. Ниже рассмотрены исключения.

«Майкрософт Уиндоуз НТ» («Майкрософт Уиндоуз 2000», «Майкрософт Уиндоуз Экс-Пи»). ОС этой серии позиционируются компанией «Майкрософт» как альтернатива стандартным (открытым) ОС и получили широкое распространение в сегменте однопользовательских настольных микрокомпьютеров («ПК») архитектуры x86/IA-32. «НТ» — дальний

потомок ОС «Ар-Эс-Экс» и «Ви-Эм-Эс» корпорации «Диджитал», вытесненных в свое время открытыми ОС с миникомпьютеров.

Для «Майкрософт Уиндоуз НТ» существуют специальные пакеты (Cygwin, UWIN, UNIX Services for Windows), эмулирующие системные вызовы, оболочку и утилиты открытых ОС на платформе этой ОС, так же, как и реализации стандартной графической платформы («Оконной системы Икс»). Кроме того, для многих программ и пакетов с графическим интерфейсом существуют «родные» переносы в «НТ» (т.е. с заменой стандартной графики на интерфейс Win32).

Обольщаться по этому поводу не стоит: опыт применения альтернативных ОС в большинстве случаев окажется более чем ущербным.

«МС-ДОС» («Майкрософт Уиндоуз 3.x, 9x, Ми»). Для ОС этой серии также существуют эмулирующие стандартную архитектуру пакеты и переносы популярных свободных программ, однако многие механизмы (например, распределение полномочий) здесь отсутствуют в принципе.

«Классическая» «МакОС». Под торговой маркой «МакОС» вплоть до версии «МакОС X» (исключительно) компания «Эппл Компьютерз» поставляла самодельные ОС для своих ПК «Эппл Макинтош», до сих пор находящиеся в эксплуатации. Под «классическую» «МакОС» перенесено лишь небольшое количество свободных программ.

Следует заметить, что сегодня «Эппл» под той же торговой маркой поставляет «МакОС X» — «бутерброд» из свободной открытой ОС «Дарвин» и проприетарных графических компонентов; под «Дарвин» существуют (или легко осуществимы) переносы большей части свободных программ и пакетов. Старые ПК «Макинтош», ресурсов которых недостаточно для запуска «МакОС X», могут быть модернизированы установкой на них «ГНУ/Линукс» соответствующей версии.

1.2 Практическая интеграция

Сильными техническими сторонами открытых ОС, выгодно отличающими их от любых альтернатив в плане практической интеграции (разворачивания, поддержания в работоспособном состоянии, непротиворечивого расширения и наращивания), являются:

- поддержка широкого спектра *аппаратных платформ* (самыми «всеядными» являются системы на основе ядра «Линукс»);

- поддержка различных *топологий* многопользовательских систем: от совокупности независимых рабочих станций до многотерминальных систем, в том числе, включающих компьютеры на разных аппаратных платформах.

В качестве существенной проблемы, сдерживающей стандартизацию во многих категориях систем, часто называют сложности с поддержкой многих устройств «потребительского» класса, ориентированных на архитектуры «IA-32» (IBM PC-совместимые) и «PowerPC» («Макинтоши»). Поставщики таких устройств нередко утаивают важные технические параметры и другую информацию, необходимую для разработки драйверов для своих видеокарт, модемов и пр.

Мы склонны считать это скорее проблемой в политике закупок, чем проблемой открытых ОС. Помимо всего прочего, владельцы таких изделий сплошь и рядом сталкиваются со сложностями и при смене версий ОС, поддерживаемых их поставщиками (производитель может прекратить свое существование или поддержку конкретного устройства или поссориться с разработчиком самой такой ОС). При приобретении оборудования, которое будет эксплуатироваться длительное время, всегда следует обращать внимание на доступность технической документации[6], а не только «пользовательских инструкций».

Поддержка различных топологий важна, поскольку позволяет найти применение, в том числе, и устаревающему (недостаточному по своим характеристикам для непосредственного исполнения нужных программ) оборудованию. Старые IBM PC-совместимые компьютеры или «Макинтоши» ранних моделей (в том числе, на основе процессоров от Motorola) могут включаться в сеть в качестве X-терминалов, и даже уже не выпускаемые алфавитно-цифровые терминалы можно присоединить к дешевому мультиплексору.

Топологии. Вообще говоря, топологии вычислительных систем (например, учебных классов) можно разделить на следующие категории.

1. *Совокупность автономных рабочих станций* (standalone WS; персо-

[6] Следует также отметить, что зачастую недоступность технической документации скрывает различного рода подлоги и дезинформацию потребителя. Например для дешевого сканера с пластиковой механикой и слабой оптикой могут заявляться недостижимые в этом классе разрешение и цветопередача; при этом производитель «забудет» указать (или укажет мелким шрифтом), что они относятся не к работе самого устройства, а к результату программной интерполяции, выполняемой «закрытым» драйвером устройства. То же относится к заявляемой поддержке неправдоподобного количества протоколов дешевыми модемами и т.п.

нальных компьютеров). Каждая станция содержит копию ОС и необходимых прикладных пакетов, пользовательские файлы хранятся также на локальном диске. Эта топология исключает затраты на разворачивание и поддержание локальной сети, однако без всякого преувеличения ее можно назвать «администраторским адом»: даже такая простая операция как обновление какого-либо программного пакета, требует «врачебного обхода» всех машин, которых может быть и десяток, и сотня. Кроме того, на пользователей ложится забота о воспроизведении своих данных в следующем сеансе (например, сохранением их на съемный носитель или жестким закреплением мест в классе за пользователями), что отнимает массу времени и приводит к неразберихе. Сильной ее стороной является высокая «живучесть»: выход из строя любого устройства означает в худшем случае утрату работоспособности одного рабочего места.

2. *Система рабочих станций без данных* (dataless WS). Рабочая станция без данных также содержит локальную копию ОС и прикладных пакетов, однако рабочие каталоги пользователей хранятся на файл-сервере локальной сети, объединяющей станцию. Та же сеть может использоваться и для централизованного администрирования систем, и встроенные административные средства открытых систем делают выполнение типовых процедур достаточно эффективным. Однако возрастают издержки за счет расходов на сеть и рисков выхода из строя сервера, что повлечет за собой неработоспособность целого класса или нескольких классов.

3. *Система бездисковых* (diskless) *рабочих станций*. Бездисковые рабочие станции подобны рабочим станциям без данных с тем дополнением, что не только данные, но и ОС и прикладные программы хранятся на файл-сервере. Сеть бездисковых станций предъявляет более высокие требования к производительности сервера и сети, чем сеть станций без данных, однако сами станции несколько дешевле, и администрировать такую систему еще проще.

4. *Система терминалов («тонких клиентов») и прикладного сервера*. В отличие от сети рабочих станций («толстых клиентов»), концепция «тонких клиентов» предполагает, что на рабочем месте исполняются только терминальные функции, а сами программы (и системные, и прикладные) запускаются на выделенном сервере. Эта топология предъявляет самые высокие требования к производительности и надежности сервера и сети, зато сами терминалы могут быть очень дешевыми и при этом надежными (нагрузка на процессор в терминале невысока, он может работать без принудительного охлаждения, диска в терминале нет в принципе, таким образом, X-терминал может быть собран вообще без движущихся частей).

(Интересной «вариацией на тему» является разработка Московского государственного индустриального университета «Горыныч», представляющая собой двух- или трехтерминальный комплекс, собранный на основе одного IBM PC-совместимого системного блока и работающий под управлением ОС «ГНУ/Линукс» (www.ctc.msiu.ru). Отличие от классической терминально-серверной архитектуры заключается в том, что дополнительные «терминалы» (видеоадаптеры и USB-клавиатуры и мыши) включаются не в локальную сеть, а в шину отдельного компьютера.).

5. *Смешанные топологии*. В одной и той же сети могут сосуществовать терминалы с рабочими станциями. Кроме того, машины с маргинальной производительностью могут использоваться в качестве рабочих станций при запуске «легких» (нетребовательных к ресурсам) программ и как терминалы — при исполнении «тяжелых» (ресурсоемких).

Существуют методики выбора оптимальной топологии исходя из профиля рабочих задач, конъюнктуры рынка и параметров уже эксплуатируемого оборудования. Скорее всего, в отличие от коммерческой среды, в реальных сегодняшних школьных условиях, включающих хроническое недофинансирование и спорадические, зависящие больше от вышестоящих органов, чем от самих школ, закупки и поставки оборудования, эти методики редко применимы. Однако сама гибкость топологии позволяет эффективным образом использовать и вновь приобретаемое, и уже имеющееся, и устаревающее оборудование.

Подбор оборудования. Все элементы перечисленных топологий, включая терминалы, рабочие станции и серверы, могут реализоваться на основе самого массового IBM PC-совместимого оборудования. Кроме того, можно задействовать менее распространенные PowerPC- и UltraSPARC-машины, а также старые «Макинтоши» на основе Motorola 68K (в качестве терминалов). В нашу школу вряд ли попадет что-то более экзотическое.

Стартовая мощность X-терминала — Intel 486DX-66 или эквивалентный по производительности процессор (хотя теория показывает возможность запуска X на процессорах i486SX и i386, нам не доводилось видеть, чтобы это приводило к удовлетворительным результатам) с 16 МБ памяти.

Рекомендуемый начальный уровень рабочих станций — PII600 или Celeron 800 со 128 МБ ОЗУ (многие программы могут нормально эксплуатироваться при гораздо меньшей производительности).

Для нормальной работы в бездисковой топологии сеть желательна (а при терминальной — необходима) как минимум 100-мегабитная.

Производительность серверов необходимо расчитывать; на сайтах произ-

водителей можно найти методики оценки.

Как уже отмечалось, нужно по возможности следить за доступностью технической документации на комплектующие (или, что то же самое, за их присутствием в списках аппаратной совместимости ведущих поставщиков «Линукс»- и «БСД»-систем).

Рекомендуемые параметры мониторов определяются Гигиеническими требованиями, принимаемыми Министерством образования; практика показывает, что их лучше соблюдать.

Еще один совет, который можно найти в рекомендациях по интеграции, устарел: одно время были популярны упрощенные «мыши» без средней кнопки, работать с которыми в современных графических средах не слишком удобно, и их рекомендовали избегать, однако сегодня большинство «мышей» снабжены колесом прокрутки, также действующим как средняя кнопка.

Администрирование. К задачам администрирования относится установка и обновление программ и пакетов, управление пользователями (распределение полномочий), резервное копирование данных, определение технической политики взаимодействия системы со внешними сетями и т.п.

Плохая новость заключается в том, что администратор — дорогая рабочая сила. Хорошая — в том, что при правильном проектировании и разворачивании системы последующее администрирование требует минимальных усилий.

При стабильном исполнении типовых задач один подготовленный специалист может осуществлять текущее администрирование *десятков независимых систем* (не считая систем, являющихся простыми копиями друг друга, например, одинаковых рабочих станций в учебном классе). Системный администратор на полный рабочий день может потребоваться разве что «набитой под завязку» техникой школе; в большинстве же случаев оптимальным решением будет приходящий специалист на неполный рабочий день или договор на выполнение административных функций со специализированной компанией (возможно, как часть договора на поставку систем или интеграцию).

«ГНУ/Линукс» и «БСД». Существуют два распространенных мифа относительно дистрибутивов, основанных на ядре «Линукс», с одной стороны, и восходящих к «БСД» — с другой.

Первый из них заключается в том, что «“Линукс” — для клиентов, “БСД” — для серверов». Это неправда не только в части «Линукс»-си-

стем, «прекрасно чувствующих себя» на серверах, но и в отношении «БСД», совершенно нормально приспособленной для «настольного» применения (по крайней мере, в ипостаси FreeBSD).

Второй сводится к тому, что «“Линукс” под GPL, “БСД” под “БСД”-лицензией». На самом деле, как «БСД»-, так и «ГНУ/Линукс»-дистрибутивы включают в себя множество пакетов под *различными* свободными лицензиями. При этом в прикладной части состав пакетов практически одинаков, да и в системной и инструментальной пересекается более чем наполовину.

Кроме этих сказок, существует масса поводов для препирательств между «фанами» этих двух концептуальных направлений свободного ПО, в каковых препирательствах мы никому не рекомендуем принимать участия. Рыночная конъюнктура сегодня такова, что в «Линукс» вкладывается больше средств, она шире используется, специфические навыки работы с ней более распространены, литература доступнее.

Реальное основное различие (помимо технических тонкостей) заключается в том, что «БСД»-сообщества более склонны к централизации, а «ГНУ/Линукс» — к диверсификации. Свободные системы «БСД» на сегодня существуют в четырех вариантах: FreeBSD, NetBSD, OpenBSD и Darwin[7]. Число известных *дистрибутивов* «ГНУ/Линукс» превышает сотню[8].

Большая часть нижеприведенной информации для перспективных пользователей «БСД» не нужна.

Дистрибутивы. «Дистрибуция» означает «распространение» или «распределение». В товарной экономике «дистрибутор» — фирма, занятая передачей продукции от производителя розничной торговле, которая может играть сколь угодно активную роль в маркетинге, но на товар прямого влияния не оказывает.

В программном обеспечении «разработчик дистрибутива» — важное *активное* звено в процессах разработки, использования и применения программ. Его роль не менее важна, чем роль разработчиков отдельных программ.

Сила свободного ПО не только в том, что любая программа может свободно использоваться как таковая (включая ее модификацию и распространение) (система на основе «Линукс» может быть собрана полностью

[7] Единым «концентратором» ссылок на сайты систем «БСД» является сайт www.bsd.org.

[8] Ссылки на сайты большинства дистрибутивов «ГНУ/Линукс» можно найти на www.distrowatch.com.

«вручную»[9], что весьма рекомендуется в курсе подготовки продвинутых администраторов и системных программистов (а также всем серьезно интересующимся тем, как системы устроены изнутри), но совершенно излишня для всех остальных), но и в том, что идеология «интеллектуальной собственности» исключена как помеха на пути объединения программ в системы.

Таким образом, конечный пользователь может получать все (или почти все) необходимые ему программы — не только составляющие операционную систему или среду, но и прикладные — из одних рук, готовыми к совместному применению, если найдется разработчик дистрибутива с близкими целями.

Первые дистрибутивы ОС на основе ядра «Линукс» («SLS» и «Slackware»; последний выпускается и сегодня, хотя больше популярен среди профессиональных администраторов, чем среди других категорий пользователей) появились десять лет назад, когда это ядро стало стабилизироваться и находить своих пользователей вне академического сообщества и сообщества системных программистов. К настоящему времени существует более сотни дистрибутивов, получивших более или менее широкую известность, и сама множественность дистрибутивов стала важной чертой и отличительным признаком ОС на ядре «Линукс».

Не стоит надеяться, что появится один «самый правильный» дистрибутив; скорее, уменьшение конкуренции в разработке дистрибутивов знаменовало бы собой кризис развития этих систем.

Когда-то бытовало чрезмерно упрощенное представление о цикле программного обеспечения, так называемая «каскадная модель». Позже стало понятно, что, коль скоро речь идет о сложных программах, а тем более о системах программ, «каскадная» идеализация неадекватна, и реальный процесс гораздо лучше описывается в терминах «спирали»; с этим связан и известный афоризм (кажется, восходящий еще к пятидесятым годам): «У разработки больших систем не бывает завершения, бывают только релизы».

Функция разработчика дистрибутива заключается не в том, чтобы просто «собрать программы в кучу», красиво упаковать и продать, а, скорее, в том, чтобы обеспечить эффективную коммуникацию между, с одной стороны, авторами программ (и прочих произведений, входящих в дистрибутивы) и, с другой, конечными пользователями, а также самих конечных пользователей. Упрощенно структуру дистрибутива можно предста-

[9] Сборка системы на базе «Линукс» «с нуля» описана в книге Linux from Scratch (www.linuxfromscratch.org). Русский перевод можно найти на http://vnc.org.ua/lfsbook/.

вить как коллектив *мэйнтейнеров* — лиц, отвечающих за поддержание *пакетов* (единиц в системе) в *хранилище* в актуальном состоянии и системной целостности в соответствии с некоторой *концепцией* в интересах *сообщества пользователей*. На границе команды разработки и пользовательского сообщества разворачиваются сервисы, которые могут служить ресурсной подпиткой разработки дистрибутива.

Поставка очередного выпуска дистрибутива (на дисках или через Сеть) — лишь один шаг в этой коммуникации[10]. Выбор дистрибутива со стороны конечного пользователя — больше, чем просто выбор товара; по сути дела это выбор «узла присоединения к инфраструктуре». Не все аспекты функционирования ОС, а тем более, прикладных платформ и пакетов, стандартизованы, поэтому часть знаний и навыков администратора специфичны для дистрибутива.

Однако еще важнее, что дистрибутив определяет ближайший круг общения и стиль общения, в которое неизбежно вступает конечный пользователь. Большая часть литературы, в которой различные дистрибутивы сравниваются между собой[11], сосредоточена на технических различиях, однако, гуманитарные и «коммунитарные» (относящиеся к жизни сообщества) аспекты представляются более важными.

Наилучший способ выбора «своего» дистрибутива заключается, видимо, в том, чтобы познакомиться с информацией, приводимой на сайте разработчика или издателя, почитать архивы списков рассылки или форумов (обращая внимание на стиль общения, отношение к новым и менее опытным участникам обсуждений), найти нескольких пользователей с определенным опытом в близкой к вашей сфере приложения и пообщаться с ними напрямую. Полезным может оказаться набор критериев, изложенный в следующих двух разделах.

Критерии для выбора дистрибутива. *Общая пользовательская аудитория и срок существования дистрибутива.* Общее правило: чем более широка аудитория, тем быстрее исправляются ошибки и тем проще найти помощь в решении той или иной задачи. По большому счету, это самый важный параметр дистрибутива, хотя и с нюансами, касающимися специфической сферы применения и языковой среды (см. следующие

[10] Следует также понимать, что торговля коробками, как правило, не является основным бизнесом компании или команды, зарабатывающей чаще всего заказной разработкой и/или продажей услуг. Продажа коробок — низкорентабельные издательские проекты, в лучшем случае окупающие управление самим изданием.

[11] Из русских публикаций в особенности отметим цикл статей Алексея Федорчука «Юникс для всех» (http://linuxshop.ru/unix4all/) и книгу: Алексей Федорчук, «Офис, графика, Web в Linux». — СПб.: BHV, 2001.

два пункта).

Пользовательская аудитория в сфере применения. Чем больше конечных пользователей применяют дистрибутив в вашей сфере деятельности (например, в школьной практике), тем проще найти помощь в решении специфических для этой сферы задач.

Пользовательская аудитория в языковой среде. Воспользоваться преимуществами развитого пользовательского сообщества может помешать языковой барьер. Важным, таким образом, является наличие достаточного количества пользователей с родным (или хорошо знакомым) для вас языком. Ее наличие (и присутствие в команде людей, пользующихся или хотя бы знакомых с родным для вас языком) также важно в плане отсутствия проблем (или легкости их решения) с локализацией.

Местная (в географическом смысле) пользовательская аудитория. Распространение электронной почты и других приложений Интернет сделало «местный» фактор менее значимым, но не отменило его. Если рядом с вами (особенно в учебном заведении, расположенном в том же городе) накоплен серьезный опыт пользования тем или иным дистрибутивом, это серьезный довод в пользу его выбора.

Документированность. Значение имеет также документированность особенностей дистрибутива и наличие свежих переводов документации.

Консервативность/склонность к экспериментированию. Некоторые составители (такие, как «Дебиан») более склонны к консервативным, проверенным временем решениям, а некоторые (например, «РедХэт») более смелы в экспериментах. Что вам важнее: иметь самые свежие версии программ или меньшую вероятность ошибки? — Не торопитесь с ответом, для него нужны опыт и приходящая только с опытом мудрость, и ответ не будет однозначным.

Спектр поддерживаемого оборудования (HCL). Наличие формального списка поддерживаемого оборудования или (hardware compatibility list, HCL) — серьезный довод в пользу дистрибутива, особенно, если ваш парк оборудования комплектуется на случайной основе. Если у вас постоянный поставщик оборудования и он сотрудничает с кем-либо из составителей дистрибутивов — это еще более серьезный довод в пользу последнего.

Поддержка необходимых программ (состав дистрибутива в прикладной части). Как ни странно, это гораздо менее значимый параметр, чем перечисленные выше. Легче самому дополнить несколькими программами дистрибутив, в остальном удовлетворяющий вышеперечисленным критериям.

Критичность несвободных компонентов. Большинство популярных дистрибутивов (кроме Debian GNU/Linux), по крайней мере, в наиболее полном варианте, включает, помимо свободных, и несвободные программы. Правильная политика заключается в том, чтобы отделять свободное от несвободного и исключать зависимость основной функциональности от несвободных программ (например, в текущем (2.2) выпуске дистрибутива ALT Linux Master все несвободное сосредоточено на девятом диске). Важным является также свободное лицензирование (или, по крайней мере, лицензия на свободное распространение в неизменном виде) документации, каковой политики придерживаются не все разработчики.

Информационная и ценовая политика разработчика. При прочих равных, преимущественно стоит входить в отношения с поставщиком, придерживающимся полной прозрачности разработки дистрибутивов. Технически это означает свободный доступ (на чтение) к дереву разработки через cvs или ftp или, по крайней мере, простую регистрационную процедуру для получения такого доступа. Разработчики, закрывающие процесс и лишь периодически сбрасывающие его результаты в релизы, скорее всего, готовят сюрпризы своим пользователям, и редко такие сюрпризы оказываются приятными

Цена изданий на дисках обычно не играет большого значения (поскольку приобретается один-два комплекта на десятки компьютеров) и варьирует незначительно из-за конкуренции между промышленно тиражированными дистрибутивами и альтернативой самостоятельного переписывания с одолженных дисков или через Интернет. Цена «компактного» (один-три диска плюс брошюрка) дистрибутива в России — порядка 200-300 р., «обширного» (шесть-десять CD плюс несколько томов документации) — от одной до трех тыс. р. Публикация дистрибутивов на DVD, возможно, уничтожит феномен «компактного» малодискового дистрибутива и приведет к дальнейшему снижению цен[12].

Альтернативным способом получения дистрибутива является его полная или попакетная загрузка через Сеть (на сайтах разработчиков практически всегда они есть), что может оказаться дороже, но оперативнее приобретения дисков. Чаще всего пользователи сочетают приобретение комплектов дисков по мере выхода очередных релизов и загрузку по Сети исправлений и обновлений в периоды между релизами.

Технические параметры дистрибутивов. *Бинарная установка или установка из исходников?* В сообществе «БСД» в качестве штатной про-

[12] Пока в России существует один прецедент издания дистрибутива на DVD — Debian GNU/Linux 3.0 в редакции ALT Linux.

цедуры установки принято «портирование», т.е. автоматизированная компиляция и сборка пакетов для целевой машины из исходников. В сообществе «ГНУ/Линукс» в качестве штатной процедуры установки принята распаковка бинарных (прекомпилированных) пакетов, и до недавнего времени (появления так называемых source-based дистрибутивов) все дистрибутивы технически поддерживали именно этот способ установки (хотя, разумеется, администратор мог и пересобрать любой пакет).

Преимущество установки из исходников — оптимизация под конкретную машину и меньший объем пакетов. Преимущество бинарной установки — более высокая ее скорость. Следует иметь в виду, что сборка некоторых пакетов на компьютере персонального класса может длиться более десяти часов, и пересборка всех часто использующихся программ может занять несколько суток.

Программа установки, управление пакетами и утилиты настройки. Как уже замечено, большинство дистрибутивов «ГНУ/Линукс», включая самые популярные, предусматривают установку с первоначальной настройкой и обновление с использованием прекомпилированных программ, собранных в *пакеты*. Пакет, который может включать одну или более программ, файлы конфигурации, документацию и т.п. является минимальной единицей установки или обновления штатными для дистрибутива средствами. В отдельные пакеты составителями собираются также исходные коды, соответствующие прекомпилированному пакету. Процедуры установки, удаления, обновления пакетов называются *управлением пакетами*.

Стандарта на пакетирование и управление пакетами не существует. Распространение получили три формата пакетов: *rpm* (впервые появившийся в дистрибутиве RedHat и применяемый сегодня большинством составителей дистрибутивов), *deb* (применяемый Debian) и *tgz* (применяемый Slackware). На формат пакета завязаны программа установки и управления пакетами («rpm» для rpm, «setup» для tgz и «dpkg» для deb), способная отслеживать *зависимости* между пакетами (ситуации, когда для нормальной работы программы из одного пакета требуется программа (возможно, определенная версия) из другого пакета, или, наоборот, когда программы из разных пакетов являются взаимоисключающими в рамках одной системы).

В последние годы развиваются усовершенствованные средства управления пакетами, позволяющие преодолевать некоторые ограничения, свойственные «rpm» и «dpkg» (в частности, отслеживать ситуацию смены имени (в отличие от номера версии) пакета). В качестве примеров таких средств можно назвать «apt» (дистрибутивы Debian, ALT Linux и

Conectiva) и «yum» (дистрибутив ASPLinux).

Различные системы пакетирования и управления пакетами примерно равномощны, но используют отличающийся синтаксис. Начинающему администратору проще перейти от администрирования одной системы к другой, если в них применяется одна и та же программа управления пакетами, чем к системе с другой программой управления пакетами.

Одно время среди составителей дистрибутивов было модно создавать различные «утилиты настройки», обычно с графическими интерфейсами, позволяющие выполнять некоторые административные задачи альтернативным способом. По большей части этот опыт следует признать неудачным, так как попытки администрировать систему такими утилитами обычно рано или поздно приводят ее в неуправляемое или даже неработоспособное состояние.

Аппаратные платформы. Наконец, следует обратить внимание на аппаратные платформы, на которые ориентирован дистрибутив. Более половины существующих дистрибутивов ориентированы исключительно на IA-32 (IBM PC-совместимые компьютеры), большинство остальных поддерживает две-три аппаратные платформы, а «Дебиан» — целых десять, включая достаточно экзотический. Поддержка даже ненужного вам «железа» при прочих равных, тем не менее, является признаком зрелости дистрибутива.

Обзор самых популярных дистрибутивов. Скорее всего, реальный выбор будет идти между дистрибутивами, либо входящими в «высшую лигу» по популярности в мире (Debian, RedHat, Mandrake и SuSE), либо разрабатывающимися в России (ALT Linux и ASPLinux)[13].

ALT Linux разрабатывается международным коллективом ALT Linux Team и публикуется одноименной московской компанией. Сильной стороной ALT является достаточно структурированное русскоязычное сообщество пользователей (включающее списки рассылки, справочные серверы, деловые связи ALT Linux и пр.), слабой — отсутствие на сегодня сколько-нибудь серьезной аппаратной базы для тестирования (и, как следствие, отсутствие формального HCL). ALT неплохо документирован[14]. Дистрибутив выходит с 2001 г., хотя следует учитывать и ранний опыт той же команды, выпускавшей до того русскую версию Linux-

[13] Реально команды разработчиков этих двух дистрибутивов международные, как и большинства дистрибутивов, считающихся «зарубежными» по месту публикации.

[14] Документация на издаваемые в России дистрибутивы ALT Linux и ASPLinux может быть найдена на http://docs.altlinux.ru и www.asplinux.ru/ru/docs, соответственно.

Mandrake

ASPLinux — в значительной мере остается производным от Red Hat, причем заявляется бинарная и аппаратная совместимость с последним. ASPLinux более консервативен, чем RedHat, поставка его включает неплохую документацию на русском. Сохраняя лучшие черты RedHat, он, в большинстве случае является более интересным, чем RedHat, вариантом, по крайней мере, для начинающего российского пользователя. Выпускается с 2001 г., хотя следует иметь в виду и более ранний опыт влившейся в коллектив ASPLinux украинской команды Black Cat Linux.

Debian GNU/Linux. «Образцово-показательный» Debian — единственный из популярных дистрибутивов, принципиально разрабатываемый в некоммерческих рамках. Официальные дистрибутивы Debian содержат пакеты исключительно со свободным софтом. У Debian наиболее формализованные и прозрачные правила взаимодействия в сообществе, причем многие процедуры технологизированы. Debian поддерживает самый широкий спектр аппаратных платформ. «Коробочные» продажи этого дистрибутива невелики, но он является самым популярным среди разработчиков ПО (около половины активных раработчиков свободного ПО пользуются именно им). В России Debian GNU/Linux издается московской компанией ALT Linux, являющейся также разработчиком собственного одноименного дистрибутива (см. выше). Debian GNU/Linux существует с 1993 г.

Linux-Mandrake. Этот дистрибутив, выпускающийся французской компанией MandrakeSoft, «отпочковался» в свое время (1996 г.) от RedHat и в какой-то момент даже догнал его по количеству коробочных продаж. Парадоксальным образом достаточно высокое качество разработки не помешало компании попасть в затруднительное положение, но компания продолжает выпуск своего дистрибутива, текущая версия которого оценивается пользователями как весьма удачная и сохраняет популярность.

RedHat Linux. Этот дистрибутив, выпускаемый американской компанией RedHat Software, наиболее популярен в мире среди непрофессиональных пользователей «ГНУ/Линукс», и является лидером по «коробочным» и OEM-продажам. Для RedHat собирается наибольшее количество несвободного софта, что важно для пользователей приложений, не имеющих удовлетворительных свободных реализаций. RedHat Linux существует с 1994 г. В России выходит «кириллическое издание» RedHat, издающееся питерской компанией Linux-ink.

SuSE Linux. С вхождением SuSE в группу аутсайдеров «ГНУ/Линукс»-дистрибуции UnitedLinux, стремящихся за счет объединения усилий отвоевать часть рыночной доли RedHat, перспективы дистри-

бутивов SuSE, имеющих, несомненно, свои преимущества, становятся весьма туманными, хотя текущее состояние проекта достаточно стабильное[15].

Среди неназванных неплохие шансы стать по-настоящему популярным имеет также относительно новый дистрибутив *Gentoo*. В отличие от перечисленных — это дистрибутив, предполагающий установку из исходников (source-based, см. выше).

Любой из перечисленных дистрибутивов (также, как и многие из неперечисленных) может быть использован для знакомства с «ГНУ/Линукс» примерно с равным успехом. Любой из них, даже в минимальной (одно-трехдисковой) поставке содержит почти все из описанных в книге пакетов.

На случай *прочих равных* (и настоятельной необходимости *немедленно* выбрать дистрибутив для первого знакомства с «ГНУ/Линукс») лектор рискнет порекомендовать Debian GNU/Linux. Категорически *не* рекомендуются (в любой ситуации) Lindows и Caldera OpenLinux, а также дистрибутивы, разработка которых прекращена (Corel Linux, Storm Linux, Stampede, HP Secure Linux).

Рекомендуется также избегать «экзотических» дистрибутивов, хотя, как отмечалось выше, такие факторы, как, например, наличие «под боком» организации или группы с большим опытом в использовании даже относительно слаборaспространенного (в мире или стране) дистрибутива, могут перевесить соображения популярности.

1.3 Почему командная строка?

Существует широко распространенное заблуждение, согласно которому графический интерфейс якобы представляет собой высшую и последнюю стадию развития пользовательских интерфейсов, а «командная строка» — это нечто примитивное и малополезное, если не устаревшее.

На самом деле, эта «иерархия» отражает не что иное, как порядок, в котором с двумя основными метафорами организации пользовательских интерфейсов познакомились пользователи персональных компьютеров. Вплоть до начала девяностых ПК были простыми и маломощными, поддержка графики и возможность комфортной работы с оконными графическими системами появилась на них недавно (даже в масштабе стремительно развивающейся информационно-технологической отрасли).

[15] Во время подготовки книги к печати было объявлено о планах поглощения компании «SuSE» компанией «Новелл».

Однако за годы до этого графические интерфейсы уже широко применялись на рабочих станциях — конечно, более дорогом, редком и специализированном оборудовании. Если быть точными, то обе концепции в их более или менее современном виде сформировались примерно в одно время.

В 1967-68 гг. Дуг Энгельбарт представил прототип т.н. WIMP-интерфейса, т.е. интерфейса, использующего понятия окон (windows), пиктограмм (icons), меню (menus) и указателей (pointers), являющихся ключевыми и для сегодняшних графических пользовательских сред.

В 1968-69 гг. Кен Томсон и Деннис Ричи представили первый релиз ОС UNIX, по сути, явившейся прототипом артикулированной системы современных понятий практической информатики, таких, как процессы и файлы, и содержащей непротиворечивый, логичный и лаконичный язык работы с соответствующими им сущностями, который спустя полтора десятка лет стал стандартным пользовательским интерфейсом ОС.

Разумеется, как одна, так и вторая метафорические системы появились не на голом месте, и их родословную можно проследить вплоть до истоков вычислительной техники: до коммуникационной панели первых компьютеров, как вещи твердой и весьма графической (или, во всяком случае, живописной) — это конец сороковых, и до языков управления заданиями в первых программных планировщиках, загрузчиках и ОС — это середина пятидесятых, соответственно. Отметим, что идея управления компьютером как «прибором» — с помощью органов управления (неважно, «в железе» или нарисованных на экране) старше идеи «диалога» при помощи слов.

При этом для большинства пользователей знакомство с «командной строкой» до недавнего времени отягощалось радикальными «кастрацией» и изменением, которым язык стандартной оболочки ОС был подвергнут при разработке упрощенных ОС для ПК (таких, как «СиПи/М», «МС-ДОС» («ПиСи-ДОС») и клонов последней). Ограничиваясь знакомством со средствами «МС-ДОС» и ее командных файлов, о метафоре «командной строки» можно составить лишь весьма превратное и убогое представление.

Для людей старорежимных, начиная от возраста автора (ему чуть за 30) и старше, компьютер удивителен и страшен своей уникальной способностью к символической активности. Мы выросли в мире «глупых» вещей, пассивных или проявляющих свою активность чисто механически (как автомобиль, который может быстро довезти, или же раздавить, если вовремя не увернешься), и неспособных к диалогу в бытовом окружении. К диалогу способны были вещи культуры (книга, картина, симфония) — но

опосредованно, в отдельных, отведенных для этого местах, в назначенное время и при тщательном отборе собеседников.

Возможно, пирожное и говорило, по Курту Левину, на своем кондитерском языке «съешь меня», но ничего подобного продемонстрированной нам недавно упаковке, говорящей «купи меня» уже обычным человеческим голосом (причем, с учетом физиоантропометрических данных приблизившегося покупателя), не было. Вещи были «глупыми», сколь бы изощренный ум своих создателей они не воплощали, а умными были только люди.

Автор до сих пор находится под влиянием первого своего опыта в десятилетнем возрасте общения с компьютером, что-то ему ответившим в ответ на введенную строку. Ответившим осмысленно. (Подробностей он не помнит, но, скорее всего, первым ответом было указание на то, что команда содержит синтаксическую ошибку.) Слово *страшен* выше — не случайно, но, в конце концов, компьютеры были тогда надежно заперты в своих вычислительных центрах.

Роль клетки для этого зверя для очень многих сегодня играет гладкая поверхность «графических интерфейсов», скрывающая диалог. Компьютер, прикрытый таким образом, уже не демонстрирует пугающую символическую наготу. Он реагирует на нажатую кнопку с картинкой — можно успокаивать себя, что это такая хитрая машинка: пылесос после нажатия кнопки сосет, а компьютер — печатает, связано все чисто механически, и никакого человечка, который с изнанки интерфейса посмотрел, куда же ты нажал и определил, что сделать с картинкой, нет.

На самом деле призрак маленьких человечков есть — за кулисами гладкой разрисованной поверхности идет обмен такими же — mutatis mutandis — сообщениями, что озадачили автора, когда он был маленьким. От нынешних маленьких их хотят попрятать, как электрический ток за изоляцией, чтобы не стукнуло. Но в шкафу каждого компьютера, будь он трижды персональный, спрятан скелет искусственного разума, при всей условности последнего выражения.

Существует влиятельная тенденция в современных гуманитарных дисциплинах, получившая распространение в основном в англоязычном мире и связанная с остроумными теориями канадского ученого Маршалла Мак-Люэна, утверждавшего, что письменная («визуальная») культура связана с гипертрофией аналитических функций человеческого интеллекта на определенных стадиях его развития, и что ее сменяет культура «аудиальная», направленная на синтез целого путем рассеивания внимания, и возврат к доиндивидуалистическому, общинному сознанию, от полиса и урба — к «глобальной деревне».

С этой сменой он связывал рост популярности телевидения (и относительное снижение популярности чтения как досуга) и «клип-культуры», а его последователи указывают и на несомненный рост в девяностые популярности «графических пользовательских интерфейсов» компьютеров, причем не только в быту, но и в деловом окружении.

Мак-Льюэн, впрочем, в своем «зондировании» (как он именовал свое творчество, весьма последовательно отказываясь называть свои позднейшие произведения «текстами») был достаточно ироничен, что в гораздо меньшей степени наблюдается в творчестве его последователей.

Разумеется, каждый имеет право, хотя оно особо и не оговорено в Конституции, на персональные фобии, и даже право постфактум рационализировать их в теориях. Однако мы полагаем, что никому не должно быть позволено транслировать эти фобии другим, тем менее — подрастающему поколению.

Компьютеры (и новая генерация «умных» приборов, машин и механизмов, включая промышленные, канцелярские и бытовые) отличаются от всего, созданного человеком ранее, способностью непосредственно манипулировать символами, воспринимать символы и сообщать символы человеку, т.е., в некотором роде вести с ним диалог.

Стоит ли называть это свойство компьютеров и программ «искусственным интеллектом» — отдельный вопрос, но сам факт, по нашему мнению, должен занять одно из основных мест в содержании учебного предмета информатики. В этом смысле, наблюдаемая тенденция к вытеснению программного инструментария, являющего это свойство в самой методически и дидактически откровенной форме, из школьных курсов кажется нам крайне неприятной и нуждающейся в коррекции.

Мы с энтузиазмом относимся к применению графических интерфейсов, как в традиционной сфере компьютерной графики, так и в новых, перспективных приложениях. Картинка часто стоит сотни слов, а энергичный жест способен выразить простую мысль быстро и однозначно.

Однако мысль о том, что пиктограммами и жестами можно заменить полноценный язык, напоминает нам лишь одну из гениальных идей академии наук в Лагадо, описанной Джонатаном Свифтом в «Третьем путешествии Гулливера»:

> «А так как слова суть только названия вещей, то автор проекта высказывает предложение, что для нас будет гораздо удобнее носить при себе вещи, необходимые для выражения наших мыслей и желаний».

Глядя на визуально-дизайнерское произведение очередного их последо-

вателя, лишь удивляешься: из какого мешка он достал значок, выражающий идею: «Вход с жующими мороженое несовершеннолетними леопардами в темное время суток запрещен»? И почему он думает, что этот значок интуитивно понятен?

> «Мне часто случалось видеть двух таких мудрецов, изнемогавших под тяжестью ноши ... При встрече на улице они снимали с плеч мешки, открывали их и, достав оттуда необходимые вещи, беседовали в течение часа».

В Лагадо попытка мудрецов осуществить масштабное внедрение своего изобретения закончилась тем, что:

> «Женщины, войдя в стачку с невежественной чернью ... пригрозили поднять восстание, требуя, чтобы языку их была предоставлена полная воля ... так простой народ постоянно оказывается непримиримым врагом науки!»

График нагляден, лишь пока цифры на нем можно разглядеть, а пиктограммы осмысленны только при выборе из немногих вариантов (даже сотню дорожных знаков выучить уже непросто). Для артикулированного и гибкого выражения идей (далеко не все из которых можно нарисовать) и их связи человечество выработало такой инструмент, как языки (естественные и формальные), и замены им пока не предвидится.

1.4 Сеанс работы и команды

Интерфейс оператора ОС представляет собой интерпретирующий язык, конструкции которого могут выполняться непосредственно при их вводе оператором (интерактивный режим) или из файлов (режим выполнения сценариев).

Как и любой язык, язык оболочки команд и ОС нуждается в изучении. Как и при изучении любого языка — естественного или искусственного — первые шаги достаточно трудны. Однако изучить его легче, чем любую развитую систему научной нотации (математической или химической) даже в объеме средней школы, и неизмеримо легче, чем иностранный язык. Вводные курсы в открытые ОС успешно прошли миллионы человек.

Загрузка и разгрузка системы

Изначальный запуск программ, составляющих операционную систему, называется *загрузкой* ОС, их завершение — ее *разгрузкой*, а загрузка сразу вслед за разгрузкой — *перезагрузкой*.

Эти действия относятся к административным, то есть для их выполнения

нужно обладать соответствующими полномочиями.

Многопользовательские машины и серверы разгружаются обычно лишь перед ремонтом или модернизацией оборудования, а перезагружаются — после обновления ядра, важных системных программ или библиотек.

Однако на настольных, портативных и мобильных машинах выполнение этих процедур в большинстве случаев делегируется всем пользователям, и нет необходимости регистрироваться в системе в качестве администратора (и даже иметь такую возможность, то есть знать пароль администратора), чтобы выполнить ее разгрузку или перезагрузку. Такие машины часто выключаются по окончании рабочего дня.

На микрокомпьютерах для загрузки системы обычно не требуется выполнять каких-то специальных действий. При включении машины автоматически выполняется следующая последовательность процедур.

1. Загружается в память и начинает исполняться *вшитый загрузчик*. Это миниатюрная программа, располагающаяся в энергонезависимой памяти (ППЗУ или флэш) на системной плате, которая пытается загрузить следующую программу-загрузчик с устройства, указанного в настройках (также располагающихся в энергонезависимой памяти). Такими устройствами могут быть сетевой адаптер, фиксированный (жесткий) диск или привод съемного носителя информации (дискеты, CD, DVD или ZIP-диска). Бездисковые машины загружаются с сетевого адаптера (для этого в локальной сети должен присутствовать сервер, предоставляющий по запросу загрузчик). Автономные машины загружаются обычно с фиксированного диска. Со съемного носителя загружаются автономные машины при установке на них новой ОС или проведении восстановительных работ.

2. *Загрузчик ОС* (LILO, GRUB или другой) загружает ядро операционной системы. Загрузчик может быть настроен на выдачу перед загрузкой *меню*, в котором перечислены различные варианты загрузки или даже различные ОС, установленные на фиксированных дисках машины или в локальной сети.

3. После этого управление передается загруженному *ядру* операционной системы, которое выполняет запрограммированную последовательность действий, включая собственное конфигурирование (подгрузку модулей), и запуск корневого процесса, запускающего системные сервисы и программы и т.д.

Такая (кажущаяся сложной) трехзвенная последовательность загрузки обеспечивает необходимую гибкость в администрировании машины. Для оператора же она сводится в большинстве случаев к тому, что неко-

торое время машина активно мигает лампочками сетевого контроллера и/или привода фиксированного диска, а на ее консоль (терминал, подключенный непосредственно к машине) выдается серия диагностических сообщений.

Показателем успешной загрузки системы является вывод на консоль (и на другие терминалы, если таковые имеются) *приглашения к регистрации*: как правило оно содержит логотип ОС и/или организации, сведения о локальной системе (включая ее имя, тип и тактовую частоту процессора, объем оперативной памяти) и строку «login:».

Некоторые машины настраиваются на автоматический запуск в ходе загрузки графической оконной системы «Икс». В этом случае (а также при регистрации на икс-терминале) приглашение к регистрации будет иметь вид *окна* с полями «login» (регистрационное имя) и «password» (пароль).

Начиная с момента передачи управления ядру (начала загрузки собственно ОС) машину не следует выключать, не выполнив процедуру *разгрузки* ОС. Разгрузка ОС и выключение машины описаны в документации. В большинстве случаев самым быстрым (хотя и не самым изящным) способом будет одновременное нажатие клавиш Control-Alt-Del и физическое выключение машины при появлении сообщения и/или логотипа вшитого загрузчика (БИОС).

Вход в систему и выход из системы

Для того, чтобы взаимодействовать с компьютером, нужно *зарегистрироваться* в ОС. Регистрируясь («входя» в систему) , вы сообщаете системе, что на определенном терминале (реальном или виртуальном) с ней работает определенный пользователь. Система полагает, что пользователь продолжает работу, пока он явным образом не *разрегистрируется* (не «выйдет» из системы).

Время от входа до выхода из системы называют *сеансом работы* в ней. Сеансом называют также совокупность команд, поданных пользователем за это время.

Программа регистрации запрашивает *регистрационное имя* (login) и *пароль* (password). Регистрационное имя — это имя, под которым пользователь занесен в список пользователей, поддерживаемый в системе. Оно уникально для машины и может быть известно другим пользователям. Пароль — это личный секрет пользователя, он не должен быть известен больше никому.

Пароли нужны для того, чтобы предотвратить регистрацию в системе под чужим именем. На машине, все пользователи которой абсолютно до-

веряют друг другу, можно устанавливать пустой пароль, тогда для регистрации в системе достаточно будет ввести регистрационное имя. При этом, если машина включена в сеть, таким пользователям должна быть запрещена «удаленная» регистрация (регистрация с других машин).

Регистрационные имена не соответствуют однозначно физическим лицам. Во-первых, целый ряд имен не допускают регистрации под собою вовсе; «от их имени» исполняются системные программы и сервисы.

Во-вторых, одно и то же лицо может регистрироваться на машине «в разных ролях». Если вы сами администрируете систему, то для выполнения административных задач вы регистрируетесь под именем главного пользователя (root). Для решения прикладных задач (чтения почты, получения информации из WWW, редактирования текстов, программирования, чего угодно) вы регистрируетесь под именем «обычного пользователя», учетную запись которого для себя создали.

Выполнять прикладные задачи, зарегистрировавшись под именем главного пользователя (root), нельзя.

Если вы легальный пользователь машины, администрируемой другим лицом, администратор сообщит вам (или предложит выбрать) ваше регистрационное имя и пароль.

Если вы сами администрируете систему, то, скорее всего, уже создали учетную запись для себя при установке системы.

Далее мы обсуждаем пользовательницу Алису, имеющую учетную запись alice на машине wonderland и знающую свой пароль (мы его не знаем).

```
login: alice
password:
[alice@wonderland alice]$
```

Рис. 1-1

На алфавитно-цифровом терминале Алиса регистрируется, вводя в ответ на приглашение «login» имя alice и завершая ввод нажатием клавиши Enter, и, затем, вводя свой пароль (пароль при вводе не отображается) и завершая его ввод нажатием той же клавиши (Рис. 1-1).

После этого программа (это уже не программа регистрации, а программа-оболочка операционной системы) выводит *подсказку* («приглашение»).

Достаточно типичная подсказка изображена выше; она состоит из регистрационного имени и имени машины, разделенных знаком «@» (читается «эт»), за которыми следует имя текущего каталога. Все это заключено в квадратные скобки, за которыми следует символ подсказки «$».

Возможно, на вашей машине строка подсказки выводится в другом формате. Как и почти всё в открытых системах, формат строки подсказки легко настраивается главным пользователем (для всех пользователей машины) или каждым пользователем для себя.

Чтобы получить строку приглашения, зарегистрировавшись в системе с автоматическим запуском оконной системы «Икс» или на икс-терминале, следует запустить программу эмуляции алфавитно-цифрового терминала (например, поставляемую вместе с «Икс» «xterm»). Способ запуска зависит от конкретного оконного менеджера и/или графической среды, его следует узнать из документации или от администратора.

Выдача подсказки означает, что оболочка готова принимать *команды*. Команда — это строка, которая будет принята к исполнению после того, как пользователь, введя ее, завершит ввод нажатием клавиши Enter.

До того, как клавиша Enter нажата, команда принята не будет. Отменить набираемую команду до нажатия Enter можно, нажав Control-U. Современные оболочки (такие, как «bash») допускают редактирование командной строки до ее ввода. Если при наборе команды допущена ошибка и замечена до нажатия Enter, можно с помощью стрелок переместить курсор на нужное место, удалить ошибочный символ нажатием клавиш Del («удаление», удаление символа над курсором) или Backspace («забой», удаление клавиши слева от курсора) и вставить правильный символ.

Простые команды состоят из одного слова. Возможно, простейшей (и одной из обязательных к заучиванию) является команда «exit». Эта команда приказывает оболочке немедленно завершиться (Рис. 1-2).

```
[alice@wonderland alice]$ exit
login:
```

Рис. 1-2

Таким образом Алиса завершила свой первый сеанс работы в системе, состоящий из одной команды выхода. Этой командой заканчивается любой сеанс работы в системе, хотя не всегда она подается в явном виде.

Простая команда

Простая команда — это строка, состоящая из:

1. (необязательных) присвоений значений переменным окружения команды;
2. (необязательного) имени команды;
3. (необязательных и допустимых только после имени команды) аргу-

ментов команды.

Эти три части команды разделяются промежутками, в качестве которых обычно выступает символ пробела[16]. Отдельные аргументы также разделяются промежутками. Промежутки в начале и конце строки игнорируются, так же, как и дополнительные промежутки (например, второй идущий подряд знак пробела).

Выше мы уже познакомились с командой «exit», которая подавалась (вводилась) без аргументов. Вот еще две команды, которые могут подаваться без аргументов: «date», «cal». Обе они относятся к датам календаря.

Команда «date» (Рис. 1-3) выводит текущую дату и время.

```
[alice@wonderland alice]$ date
Пнд Июн 16 13:58:13 MSD 2003
```

Рис. 1-3

Команда «cal» выводит календарь (Рис. 1-4).

```
[alice@wonderland alice]$ cal
      Июнь 2003
Пн Вт Ср Чт Пт Сб Вс
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

Рис. 1-4

Поданная без аргументов, команда «cal» выводит календарь на текущий месяц. Однако она может вывести календарь и на другой месяц, а также на целый год (Рис. 1-5).

[16] Промежутком также является знак табуляции, однако не следует применять табуляцию в этом качестве; большинство современных оболочек использует клавишу табуляции в интерактивном режиме особым способом. На самом деле, промежутком является и символ новой строки, который просто так ввести также не удастся, поскольку нажатие Enter означает ввод (передачу на исполнение) набранной команды.

```
[alice@wonderland alice]$ cal 7 2003
      Июль 2003
Пн Вт Ср Чт Пт Сб Вс
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
[alice@wonderland alice]$ cal 2004
                              2004

       Январь                     Февраль                      Март
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
          1  2  3  4                        1    1  2  3  4  5  6  7
 5  6  7  8  9 10 11    2  3  4  5  6  7  8    8  9 10 11 12 13 14
12 13 14 15 16 17 18    9 10 11 12 13 14 15   15 16 17 18 19 20 21
19 20 21 22 23 24 25   16 17 18 19 20 21 22   22 23 24 25 26 27 28
26 27 28 29 30 31      23 24 25 26 27 28 29   29 30 31

       Апрель                       Май                        Июнь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
          1  2  3  4                  1  2        1  2  3  4  5  6
 5  6  7  8  9 10 11    3  4  5  6  7  8  9    7  8  9 10 11 12 13
12 13 14 15 16 17 18   10 11 12 13 14 15 16   14 15 16 17 18 19 20
19 20 21 22 23 24 25   17 18 19 20 21 22 23   21 22 23 24 25 26 27
26 27 28 29 30         24 25 26 27 28 29 30   28 29 30
                       31
        Июль                       Август                     Сентябрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
          1  2  3  4                     1           1  2  3  4  5
 5  6  7  8  9 10 11    2  3  4  5  6  7  8    6  7  8  9 10 11 12
12 13 14 15 16 17 18    9 10 11 12 13 14 15   13 14 15 16 17 18 19
19 20 21 22 23 24 25   16 17 18 19 20 21 22   20 21 22 23 24 25 26
26 27 28 29 30 31      23 24 25 26 27 28 29   27 28 29 30
                       30 31
       Октябрь                     Ноябрь                      Декабрь
Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс   Пн Вт Ср Чт Пт Сб Вс
             1  2  3    1  2  3  4  5  6  7           1  2  3  4  5
 4  5  6  7  8  9 10    8  9 10 11 12 13 14    6  7  8  9 10 11 12
11 12 13 14 15 16 17   15 16 17 18 19 20 21   13 14 15 16 17 18 19
18 19 20 21 22 23 24   22 23 24 25 26 27 28   20 21 22 23 24 25 26
25 26 27 28 29 30 31   29 30                  27 28 29 30 31
```

Рис. 1-5

Как мы видим, вывод команды может существенным образом зависеть от аргументов. В данном случае смысл аргументов прозрачен: в первом примере ими были порядковый номер месяца и год, соответствующие месяцу, на который мы хотели получить календарь, а во втором — только год.

Синтаксис (допустимые значения и, во многих случаях, порядок следования) аргументов зависит от конкретной команды, так же, как их семантика (смысл).

Нарушение синтаксиса (*синтаксическая ошибка* оператора) влечет за собой сообщение об ошибке (Рис. 1-6).

```
[alice@wonderland alice]$ cal 13 2003
cal: недопустимый номер месяца: используйте 1-12
```

Рис. 1-6

Иногда оператор может ввести команду, корректную синтаксически, но не соответствующую задаче, то есть совершить *семантическую ошибку*. В следующем примере (Рис. 1-7) Алиса хотела получить календарь на апрель пятого года, но перепутала порядок следования аргументов.

```
[alice@wonderland alice]$ cal 5 4
          Май 4
Вс Пн Вт Ср Чт Пт Сб
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Рис. 1-7

Результатом выполнения команды стал вывод календаря за май 4 г. У программы, исполняющей команду, во многих случаях нет шанса «догадаться» о семантической ошибке. В данном случае Алиса, возможно, заметит свою ошибку, поскольку календарь предваряется названием месяца и года, но далеко не всегда ошибка будет такой очевидной[17].

Ситуация, когда команда, допускающая передачу ей параметров, предполагает некоторые значения этих параметров, если они ей не переданы явно, достаточно типична. В этом случае говорят о поведении команды *по умолчанию*[18]. Например, команда cal по умолчанию выводит календарь на текущий месяц.

Страницы руководства по простым командам

Текущая версия стандарта на ОС описывает 176 команд. В различных системах их доступно от нескольких десятков до нескольких тысяч, и выучить детали синтаксиса и семантики каждой практически невозможно. Поэтому для систем, поддерживающих мобильность пользователей, стандартом предусмотрена выдача электронных страниц руководства по команде «man»[19]. Например, «man cal» должна вывести информацию, соответствующую приведенной для команды «cal» в Справочных материалах.

В некоторых случаях названия сущностей, описанных в различных стра-

[17] По поводу приведенного примера заметим еще, что год «4» соответствует не две тысячи четвертому, и не тысяча девятьсот четвертому годам, а ровно четвертому году от Р.Х.

[18] «By default», т.е. «при отказе» от явного указания параметров.

[19] Сокр. от англ. manual — «руководство».

ницах руководства, совпадает. Например, названию «link» соответствует не только команда операционной системы, но и системный вызов (используемый программистами). В таком случае можно явно указать номер требуемого раздела. Команды ОС описаны в разделе 1, соответственно, получить страницу руководства по команде «link» можно командой «man 1 link»[20].

В большинстве случаев вывод «man» не поместится на экране терминала. В этом случае вывод будет автоматически пропущен через так называемый фильтр постраничного вывода.

Подобно текстовому редактору, фильтр постраничного вывода отображает текст на экране, позволяет «пролистывать» его вверх или вниз, но без возможности редактирования. В большинстве случаев в качестве фильтра будет выступать либо стандартная команда «more», либо команда «less». В любой из них прекратить отображение можно нажатием клавиши «q», пролистнуть текст на экран вперед — клавишей «f», на экран назад — «b».

1.5 Файлы и файловые структуры

Создание и удаление файлов

Именем файла в открытой ОС может быть любая строка поддерживаемых символов, не содержащая нулевого символа[21] и символа косой черты («/»), но рекомендуется ограничиться символами, входящими в *переносимый набор символов имен файлов*. Регистр (заглавность или строчность) букв является значимым.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -
А Б В Г Д Е Ё Ж З И Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ы Ъ Э Ю Я
а б в г д е ё ж з и й к л м н о п р с т у ф х ц ч ш щ ь ы ъ э ю я
```

Рис. 1-8

Далее Алиса будет применять имена из символов, входящих в переносимый набор, а также кириллических букв, входящих в русский алфавит (Рис. 1.8).

Создать файл можно командой «touch», указав ей в качестве аргумента

[20] В документации, литературе и переписке часто имена команд ОС указываются с цифрой 1 в скобках, т.е. вместо «команда “link”» пишут «link(1)». Поскольку мы (почти) не обсуждаем сущностей, могущих оказаться омонимичными именам команд, мы такого соглашения не принимаем.

[21] Нулевой символ не может быть так просто введен с клавиатуры, поэтому сейчас вам не нужно об этом беспокоиться.

имя несуществующего файла. До этого стоит удостовериться, что такого файла действительно не существует, с помощью команды «ls» (сокращение от английского «list» («перечислить»)), *выводящей список файлов* (если они существуют), имена которых перечислены в качестве ее аргументов (Рис. 1-9).

```
[alice@wonderland alice]$ ls 1й_файл
ls: 1й_файл: Такого файла или каталога нет
[alice@wonderland alice]$ touch 1й_файл
[alice@wonderland alice]$ ls 1й_файл
1й_файл
```

Рис. 1-9

На Рис. 1-10 среди аргументов, указанных в командной строке, присутствует «-l» («эль»). Аргумент, начинающийся с дефиса, называется *ключом* команды. Большинство стандартных команд могут применяться с ключами, модифицирующими их действие. Аргумент, не являющийся ключом, называется *операндом*.

```
[alice@wonderland alice]$ ls -l 1й_файл
Итого: 0
-rw-r--r--    1 alice    alice           0 Июн 16 23:18 1й_файл
```

Рис. 1-10

Ключ «-l» задает «длинный» формат вывода команды. Перед списком файлов выводится строка с количеством блоков (обычно 512-байтных), занимаемых перечисленными файлами. Файлу соответствует при этом строка-список из семи *полей* (они разделены символом табуляции, отображаемым при выводе на экран пробелом или серией пробелов), перечисленных в таблице на Рис. 1-11.

-rw-r--r--	1	alice	alice	0	Июн 16 23:18	1й_файл
тип файла и права доступа	количество указателей на файл	имя владельца	имя группы-владельца	размер	время модификации	имя файла

Рис. 1-11

Поле «размер» — это *размер* или *длина* файла в байтах. Алиса лишь создала файл, но ничего не записала в него; такой файл имеет нулевую длину и называется *пустым файлом*.

Тем не менее, у него есть все атрибуты файла, включая *время последней модификации* (в данном случае «последней модификацией» явилось само создание файла) и собственно *имя*.

Удалить существующий файл можно командой «rm» с именем файла в качестве аргумента (Рис. 1-12).

```
[alice@wonderland alice]$ ls 1й_файл
1й_файл
[alice@wonderland alice]$ rm 1й_файл
[alice@wonderland alice]$ ls 1й_файл
ls: 1й_файл: Такого файла или каталога нет
```

Рис. 1-12

Каталоги

Если команда «ls» подается без операндов (т.е. если она подана без аргументов вообще или только с аргументами-ключами), результатом станет вывод списка имен всех файлов в *текущем каталоге* (Рис. 1-13).

```
[alice@wonderland alice]$ ls
1й_файл  tmp
[alice@ wonderland alice]$ ls -l
Итого: 1
-rw-r--r--    1 alice    alice           0 Июн 16 23:18 1й_файл
drwx------    2 alice    alice          48 Июн 10 08:17 tmp
```

Рис. 1-13

Дело в том, что в виде файлов хранится вся информация в системе — все программы, данные, необходимые для их работы, данные пользователей (включая тексты, изображения, звук), и даже, как правило, само ядро ОС. В минималистической встроенной системе файлов может быть сотни, в типичной настольной системе — тысячи, на сервере, обслуживающем многих пользователей — десятки, сотни тысяч или даже миллионы.

Файлы организуются в *файловую структуру*, задающую их логическое расположение. Файловая структура открытых ОС является иерархической: файлы *содержатся* в особых файлах — *каталогах*, каталоги, в свою очередь, могут содержаться в других каталогах и т.д. Вершиной файловой структуры служит каталог «/», называемый *корневым каталогом* файловой структуры.

В открытых ОС *логическая файловая структура независима от физического размещения файлов* на дисках или других носителях. Соответствие физического носителя или его части (раздела жесткого диска) фрагменту файловой структуры системы устанавливается в ходе *монтирования* этого носителя.

Несъемные носители (например, жесткие магнитные диски), как правило, монтируются в ходе загрузки ОС по предзаданному сценарию, а демонтируются, соответственно, в ходе разгрузки. Съемные носители должны монтироваться после их установки в привод и демонтироваться

перед физическим снятием. Монтирование носителя — административная процедура, но на настольных машинах монтирование съемных носителей, как правило, делегируется пользователю, а некоторые современные ОС (в частности, «ГНУ/Линукс») позволяют его автоматизировать.

```
/
 bin/
 boot/
 dev/
 etc/
 home/
      alice/
             tmp/
             1й_файл
      rabbit/
      queenh/
 lib/
 mnt/
 root/
 sbin/
 tmp/
 var/
```

Рис. 1-14

Файловую структуру можно представить в виде дерева с корнем в корневом каталоге и вершинами во вложенных каталогах. На рис. 1-14 пример такого дерева изображен в частично развернутом виде.

Созданный Алисой файл «1й_файл», так же, как и каталог «tmp/», который выше Алиса наблюдала в выводе команды «ls», поданной без параметров, находятся в каталоге «alice/», который, в свою очередь, находится в каталоге «home/», находящемся в корневом каталоге файловой структуры. Знак «косая черта» («/», часто читается «слэш») завершает имя каталога при указании *полного или относительного имени файла*.

В выводе команды «ls -l» файлы-каталоги отличаются от обычных файлов тем, что в первой позиции поля «тип файла и права доступа» стоит буква «d». Тип обычного файла обозначается символом дефиса («-»).

Полное (или *абсолютное*) имя файла однозначно идентифицирует конкретный файл в системе. Краткое имя однозначно идентифицирует файл лишь в отдельном каталоге. В вышеприведенном примере каталоги «/bin/tmp/» и «/home/alice/tmp/» имеют совпадающие краткие имена, но различные *пути* к ним (путем называется часть полного имени, исключающая краткое имя); соответственно, различаются и их полные имена.

Полное (абсолютное) имя в документации и литературе всегда указывается с ведущим слэшем (т.е. начинается с «/»); таким же образом его следует указывать и в качестве аргумента команд ОС.

Ранее, когда Алиса знакомилась с командами «ls», «touch», «rm», она ука-

зывала имя файла «1й_файл» без ведущего слэша (и без указания каталога вообще). Такое именование называется *относительным*. При относительном указании имени файла путь к нему указывается относительно *текущего каталога*. Текущий (рабочий) каталог оболочки определен в каждый момент времени (при вводе каждой команды).

При регистрации пользователя текущим становится *домашний каталог* пользователя, определенный администратором при создании учетной записи этого пользователя. Обычно домашние каталоги пользователей создаются в каталоге «/home/» с именами, совпадающими с регистрационными именами пользователей. Для пользовательницы Алисы, таким образом, домашним будет каталог «/home/alice/» (см. соответствующую вершину дерева файловой структуры, изображенной на рис. выше). Узнать свой домашний каталог пользователь может в любой момент, подав команду «echo ~» (описание команды «echo» см. в Справочных материалах).

В простейшем случае относительное имя файла совпадает с его кратким именем. В примерах с командами «ls», «touch», «rm» выше все имена указывали на файлы в текущем каталоге, которым был домашний каталог Алисы (полное имя этого каталога «/home/alice/»), поэтому выводилась информация о файлах, создавался файл и удалялся файл в этом каталоге.

Отдельный пользователь, в зависимости от цели, с которой он пользуется системой, может работать с десятками, сотнями, тысячами и более файлов. Держать их все в рабочем каталоге неудобно, поскольку пришлось бы придумывать и применять весьма сложные системы их именования. Разумнее рассредоточить их по каталогам, вложенным в свой рабочий каталог.

Создать каталог можно, командой «mkdir», а *удалить пустой* (не содержащий файлов) *каталог* — командой «rmdir» («mkdir» — это сокращение от англ. словосочетания «make directory» («создать каталог»), а «rmdir» — от «remove directory» («удалить каталог»)) с именем каталога в качестве параметра (Рис. 1-15).

```
[alice@wonderland alice]$ mkdir 1й_каталог
[alice@wonderland alice]$ ls -l
Итого: 1
-rw-r--r--    1 alice    alice           0 Июн 16 23:18 1й_файл
drwxr-xr-x    2 alice    alice          48 Июн 22 21:03 1й_каталог
drwx------    3 alice    alice          72 Июн 20 18:27 tmp
[alice@wonderland alice]$ rmdir 1й_каталог
[alice@wonderland alice]$ ls -l
Итого: 1
-rw-r--r--    1 alice    alice           0 Июн 16 23:18 1й_файл
drwx------    3 alice    alice          72 Июн 20 18:27 tmp
```

Рис. 1-15

Сделать каталог текущим можно, командой «cd» (сокращение от англ. словосочетания «change directory» («сменить каталог»)) с именем каталога в качестве параметра (Рис. 1-16).

```
[alice@wonderland alice]$ cd 1й_каталог
[alice@wonderland 1й_каталог]$ ls
[alice@wonderland 1й_каталог]$ pwd
/home/alice/1й_каталог
```

Рис. 1-16

Обратите внимание, что, после того, как Алиса сменила текущий каталог, его краткое имя появилось в ее подсказке. В вашей системе это может быть и не так, но узнать полное имя текущего каталога можно в любой момент, подав команду «pwd» без параметров.

Вместо «текущий каталог в настоящий момент соответствует такому-то» часто говорят «оболочка (или даже пользователь) *находится* в таком-то каталоге», а вместо «сменить текущий каталог на такой-то» — «*перейти* в такой-то каталог».

Подавая команду перехода в каталог, Алиса использовала его относительное имя, совпадающее в данном случае с его кратким именем. Но она могла указать и его полное имя.

Вернуться обратно (т.е. сменить текущий каталог снова на «/home/alice/») она может, указав его полное имя. На самом деле, есть способ проще. «Подняться» на одну ступень по иерархии каталогов можно, использовав специальное имя каталога «..», содержащееся в любом каталоге. Выше Алиса не видела его в списках файлов, выводившихся по команде «ls», поскольку это имя начинается с точки и, соответственно, файл является «скрытым». Увидеть имена скрытых (наряду с прочими) файлов можно командой «ls -a» (Рис. 1-17).

```
[alice@wonderland 1й_каталог]$ ls -a
.  ..
[alice@wonderland 1й_каталог]$ ls -a -l
Итого: 2
drwxr-xr-x    2 alice    alice          48 Июн 22 21:35 .
drwx------    7 alice    alice         528 Июн 22 21:35 ..
[alice@wonderland 1й_каталог]$ ls -al
Итого: 2
drwxr-xr-x    2 alice    alice          48 Июн 22 21:35 .
drwx------    7 alice    alice         528 Июн 22 21:35 ..
[alice@wonderland 1й_каталог]$ cd ..
[alice@wonderland alice]$
```

Рис. 1-17

В данном случае «пустой» каталог содержит два файла-каталога «.» и «..», первый из которых является самим каталогом, в котором он содержится, а второй — каталогом на ступень выше в иерархии. Обратите внимание,

что два ключа (например, «-a» и «-l») можно указать и в сокращенной форме — дефис и следующие за ним буквы ключей без пробела (в данном случае, «-al»).

И, наконец, перейти в домашний каталог из любой вершины в файловой структуре можно командой «cd» без параметров.

Копирование, перемещение, переименование файлов

Командой «cp» можно скопировать файлы, командой «mv» — переименовать (переместить) их. Обе эти команды имеют два отличающихся по семантике варианта.

1) Если последним операндом является имя существующего каталога, то файлы, имена которых указаны в качестве предшествующих операндов, копируются или перемещаются в этот каталог.

2) Если последним операндом является имя обычного файла, то файл, имя которого указано в качестве предшествовавшего операнда, копируется или переименовывается в этот файл.

Указание в этих командах единственного операнда, а также указание более двух операндов в случае, если последний из них не является именем существующего каталога — ошибка.

```
[alice@wonderland alice]$ mkdir еще_каталог
[alice@wonderland alice]$ cd еще_каталог/
[alice@wonderland еще_каталог]$ touch первый второй третий
[alice@wonderland еще_каталог]$ mkdir и_еще_каталог
[alice@wonderland еще_каталог]$ ls
и_еще_каталог  второй  первый  третий
[alice@wonderland еще_каталог]$ mv третий второй и_еще_каталог
[alice@wonderland еще_каталог]$ ls
и_еще_каталог  первый
[alice@wonderland еще_каталог]$ ls и_еще_каталог
второй  третий
[alice@wonderland еще_каталог]$ mv первый четвертый
[alice@wonderland еще_каталог]$ ls
и_еще_каталог  четвертый
[alice@wonderland еще_каталог]$ cp и_еще_каталог/второй и_еще_каталог/третий .
[alice@wonderland еще_каталог]$ ls
и_еще_каталог  второй  третий  четвертый
[alice@wonderland еще_каталог]$ ls и_еще_каталог
второй  третий
```

Рис. 1-18

Приведенный на Рис. 1-18 пример демонстрирует действие команд «cp» и «mv».

Чтобы избежать случайного удаления файлов в случае, если при копировании или перемещении файлов имена копируемых или переименовываемых файлов совпадают с именами существующих, можно использовать ключ «-i» (Рис. 1-19).

```
[alice@wonderland еще_каталог]$ cp -i и_еще_каталог/второй .
cp: переписать `./второй'? y
[alice@wonderland еще_каталог]$ mv -i второй вложенный_каталог
mv: переписать `вложенный_каталог/второй'? y
```

Рис. 1-19

Генерация имен файлов

В качестве операндов чаще всего выступают имена файлов, и во многих случаях операцию желательно выполнить не над одним, а над целым списком файлов. Стандартная оболочка реализует особый механизм для указания списков имен файлов, если эти имена формально-синтаксически схожи (начинаются с одной буквы, заканчиваются одним расширением и т.п.). Этот механизм называется «глоббингом», *генерацией имен файлов* или *раскрытием метасимволов в именах*. Он заимствован и многими альтернативными системами, но в отличие от большинства из них, в открытых системах раскрытие метасимвола осуществляется оболочкой, а не командой.

- Вопросительный знак («?») соответствует любому одному символу в имени файла. Если у нас в каталоге присутствуют файлы «a1», «a2», «a3», «b1», «b2», «b3», «aa1», *шаблон имени* (метаимя) «a?» раскроется в список «a1 a2 a3», а шаблон «?1» — в «a1 b1».
- Звездочка (астериск, «*») соответствует последовательности из нуля или большего количества любых символов. В том же каталоге «a*» раскроется в список «a1 a2 a3 aa1», а «*1» — в «a1 b1 aa1».
- Метаконструкция из последовательности символов, заключенных в квадратные скобки («[» и «]»), соответствует любому одному символу из этой последовательности. В том же каталоге «[abc]2» раскроется в список «a2 b2».
- В квадратных скобках могут содержаться диапазоны, разделенные дефисом («-»). Они означают любой символ, входящий в этот диапазон с учетом алфавитного порядка следования символов. В нашем каталоге «[a-c]3» раскроется в «a3 b3».
- Список может быть предварен знаком отрицания (сиркумфлекс, «^»), в этом случае он означает любой символ, не входящий в список. Если в шаблон нужно буквально включить символ «-», его следует поставить на первое или последнее место, а «^» — на любое место, кроме первого.

Конструкция в квадратных скобках может быть сколь угодно сложной (например, «[a-ckw-z]» означает «любой символ с “a” по “c”, или “k”, или

с “w” по “z”»), и она всегда соответствует *одному* символу в раскрываемых именах.

Вопросительные знаки, звездочки и квадратно-скобочные конструкции могут произвольно сочетаться. Список всегда раскрывается в алфавитном порядке.

```
[alice@wonderland еще_каталог]$ mkdir мета
[alice@wonderland alice]$ cd мета
[alice@wonderland мета]$ touch a1 a2 a3 b1 b2 b3 aa1
[alice@wonderland мета]$ ls
a1  a2  a3  aa1  b1  b2  b3
[alice@wonderland мета]$ ls a?
a1  a2  a3
[alice@wonderland мета]$ ls ?1
a1  b1
[alice@wonderland мета]$ ls a*
a1  a2  a3  aa1
[alice@wonderland мета]$ ls *1
a1  aa1  b1
[alice@wonderland мета]$ ls [abc]2
a2  b2
[alice@wonderland мета]$ ls [a-c]3
a3  b3
```

Рис. 1-20

В примере на Рис. 1-20 Алиса создает каталог с перечисленными выше файлами и получает списки файлов, соответствующих некоторым из перечисленных шаблонов.

Еще одним полезным метасимволом является тильда («~»), выступающая в качестве такового только в случае, когда стоит первой в аргументе. Отдельная тильда раскрывается в полное имя домашнего каталога текущего пользователя. Тильда, за которой следует (без пробела) регистрационное имя пользователя, раскрывается в полное имя его домашнего каталога. Если оболочке не удается раскрыть метасимвол, он передается команде в буквальном виде (Рис. 1-21).

```
[alice@wonderland 1й_каталог]$ echo ~
/home/alice
[alice@wonderland alice]$ echo ~rabbit
/home/rabbit
[alice@wonderland alice]$ echo ~bob
~bob
```

Рис. 1-21

Экранирование специальных символов

Специальное значение символов «?», «*», «[», «]», «~» при указании имен файлов и является причиной, по которой их (а также другие символы, имеющие специальное значение для оболочки) не рекомендуется вводить в имена файлов. Однако пользователь может столкнуться с ситу-

ацией, в которой ему все же нужно выполнить некоторые действия с файлом, чье имя содержит такие символы.

Алиса перенесла в каталог «Старые_файлы/» файл «Домашняя страница [13].htm» из системы, которой ранее пользовалась.

Как ей к нему обратиться? Буквальное указание в командной строке цепочки символов, совпадающей с именем файла, очевидно, не приведет к разумному результату, поскольку будет интерпретировано как список, состоящий из имени «Домашняя» и шаблона «страница[13].htm» (которому могут соответствовать файлы «страница1.htm» и «страница3.htm»).

```
[alice@wonderland Старые_файлы]$ ls
Домашняя страница[13].htm
[alice@wonderland Старые_файлы]$ ls Домашняя страница[13].htm
ls: Домашняя: No such file or directory
ls: страница[13].htm: No such file or directory
```

Рис. 1-22

В лучшем случае эти файлы не будут найдены (Рис. 1-22), в худшем будут найдены другие файлы, чьи названия случайно совпадут с элементами невольно введенного «списка» или результатами раскрытия «шаблона».

Чтобы указать в командной строке файл, чье имя содержит специальные символы, эти символы необходимо *экранировать*, т.е. «защитить» от раскрытия. Экранировать отдельный символ можно, поставив перед ними символ обратной косой черты («\», «бэкслэш»). Цепочка «Домашняя\ страница\[13\].htm» раскрывается в цепочку «Домашняя страница [13].htm» (Рис. 1-23).

```
[alice@wonderland Старые_файлы]$ ls Домашняя\ страница\[13\].htm
Домашняя страница[13].htm
```

Рис. 1-23

Экранировать бэкслэшем можно любой специальный символ. Если необходимо, чтобы в цепочке был раскрыт сам символ «\», он также экранируется.

```
[alice@wonderland Старые_файлы]$ echo Специальные символы в шаблонах --- это
вопросительный знак \?, звездочка \*, квадратные скобки \[ и \]. Их можно
экранировать обратной косой чертой \\
Специальные символы в шаблонах --- это вопросительный знак ?, звездочка *,
квадратные скобки [ и ]. Их можно экранировать обратной косой чертой \
```

Рис. 1-24

Другой способ экранировать специальные символы от интерпретации как шаблонных — заключить имя файла целиком в апострофы («'») или кавычки («"»).

```
[alice@wonderland Старые_файлы]$ ls "Домашняя страница[13].htm"
Домашняя страница[13].htm
[alice@wonderland Старые_файлы]$ ls 'Домашняя страница[13].htm'
Домашняя страница[13].htm
```

Рис. 1-25

Экранирование апострофами несколько отличается от экранирования кавычками, но эти отличия мы обсудим позже.

Хотя используя экранирование можно создавать, перемещать и копировать, уничтожать файлы, имена которых состоят практически из любых символов, включать специальные (как шаблонные, так и прочие) символы в имена файлов крайне не рекомендуется, так как это заметно повышает вероятность ошибки при вводе.

Оболочка не придает никакого особого значения точке в имени файла (кроме случаев, когда имя начинается с точки), и «расширение имени файла» — это лишь интерпретация пользователя (и, возможно, некоторых программ). Поэтому в отличие от ряда альтернативных систем шаблон «*.» означает не «все файлы с именами без расширений» (как в «РСЭКС-11» или «МС-ДОС»), но буквально «все файлы с именами, заканчивающимися на точку».

Перенаправление ввода-вывода

Команды «cp», «ls», «mkdir», «mv», «rm», «rmdir», «touch», с помощью которых Алиса манипулировала файлами в примерах выше, обычно относят к «файловым утилитам». Все они позволяют манипулировать файлом (копировать, выводить информацию о нем, переименовывать или перемещать, создавать) как единым целым. Любые действия файловых утилит совершенно безразличны к содержимому файлов.

Команда «cat» обычно относится к «текстовым утилитам». Строго говоря, она может обрабатывать и двоичные файлы, но применительно к ним, как правило, операция объединения (а именно объединение содержимого нескольких файлов в один и является «титульной» функцией этой команды, название которой представляет собой сокращение от английского «(con)catenate» — «(кон)катенировать», «сцеплять») бессмысленна.

```
[alice@wonderland alice]$ cat
```

Рис. 1-26

Введя команду «cat» без аргументов (Рис. 1-26), Алиса сталкивается с новой для себя ситуацией: ее подача не приводит ни к какому видимому результату, никакого вывода на экране не появляется, но и подсказки, ко-

торая была бы знаком успешного завершения команды, тоже нет. Дело в том, что команда «cat», в отличие от ранее рассмотренных, не только выводит, но и *вводит* данные.

Вводя произвольные строки[22] (Рис. 1-27), Алиса обнаруживает, что каждая введенная ею строка после нажатия Enter вновь выводится на терминал. Команда «cat», поданная без аргумента, копирует содержимое ввода в вывод построчно[23].

Закончить ввод можно, введя в начале очередной строки символ конца файла Control-D. Ввод этого символа приводит к немедленному завершению ввода, т.е. символ завершения строки уже не ожидается, а сам символ не отображается при вводе.

```
[alice@wonderland alice]$ cat
Слушало, слышало вещее мое все эти речи еще за месяц!
Слушало, слышало вещее мое все эти речи еще за месяц!
То есть,  я  говорю, что нашему брату, хуторянину, высунуть нос из своего за-
холустья в большой свет --- батюшки мои!
То есть,  я  говорю, что нашему брату, хуторянину, высунуть нос из своего за-
холустья в большой свет --- батюшки мои!
^D
[alice@wonderland alice]$
```

Рис. 1-27

Говоря в главах выше о выводе, а сейчас и о вводе, мы подразумевали вывод на экран терминала и ввод с клавиатуры терминала. Это является умолчанием для всех стандартных команд. Однако одним из самых важных свойств открытых ОС является возможность *перенаправления ввода-вывода*.

В примере на Рис. 1-28 Алиса *перенаправляет вывод* команды «cat», используя символ «>» со следующим за ним именем файла («Вечера»), в который перенаправляется вывод.

```
[alice@wonderland alice]$  cat >Вечера
Слушало, слышало вещее мое все эти речи еще за  месяц!
То есть,  я  говорю, что нашему брату, хуторянину, высунуть нос из своего за-
холустья в большой свет -- батюшки мои!
^D
[alice@wonderland alice]$ ls -l Вечера
-rw-r--r--    1 alice    alice         173 Июн 23 03:09 Вечера
```

Рис. 1-28

В этом примере строки ввода уже не дублируются выводом на терминал,

---

[22] Это первые предложения второго абзаца «Вечеров на хуторе близ Диканьки» Н.В.Гоголя.

[23] Стандарт предусматривает также ключ «-u», обеспечивающий побайтное, а не построчное копирование, однако ввиду буферизации ввода-вывода терминалом большинства систем такое различие не будет заметно в нашем примере.

а поданная после завершения ввода команда «ls» показывает, что действительно появился файл с названием «Вечера» и размером 173 байта, что соответствует длине введенного текста[24].

Перенаправляться может не только вывод, но и ввод, и Алиса может воспользоваться этим, чтобы вывести с помощью все той же команды «cat» содержимое созданного ею файла на терминал. Перенаправление ввода осуществляется указанием имени файла после символа «<» (Рис. 1-29).

```
[alice@wonderland alice]$ cat <Вечера
Слушало, слышало вещее мое все эти речи еще за  месяц!
То есть,  я  говорю, что нашему брату, хуторянину, высунуть нос из своего за-
холустья в большой свет -- батюшки мои!
```

Рис. 1-29

Эти два символа («>» для перенаправления вывода и «<» для перенаправления ввода) легко запомнить, поскольку они графически соответствуют «стрелкам», указывающим «в файл» или «из файла». Промежутки до и после символов «<» и «>» игнорируются.

Перенаправление ввода и вывода может использоваться и одновременно. Команда на Рис. 1-30 копирует построчно файл «Вечера» в файл «Вечера_2».

```
[alice@wonderland alice]$ cat <Вечера >Вечера_2
[alice@wonderland alice]$ ls -l Вечера_*
-rw-r--r--    1 alice    alice         173 Июн 23 03:25 Вечера_2
-rw-r--r--    1 alice    alice         173 Июн 23 03:09 Вечера
```

Рис. 1-30

Результат, в общем, совпадает с результатом простого копирования «cp Вечера Вечера_2», хотя и достигается другим способом. Порядок указания файлов перенаправления ввода и вывода значения не имеет.

Если файл, в который перенаправляется вывод, уже существует, он будет опустошен. Оболочка при этом может запрашивать подтверждение на опустошение файла.

Часто бывает желательно перенаправить вывод в файл, не уничтожая его содержимого, а *дописывая* новые строки к уже существующим. Для этого оболочка поддерживает *перенаправление вывода в конец существующего файла*, обозначаемое «двойной стрелкой» «>>» (Рис. 1-31).

[24] В восьмибитной (однобайтовой) кодировке. В кодировке UTF8 размер файла был бы почти в два раза больше, поскольку одному кириллическому символу в ней соответствует двухбайтовая (шестнадцатибитовая) последовательность.

```
[alice@wonderland alice]$ cat >>Вечера
Это все равно как, случается, иногда зайдешь в покои великого пана: все
обступят тебя и пойдут дурачить.
^D
[alice@wonderland alice]$ ls -l Вечера*
-rw-r--r--    1 alice    alice         285 Июн 23 21:10 Вечера
-rw-r--r--    1 alice    alice         173 Июн 23 03:25 Вечера_2
[alice@wonderland alice]$ cat <Вечера
Слушало, слышало вещее мое все эти речи еще за  месяц!
То есть,  я  говорю, что нашему брату, хуторянину, высунуть нос из своего за-
холустья в большой свет -- батюшки мои!
Это все  равно как,  случается,  иногда  зайдешь  в  покои  великого пана: все
обступят тебя и пойдут дурачить.
```

Рис. 1-31

Как и перенаправление вывода в файл, перенаправление вывода в конец файла может применяться совместно с переназначением ввода.

Важно понимать, что возможность переназначения ввода-вывода является свойством не отдельных программ, а системы в целом. Символы перенаправления и соответствующие имена файлов *не* передаются самим командам в качестве аргументов. Оболочка самостоятельно назначает файлы ввода-вывода любой команды. В частности, можно переназначить вывод любой из рассматривавшихся выше команд, например, «ls» (Рис. 1-32).

```
[alice@wonderland alice]$ ls Вечера* >список
[alice@wonderland alice]$ cat <список
Вечера
Вечера_2
```

Рис. 1-32

Кроме стандартного ввода и стандартного вывода, каждая команда открывает еще один файл для вывода ошибок. Команда записывает в стандартный вывод то, что от нее ожидается, а в вывод ошибок — пишет сообщения об ошибках, предупреждения и диагностические сообщения. Как и первые два файла, по умолчанию вывод ошибок ассоциирован с терминалом. Перенаправление стандартного ввода и стандартного вывода не влияет на вывод ошибок.

```
[alice@wonderland alice]$ ls Вечера_3 >список
ls: Вечера_3: No such file or directory
[alice@wonderland alice]$ ls -l список
-rw-r--r--    1 alice    alice           0 Июн 23 22:34 список
```

Рис. 1-33

В примере на Рис. 1-33 сообщение об отсутствии файла «Вечера_3» было выведено на терминал, хотя стандартный вывод был перенаправлен в файл «список».

Перенаправить вывод ошибок можно, использовав конструкцию «2>» со следующим за ней именем файла (Рис. 1-34).

```
[alice@wonderland alice]$ ls Вечера_3 2>список
[alice@wonderland alice]$ cat <список
ls: Вечера_3: No such file or directory
```

Рис.1-34

Двойка перед символом перенаправления в этой конструкции означает порядковый номер (дескриптор) канала ввода-вывода; стандартный ввод и стандартный вывод имеют дескрипторы 0 и 1, соответственно[25], а запись «<», «>», «>>» является сокращением от «0<», «1>», «1>>», соответственно[26].

Так же, как и стандартный вывод, вывод ошибок может быть переназначен в конец существующего файла конструкцией «2>>». Если же необходимо переназначить и стандартный вывод, и вывод ошибок в один файл, в командную строку следует, помимо переназначения стандартного вывода включить еще и конструкцию «2>&1», означающую «переназначить второй канал туда же, куда и первый» (Рис. 1-35).

```
[alice@wonderland alice]$ ls Вечера_2 Вечера_3 >список 2>&1
[alice@wonderland alice]$ cat <список
ls: Вечера_3: No such file or directory
Вечера_2
```

Рис. 1-35

Стандартные файлы-устройства

Бывает желательно подавить стандартный вывод или вывод ошибок вообще. Некоторые команды предусматривают для этого особые ключи, но в общем случае можно воспользоваться все той же возможностью переназначения вывода. Для этого в любой стандартной системе существует специальный файл, представляющий собою фиктивное «нуль-устройство». Его полное имя «/dev/null». Запись в него любых данных не приводит к какому-либо результату, они как бы «бесследно исчезают»[27]. В «/dev/null» можно переназначить как стандартный вывод, так и вывод ошибок.

На «/dev/null» можно также переназначить и стандартный ввод; из него

[25] Программистам дескрипторы 0, 1, 2 известны также как «STDIN», «STDOUT», «STDERR».

[26] Команда может открывать и большее количество файлов ввода-вывода, но лишь три из них ассоциируются с терминалом автоматически при ее подаче.

[27] Отсюда фольклорное название «битодробилка» и многочисленные шутки, касающиеся того, как бы могло выглядеть соответствующее устройство, выполненное «в железе», и как бороться с переполнением «/dev/null».

всегда читается пустой файл: подача команды «cat </dev/null >пустой_файл» приведет к появлению в текущем каталоге пустого файла «пустой_файл» (или опустошению существующего файла с таким именем).

Еще один интересный специальный файл-устройство — «/dev/tty». Это весьма абстрактное устройство, соответствующее терминалу, с которого запущена оболочка. Перенаправление вывода в или ввода из этого устройства не дает никакого видимого эффекта, поскольку совпадает с умолчанием (можно считать, что, если в команде явным образом не присутствуют перенаправления, ввод-вывод неявным образом направлен так: «</dev/tty >/dev/tty 2>/dev/tty»), но его указание может пригодиться для команд, вводящих текст из файла, указанного в качестве операнда, или выводящих текст в такой файл.

Если Алиса взглянет на перечисленные файлы с помощью команды «ls -l» (Рис. 1-36), она обнаружит, что в поле «тип» присутствует не встречавшийся до сих пор символ «c».

```
[alice@wonderland alice]$ ls -l /dev/null /dev/tty
crw-rw-rw-    1 root     root       1, 3 Янв 20 19:24 /dev/null
crw-rw-rw-    1 root     root       5, 0 Июн 24 20:11 /dev/tty
```

Рис. 1-36

Символ «c» означает «устройство с посимвольным вводом-выводом»[28].

Эти файлы устройств должны присутствовать в любой стандартной открытой системе, так же, как и содержащий их каталог «/dev/». Кроме них, в большинстве реализаций этот каталог содержит множество (сотни или даже тысячи) файлов (иногда организованных в подкаталоги), представляющих различные физические или виртуальные устройства. *Любое устройство в открытой ОС представлено в виде файла.* Некоторые из них (например, терминалы) представляют собой устройства с посимвольным вводом-выводом, некоторые (например, магнитные диски) — с поблочным. Тип файла-устройства с поблочным вводом-выводом обозначается буквой «b».

Оболочка как команда

Пожалуй, наиболее убедительной демонстрацией единства принципов открытых систем является следующий пример. Запишем в файл «сцена-

[28] Также в строках вывода команды «ls -l» отсутствует поле «размер»; его место занимают пары чисел, называемых *старшим и младшим номерами устройства* (в данном случае «5, 1», «1, 3» и «5, 0»). Их значения зависят от конкретной реализации и интересны, как правило, администратору системы, но не обычному пользователю.

рий» строки, соответствующие каким-либо уже известным нам командам (например, «date», «cal») и дадим команду «sh <сценарий» (Рис. 1-37).

```
[alice@wonderland alice]$  cat >сценарий
date
cal
^D
[alice@wonderland alice]$ sh <сценарий
Пнд Июн 23 22:55:39 MSD 2003
      Июнь 2003
Пн Вт Ср Чт Пт Сб Вс
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

Рис. 1-37

Команда «sh» является ни чем иным, как запуском еще одной оболочки, такой же[29], как та, с которой Алиса работает, вводя команды с терминала. Поскольку ввод перенаправлен, команды читаются не с терминала, а из файла.

Последовательность команд, записанная в файл, представляет собой своего рода программу, которую часто называют *сценарием* или скриптом[30]. На самом деле оболочка предоставляет в распоряжение пользователя развитый директивный язык программирования, с основами которого мы познакомимся позже, а пока следует заметить, что значительная часть самих ОС обычно пишется на этом языке.

«Владение» файлом и «права» на него

Рассмотрим следующий пример (Рис. 1-38).

```
[alice@wonderland alice]$ touch файл
[alice@wonderland alice]$ ls -l файл
-rw-r--r--    1 alice    alice           0 Июл  1 10:42 файл
[alice@wonderland alice]$ chmod u-w файл
[alice@wonderland alice]$ ls -l файл
-r--r--r--    1 alice    alice           0 Июл  1 10:42 файл
[alice@wonderland alice]$ cat >файл
-bash: файл: В доступе отказано
```

Рис. 1-38

Алиса известными нам уже командами «touch» и «ls» создает файл

---

[29] В большинстве реальных ситуаций пользователь работает не со стандартной оболочкой «sh», а с усовершенствованной оболочкой, такой как «bash» или «zsh», совместимой сверху вниз со стандартной. При этом команда «sh» может быть, в зависимости от конкретной ОС, либо синонимом такой усовершенствованной, либо также включенной в систему строго стандартной оболочкой.

[30] От англ. «script» («сценарий»).

«файл» и убеждается в том, что он на самом деле создан. Затем она выполняет команду «chmod u-w файл» и обнаруживает, что поле «тип и права доступа» в выдаче «длинного» списка файлов претерпело некоторые изменения. Попытка перенаправить в этот файл ввод порождает сообщение оболочки «В доступе отказано».

Вспомним еще раз значения полей в «длинном» формате списка файлов, получаемого по команде «ls -l» (Рис. 1-39).

`-r--r--r--`	`1`	`alice`	`alice`	`0`	`Июл 1 10:42`	`файл`
тип файла и права доступа	количество указателей на файл	имя владельца	имя группы-владельца	размер	время модификации	имя файла

Рис. 1-39

Вот как устроено поле «тип и права» (Рис. 1-40).

-	***r***	**-**	**-**	***r***	**-**	**-**	***r***	**-**	**-**
тип файла	права владельца			права группы			права остальных пользователей		
	чтение	запись	исполнение	чтение	запись	исполнение	чтение	запись	исполнение

Рис. 1-40

Оно всегда содержит десять символов. Значение первого символа нам уже известно: это «тип файла», которому могут соответствовать «-» (обычный файл), «d» (каталог) и некоторые другие символы, соответствующие специальным файлам (таким, как файлы устройств).

Остальные девять символов составляют три триады, выражающие *права на файл* в так называемой «rwx»-нотации. Они соответствуют трем категориям пользователей, определяемым относительно каждого файла.

В первую категорию «владелец» входит один пользователь, являющийся «владельцем» данного файла. Это пользователь, чье имя указано в соответствующем (третьем) поле «длинного» формата списка файлов. Обычно владелец файла — это создавший его пользователь.

Во вторую категорию входят все пользователи, входящие в группу пользователей. Группы пользователей — это механизм, введенный в стандарт открытых систем специально для распределения прав на файлы. Создание групп, включение в них пользователей и исключение пользователей из групп — административные действия. Файл имеет группу-владельца, совпадающую с текущей группой создавшего его пользователя на момент создания, а текущая группа обычно совпадает с первичной группой пользователя (пользователь может входить более, чем в одну группу).

В примере имя группы совпадает с именем пользователя[31], но это разные сущности.

В третью категорию входят все остальные пользователи, т.е. все пользователи за исключением владельца и членов группы-владельца.

Для каждой категории определяются отдельные «правомочия» доступа к файлу.

- Правомочие чтения разрешает чтение содержимого файла. Значение этого символа может быть «-» (запрещено) или «r» (разрешено; от англ. «read» («читать»)).
- Правомочие записи разрешает модификацию файла, его значение может быть «-» (запрещено) или «w» (разрешено; от англ. «write» («писать»)).
- Правомочие исполнения разрешает выполнение программы, содержащейся в файле, путем указания ее имени. Значение этого бита может быть «-» (запрещено) или «x» (разрешено; от англ. «eXecute» («исполнять»))[32]. Зачем нужно правомочие исполнения, мы рассмотрим ниже.

Ограничения системы прав на файлы не действуют для главного пользователя (root). Главный пользователь может читать любые файлы в структуре, писать любые файлы, кроме расположенных в подструктурах, смонтированных только для чтения, и исполнять любые файлы, исполнение которых разрешено хотя бы одной категории пользователей.

Изменять права доступа к файлу может лишь его владелец (или главный пользователь системы). Для этого служит команда «chmod» (от англ. «change mode» («изменить режим» доступа к файлу), уже встретившаяся нам в примере выше. Синтаксис этой команды поддерживает две нотации — символическую и числовую. Мы рассмотрим лишь символическую.

Символическая нотация представляет собой операнд-слитную запись клауз из трех составляющих: категории пользователя, вид назначения прав и собственно назначаемые правомочия. Операнд может включать в себя более одной клаузы. Клаузы разделяются запятыми (*без промежутков*).

---

[31] Это концепция так называемых «собственных групп», поддерживаемая во многих системах «ГНУ/Линукс».

[32] В этом же поле отображаются и другие правомочия, которые мы не обсуждаем. В случае, если в этом поле присутствует другая буква, строчная буква (например, «s») означает исполняемый файл, а заглавная (например, «S») — неисполняемый.

Категории пользователей соответствуют описанным выше и обозначаются последовательностями букв:

- «u» — владелец файла; от англ. «user» («пользователь»));
- «g» — группа-владелец файла; от англ. «group» («группа»));
- «o» — остальные пользователи; от англ. «other» («прочие»)).
- «a» — все пользователи (от англ. «all» («все»)), это сокращенная запись для «ugo».

Вид назначения прав может быть трояким:

- «+» — добавить правомочия;
- «-» — отнять правомочия;
- «=» — установить права, в точности соответствующие назначаемым.

Назначаемые правомочия обозначаются последовательностями уже известных нам букв «rwx»-нотации «r», «w», «x», соответствующим правам на чтение, запись и исполнение.

Таким образом,

- «chmod u-w файл» отнимет правомочие записи у владельца;
- «chmod g+rw файл» добавит правомочия чтения и записи группе-владельцу;
- «chmod go=r» установит правомочия группы-владельца и прочих пользователей в точности равными «только чтению»;
- «chmod a+x» добавит правомочие исполнения всем пользователям;
- «chmod u=rwx,g=rw,o=r» установит правомочия чтения, записи и исполнения для пользователя, чтения и записи для группы и чтения для всех остальных.

Маска прав по умолчанию

Когда пользователь создает файл (командой «touch» или перенаправлением вывода другой команды), права доступа к нему устанавливаются равными *маске прав по умолчанию*, за исключением того, что правомочие исполнения обычному файлу не присваиваются. Права по умолчанию задаются командой «umask».

Команда «umask -S» без параметров выводит в символическом виде маску прав по умолчанию. Команда «umask» с параметром в «ugo»-нотации (такой же, как у команды «chmod») добавляет, отнимает или устанавлива-

ет права в маске прав.

```
[alice@wonderland alice]$ umask -S
u=rwx,g=rx,o=rx
[alice@wonderland alice]$ touch файл_1
[alice@wonderland alice]$ ls -l файл_1
-rw-r--r--    1 alice    alice           0 Июл  1 13:43 файл_1
[alice@wonderland alice]$ umask o-r
[alice@wonderland alice]$ umask -S
u=rwx,g=rx,o=x
[alice@wonderland alice]$ touch файл_2
[alice@wonderland alice]$ ls -l файл_2
-rw-r-----    1 alice    alice           0 Июл  1 13:44 файл_2
```

Рис. 1-41

В примере на Рис. 1-41 Алиса выводит маску, создает файл «файл_1», убеждается в том, что права на вновь созданный файл соответствуют маске, отнимает у прочих пользователей вновь создаваемых файлов правомочие чтения, создает файл «файл_2» и убеждается в том, что права на него соответствуют новому значению маски.

Утилита «umask» не является файловой и изменение значения маски не влияет на права существующих файлов. Значение маски сохраняется до нового их изменения командой «umask» или конца сеанса работы с оболочкой.

Особенности прав на каталоги

Следующий пример может показаться контринтуитивным.

```
[alice@wonderland alice]$ ls -l файл
-r--r--r--    1 alice    alice           0 Июл  1 10:42 файл
[alice@wonderland alice]$ rm файл
rm: удалить файл `файл'? y
[alice@wonderland alice] $ ls -l файл
ls: файл: No such file or directory
```

Рис. 1-42

У Алисы нет прав на запись в файл «файл». Тем не менее, она может удалить его командой «rm» (Рис. 1-42).

Но никакого парадокса в этом нет. Удаление файла не является изменением его содержания. Удаление файла — это изменение *каталога*, в котором он содержится и, соответственно, разрешение или запрещение удаления файла зависит не от прав на него, но от прав на каталог (мы помним, что каталог — это тоже файл).

```
[alice@wonderland alice]$ umask -S
u=rwx,g=rx,o=rx
[alice@wonderland alice]$ mkdir каталог_1
[alice@wonderland alice]$ touch каталог_1/файл_1
[alice@wonderland alice]$ chmod u-w каталог_1/файл_1
[alice@wonderland alice]$ rm каталог_1/файл_1
rm: удалить файл `каталог_1/файл_1'? y
[alice@wonderland alice]$ touch каталог_1/файл_1
[alice@wonderland alice]$ chmod u-w каталог_1/
[alice@wonderland alice]$ rm каталог_1/файл_1
rm: удалить файл `каталог_1/файл_1'? y
rm: невозможно удалить `каталог_1/файл_1': Недостаточно прав
[alice@wonderland alice]$ touch каталог_1/файл_2
touch: создание `каталог_1/файл_2': Недостаточно прав
```

Рис. 1-43

В примере на Рис. 1-43 Алиса создает каталог «каталог_1», создает в нем файл «файл_1», отнимает у владельца (себя) права на запись, тем не менее, удаляет его, затем создает такой же файл и отнимает у себя права на запись в этот каталог. После этого попытка удаления файла приводит к выводу сообщения о нехватке прав для совершения этой операции.

Соответственно, и создать файл в каталоге, прав записи на который у нее нет, она не сможет[33].

Обратите внимание, что отсутствие права записи в каталог не отнимает у Алисы права на изменение содержимого находящихся в нем файлов (Рис. 1-44).

```
[alice@wonderland alice]$ ls -l каталог_1/
-rw-r--r--    1 alice    alice           0 Июл  1 14:25 файл_1
[alice@wonderland alice]$ cat >каталог_1/файл_1
123
[alice@wonderland alice]$ cat <каталог_1/файл_1
123
[alice@wonderland alice]$ ls -l каталог_1/
-rw-r--r--    1 alice    alice           4 Июл  1 14:25 файл_1
[alice@wonderland alice]$
```

Рис. 1-44

Это вполне логично, т.к. изменение содержимого никак не влияет на запись в каталоге. Однако здесь есть одна тонкость. Обратите внимание, что первая команда «ls -l» показала длину файла равной 0 байт (что естественно, т.к. этот файл создавался как пустой), а вторая — 4 байта. Разве информация о длине файла не является частью записи о нем в каталоге?

[33] Во многих системах крайне полезным оказывается определить особый режим для некоторых каталогов, при котором пользователь имеет право создавать в нем новые файлы, но удалять может лишь те, что принадлежат ему. Для этого стандарт предусматривает расширение набора прав, известное, как «липкий бит».

Вся правда о файлах

Дело в том, что понятие о файлах и их «нахождении» в каталоге выше давалось нами в несколько упрощенной форме. Если быть точными, каталог содержит не «файлы», а *записи о файлах*, вполне подобно тому, как библиотечный каталог содержит не книги, а записи о книгах (или библиографические карточки), а сами книги хранятся на полках[34]. Часть полей, выдаваемых командой «ls -l», относится к файлу как единице хранения («книге на полке»), а часть — к записи о нем в каталоге («библиографической карточке»).

Атрибутом записи о файле в каталоге является поле «имя».

Атрибутам файла как единицы хранения (его называют *индексным узлом* или и-узлом) соответствуют поля «тип и права», «количество указателей», «владелец», «группа-владелец», «размер», «время модификации».

Кстати говоря, поле «количество указателей» и содержит число «библиографических карточек» (записей в каталогах), соответствующих «книге» (и-узлу). Мы до сих пор имели дело только с и-узлами, которым соответствует одна запись (так обычно и бывает с файлами, создаваемыми пользователями), но так же, как книге могут соответствовать разные карточки (одна в предметном каталоге, другая в алфавитном каталоге названий, третья в алфавитном каталоге авторов...), на один и тот же и-узел могут ссылаться записи в разных каталогах (или разные записи в одном каталоге под разными именами). Создание и удаление дополнительных имен («ссылок») нами рассматриваться здесь не будет.

В то время, как правомочия чтения и записи на каталог вполне прозрачны (разрешение чтения позволяет прочитать список содержащихся в нем файлов (например, командой «ls»), а записи — модифицировать этот список, т.е. создавать и удалять содержащиеся в этом каталоге файлы), правомочие исполнения имеют для каталога особый смысл. Оно означает «право прохождения сквозь», т.е. право на обращение к файлам, содержащимся в каталоге и в его подкаталогах, даже если права на чтение самого каталога нет.

1.6 Процессы

Наряду с файлом, понятие *процесса* является важнейшим в концепции открытых операционных систем.

[34] Именно поэтому метафора «папка» для каталога является неприемлемой.

Процесс — это обладающая уникальным идентификатором единица исполняемого кода[35] в памяти.

Подавая простую команду из оболочки, оператор дает ОС указание запустить другой процесс. В ходе исполнения процесс может *порождать* другие процессы и проходить целый ряд *состояний*, некоторые из которых будут ниже описаны. Сама оболочка также является процессом, порожденным, как правило, процессом регистрации в системе, который, в свою очередь, как правило, порождается особым инициализационным процессом.

Подобно файлам, процессы в своем отношении друг к другу могут быть представлены в виде иерархии (дерева). В отличие от иерархии файлов, ребра этого дерева представляют не отношения вложенности, но отношения порождения («родитель-ребенок»). Процесс не может появиться в системе иначе, нежели будучи порожденным другим процессом, за очевидным исключением «корневого» процесса, запускаемого самим ядром при загрузке системы. Само ядро не является процессом[36].

Исследовать процессы можно стандартной командой «ps». Поданная без параметров, она выводит информацию о текущей оболочке и порожденных ею процессах.

```
[alice@wonderland alice]$ ps
  PID TTY          TIME CMD
 1956 tty1     00:00:00 bash
 2006 tty1     00:00:00 ps
```

Рис. 1-45

В выводе на Рис. 1-45 присутствуют четыре колонки. «PID» — это уникальный для системы *идентификатор процесса* (он устанавливается при порождении процесса и сохраняется неизменным до его завершения», «TTY» — *терминал*, с которого запущен процесс, «TIME» — *время процесса* (сумма квантов процессорного времени, потребленного процессом на момент «снимка» его состояния), «CMD» — *команда*,

---

[35] Формальное определение стандарта гласит, что процесс — «Адресное пространство с одним или более витком, исполняющимся в нем, и системными ресурсами, необходимыми для исполнения этих витков». Виток (thread, поток) в свою очередь определяется как «поток управления». Понятие витка в этом курсе нам не понадобится, поскольку отдельные витки недоступны оператору оболочки. Хотя стандарт и определяет процесс через виток, понятие витка является менее фундаментальной позднейшей добавкой к концепции открытых ОС и, по мнению многих авторитетных аналитиков, является лишь средством повышения производительности.

[36] В некоторых системах части кода ядра все же представлены в виде процессов, «усыновленных» инициализационным процессом.

подача которой привела к порождению процесса.

В данном случае Алиса получила информацию о двух процессах: оболочке «bash» и внешней команде «ps»[37].

Команда «ps -A» выводит информацию обо всех процессах в системе[38]. В примере на Рис. 1-46 мы, подав команду из эмулятора терминала, для наглядности использовали ключ «-A» вместе с ключом «-l» («эль»), задающим «длинный» формат вывода (с дополнительными полями) и нестандартным ключом «-H», представляющим с помощью отступов в поле «CMD» отношения между процессами (вывод немного сокращен).

```
[maksim@wonderland maksim]$ ps -AlH
F S   UID   PID  PPID  C PRI  NI ADDR    SZ  WCHAN TTY          TIME CMD
4 S     0     1     0  0  68   0    -   315 1446e2 ?        00:00:03 init
5 S     0  1162     1  0  68   0    -   312 1446e2 ?        00:00:00   apmd
5 S     0  1216     1  0  68   0    -   368 1210c6 ?        00:00:00   crond
4 S     0  1456     1  0  69   0    -   554 11c541 tty1     00:00:00   login
4 S   504  1775  1456  0  69   0    -   714 11c541 tty1     00:00:00     bash
0 S   504  1802  1775  0  69   0    -   488 11c541 tty1     00:00:00
startx
0 S   504  1810  1802  0  69   0    -   559 11c541 tty1     00:00:00
xinit
4 S     0  1811  1810  0  77   0    -  6353 1446e2 ?        00:01:51           X
0 S   504  1815  1810  0  69   0    -  1022 1446e2 tty1     00:00:02
blackbox
0 S   504  1891  1815  1  69   0    - 23938 1446e2 ?        00:03:05
soffice.bin
0 S   504  1941  1815  0  69   0    - 10426 144d2c ?        00:00:48
mozilla-bin
0 S   504  2030  1815  0  72   0    -  2069 1446e2 ?        00:00:00
xterm
0 S   504  2032  2030  0  70   0    -   715 11c541 pts/0    00:00:00
bash
0 R   504  2052  2032  0  74   0    -   625      - pts/0    00:00:00
ps
4 S     0  1457     1  0  69   0    -   550 11c541 tty2     00:00:00   login
4 S   505  1978  1457  0  69   0    -   687 164f79 tty2     00:00:00     bash
```

Рис. 1-46

Несколько иной набор параметров процесса можно получить, использовав вместо ключа «-l» ключ «-w», а ключ «-o» позволяет вывести для каждого процесса произвольный набор параметров из числа поддерживаемых системой, указав их мнемонику в качестве аргумента этого ключа.

Стандартом определено пятнадцать параметров, к которым могут добавляться параметры, специфичные для конкретной системы. Мы разберем лишь некоторые из них.

---

[37] Точность представления «снимка» в различных реализациях варьирует; так что не пугайтесь, если при подаче такой команды, допустим, ОС «Солярис», не увидите информации о «ps» — используемая при выводе таблица просто не успела обновиться.

[38] Обычному пользователю предоставление информации о чужих процессах может быть ограничено по соображениям безопасности.

UID — это идентификатор пользователя-владельца процесса. Как и у файла, у процесса есть владелец. В данном примере (при использовании ключа «-l») идентификатор выводится в числовом виде; если бы был задан ключ «-w», мы бы увидели, что числовому идентификатору 504 соответствует символический идентификатор «maksim», 505 — «alice». Числовой идентификатор 0 всегда соответствует главному пользователю «root».

Обычно UID наследуется от процесса-родителя. Исключение составляют процессы-оболочки, запускаемые программой регистрации — их UID соответствует идентификатору зарегистрировавшегося пользователя, хотя UID самой программы регистрации — 0.

Еще одно исключение — процессы, порожденные запуском программы из файла с установленным битом SUID. Их UID соответствует не породившему их процессу, а владельцу исполняемого файла. SUID (и подобный ему по эффекту бит GUID) — это мощный (и очень опасный) инструмент обхода системы распределения полномочий в ОС, поскольку позволяет пользователю запускать процессы с полномочиями выше собственных (в том числе, с полномочиями главного пользователя). Установить SUID бит может только главный пользователь. В аккуратно построенной и администрируемой системе количество программ с установленным SUID (и/или GUID) битом минимально.

В нашем примере этот механизм с очевидностью использован при запуске процесса «X» (Икс-сервер — основной компонент графической системы, предоставляющий в распоряжение Икс-клиентов (программ с графическим интерфейсом) виртуальный X-терминал, связанный с физическими видеоадаптером, клавиатурой, мышью и системным динамиком), чьим родителем является процесс «xinit» с UID равным 504. Существенно, что Икс-клиенты (процессы «blackbox», «soffice.bin», «mozilla-bin», «xterm») выполняются с обычным пользовательским UID.

PID, как мы уже знаем, это уникальный идентификатор процесса[39], а *PPID* — идентификатор его родителя. Обратите внимание на соответствие между PPID различных процессов в примере и расположением их в сформированном ключом «-H» «дереве».

TIME — время процесса — это совокупное количество процессорного времени, потребленного процессором на выполнение этого процесса за время его существования.

[39] Обратите внимание, что процессы, порожденные при подаче одинаковых команд (например «login» на tty1 и tty2 или «bash» на tty1, tty2 и pts/0), имеют разные идентификационные номера.

S — это состояние процесса. Запущенный процесс может находиться в одном из четырех стандартных состояний: «R» (выполняемый), «S» (ожидающий ввода-вывода), «T» (приостановленный — приостановку процессов мы обсудим ниже), «Z» («зомбированный», уже завершенный, но не успевший сообщить об этом процессу-родителю).

Итак, в примере на Рис. 1-46 мы видим:

- находящийся в корне дерева процесс «init»;
- два порожденных им процесса, не имеющих управляющего терминала: демон управления системой энергосбережения «apmd»[40] и демон периодического исполнения заданий «crond»;
- порожденный процессом init процесс «login» с tty1 в качестве управляющего терминала;
- порожденный этим процессом «login» процесс «bash» — экземпляр оболочки также с tty1 в качестве управляющего терминала;
- порожденный процессом «bash» процесс «xinit» (это сценарий, запускающий компоненты графической среды);
- порожденные процессом «xinit» процессы «X» (это сервер оконной системы X, он запущен в качестве демона, т.е. без управляющего терминала) и «blackbox» (это менеджер окон графической среды);
- порожденные процессом «blackbox» процессы «soffice.bin» (это словарный процессор «OpenWriter», в котором набирается данный текст), «mozilla-bin» (браузер «Мозилла»), «xterm» (эмулятор текстового терминала). Они не имеют управляющего терминала;
- порожденный эмулятором терминала процесс оболочки «bash» с псевдотерминалом pts/0, назначенным при запуске эмулятора терминала, в качестве управляющего;
- порожденный этой оболочкой процесс «ps», который и осуществил приведенный в примере вывод;
- порожденный процессом «init» процесс «login» и порожденный им процесс «bash» на терминале tty2.

---

[40] Собственно говоря, определить, что представляет собой тот или иной процесс, можно, подав команду «man» с именем команды из столбца CMD в качестве аргумента. В «ГНУ/Линукс» этот фокус не пройдет с квазипроцессами, которыми представлены некоторые компоненты ядра, поскольку они не имеют страниц руководства.

Управление заданиями и сигнализация процессов

В среде стандартной оболочки и команд открытой ОС запустить бесконечный процесс можно, введя команду «(while : ; do : ; done)», запускающую бесконечный цикл в подчиненном экземпляре оболочки (Рис. 1-47).

Пока не нужно беспокоиться о понятности синтаксиса управляющих конструкций.

```
[alice@wonderland alice]$ ( while : ; do  : ; done )
```

Рис. 1-47

Если Алиса все сделала правильно, то сценарий сам по себе уже не остановится никогда (скорее всего, до разгрузки системы). Приглашения оболочки Алиса тоже уже не получит, поэтому даже не сможет выйти из системы.

Справиться с этой ситуацией ей поможет клавиатурная комбинация Control-C. Как и комбинация Control-D, она не отображается на экране, но после ее нажатия Алиса получает приглашение оболочки и при помощи команды «ps» убеждается, что никаких процессов, кроме самой оболочки и «ps», под этой оболочкой не выполняется.

```
^C
[alice@wonderland alice] ps
  PID TTY          TIME CMD
 2635 pts/1    00:00:00 bash
 2666 pts/1    00:00:00 ps
[alice@wonderland alice]$
```

Рис. 1-48

Клавиатурная комбинация Control-C побуждает драйвер терминала отправить сигнал нормального завершения выполняемому процессу (в данном случае, подчиненной оболочке).

Клавиатурная комбинация Control-Z побуждает драйвер терминала отправить выполняемому процессу другой сигнал — приостановки[41].

[41] На самом деле сигнал приостановки и клавиатурная комбинация Control-Z не описаны в стандарте, но вы найдете эту возможность практически в любой открытой ОС.

```
[alice@wonderland alice]$ ( while : ; do  : ; done )
^Z
[1]+  остановлен              ( while : ; do  : ; done )
[alice@wonderland alice]$ ps -l
S   UID   PID  PPID  TTY          TIME CMD
S   505  2727  2722  pts/2    00:00:00 bash
T   505  2755  2727  pts/2    00:00:01 bash
R   505  2756  2727  pts/2    00:00:00 ps
```

Рис. 1-49

После нажатия Control-Z оболочка выдает сообщение, состоящее из числа в квадратных скобках, слова «остановлен» («stopped» в стандартной локали) и введенной ранее команды (Рис. 1-49).

Число в квадратных скобках — это номер *задания*. Заданием является любая начавшая выполняться простая команда.

Состояние соответствующего процесса (колонка «S» в выводе «ps -l» (эль)) обозначено буквой «T», означающей, что процесс *остановлен*. Задание, соответствующее такому процессу, также называется остановленным.

Возобновить исполнение задания можно двумя способами. Команда «fg» возобновляет выполнение задания *на переднем плане*, а команда «bg» — *на заднем плане* (или *в фоновом режиме*)[42]. Заданием переднего плана называется задание, завершения *ведущего процесса* (первого процесса, запущенного подачей команды) которого ожидает оболочка перед выводом очередного приглашения, и которое может свободно выводить данные на управляющий терминал и вводить их с терминала.

В любой момент времени на переднем плане каждого управляющего терминала находится не более одного задания. Заданий заднего плана может быть неограниченное[43] количество.

[42] В некоторых «доюниксовых» ОС понятие переднего и заднего планов выполнения было связано с приоритетами заданий или с выполнением одного из них в режиме «реального времени». В открытых ОС понятие приоритета и понятие плана выполнения взаимонезависимы.

[43] В разумных пределах.

```
[alice@wonderland alice]$ ( while : ; do  : ; done )
^Z
[1]+  остановлен              ( while : ; do  : ; done )
[alice@wonderland alice]$ fg
^Z
[1]+  остановлен              ( while : ; do  : ; done )
[alice@wonderland alice]$
bg[1]+ ( while : ; do  : ; done )
[alice@wonderland alice] ps -l
S   UID   PID  PPID  TTY          TIME CMD
S   505  2727  2722  pts/2    00:00:00 bash
R   505  2765  2727  pts/2    00:00:03 bash
R   505  2766  2727  pts/2    00:00:00 ps
```

Рис. 1-50

В примере на Рис. 1-50 Алиса запускает сценарий «цикл», останавливает его нажатием Control-Z, затем возобновляет его выполнение на переднем плане командой «fg», снова останавливает, и затем возобновляет на заднем плане командой «bg». После этого Алиса сразу получает приглашение и, введя команду «ps -l», видит соответствующий выполнению сценария процесс «bash» (PID 2765) с состоянием «R» («выполняемый»).

Нажатие клавиатурных комбинаций Control-C и Control-Z всегда вызывает передачу сигнала заданию переднего плана. Заданию заднего плана передавать сигнал можно только явно, для чего служит команда «kill». Указание в качестве ее единственного аргумента идентификатора процесса приводит к тому, что процессу передается сигнал «нормально завершиться» (это соответствует нажатию комбинации Control-C для задания переднего плана) (Рис. 1-51).

```
[alice@wonderland alice]ps -l
S   UID   PID  PPID  TTY          TIME CMD
S   505  2727  2722  pts/2    00:00:00 bash
R   505  2765  2727  pts/2    00:00:03 bash
R   505  2766  2727  pts/2    00:00:00 ps
[alice@wonderland alice]kill 2765
[1]+  завершен                ./цикл
```

Рис. 1-51

Подача команды «kill» с ключом «-s» и *идентификатором сигнала* в качестве параметра этого ключа позволяет подать процессу произвольный сигнал. Стандартом определены восемь сигналов, перечисленных в таблице на Рис. 1-52.

значение	идентификатор	смысл	
0 (ноль)	0	(зарезервировано)	
1	SIGHUP	отключение управляющего терминала	

значение	идентификатор	смысл	
2	SIGINT	запрос на прерывание	процессу переднего плана передается по нажатию Control-C
3	SIGQUIT	запрос на выход из программы	
6	SIGABRT	запрос на аварийное завершение	
9	SIGKILL	запрос на безусловное завершение	не может быть перехвачен процессом
14	SIGALRM	запрос на возобновление выполнения	
15	SIGTERM	запрос на нормальное завершение	умолчание команды «kill»

Рис. 1-52

Реализация может предусматривать большее их количество[44]. Практически во всех системах реализован сигнал SIGSTOP, его отправка процессу переднего плана большинством современных оболочек осуществляется нажатием Control-Z, как описано выше.

На пользовательском уровне применяются обычно сигналы SIGTERM и SIGKILL. Отличие их в том, что при получении первого из них процесс по возможности завершается «чисто»: сбрасывает содержимое внутренних буферов в файлы и закрывает их, а второго — завершается немедленно. Второй используется обычно для «убиения» процесса, выполняющего ошибочную программу.

До сих пор Алиса отправляла сигналы по собственной инициативе. Отправляющим процессом выступали оболочка (при передаче сигналов процессу переднего плана нажатием клавиш) или команда «kill». Но передача сигналов (межпроцессная коммуникация) может осуществляться между любыми процессами, и является широко используемым в системном и прикладном программировании механизмом ОС. Как и при доступе к файлам, при доступе к процессам ОС руководствуется системой распределения полномочий. Стандартное поведение проще, чем в случае с файлами: процесс, запущенный обычным пользователем (не

[44] Например, сигналы об аппаратных или системно-программных ошибках, получение которых, как правило, приводит к аварийному завершению процесса-получателя с записью файла дампа памяти (core) в домашний каталог пользователя-хозяина процесса. Список известных системе сигналов (их число может приближаться к сотне) можно получить по команде «kill -l», но смысла указанных там идентификаторв, как правило, приходится доискиваться в технической документации.

главным пользователем), может сигнализировать процессу, запущенному тем же пользователем, и не может сигнализировать процессу, которым «владеет» другой пользователь. В примере на Рис. 1-53 попытка Алисы «убить» процесс, принадлежащий другому пользователю, привела к сообщению об ошибке.

```
[alice@wonderland alice]$ ps -Al
F S   UID   PID  PPID  C PRI  NI ADDR    SZ  WCHAN TTY          TIME CMD
<...>
4 S   505  1569  1455  0  69   0    -   722 11c541 tty1     00:00:00 bash
4 S   504  2077  1456  0  69   0    -   716 164f79 tty2     00:00:00 bash
0 R   505  2113  2047  0  78   0    -   622      - tty1     00:00:00 ps
[alice@wonderland alice]$ kill 2077
-bash: kill: (2077) - Недостаточно прав
```

Рис. 1-53

Реализацией может быть определено более сложное поведение[45].

Программа может переопределить смысл сигналов, которые получает процесс (в частности, отменить завершение процесса), за исключением сигнала SIGKILL. Все стандартные команды ОС обрабатывают сигналы стандартным образом[46].

Если вам не удается завершить запущенный вами процесс подачей команды «kill -s SIGKILL», значит, в системе возникли *очень* серьезные неполадки.

Вы также можете столкнуться с ситуацией, когда программа переопределяет SIGINT, SIGSTOP и входит в бесконечный цикл (или ожидает события, наступления которого в обозримом будущем не предвидится). Запустив такую программу на переднем плане, вы не сможете завершить ее нажатием Control-C или приостановить нажатием Control-Z, а подать SIGKILL командой «kill» также будет невозможно, поскольку оболочка ожидает завершения процесса переднего плана.

Простого выхода из этой ситуации нет, но обычно можно зарегистрироваться на другом терминале (включая виртуальную консоль) и «убить» хитрый процесс командой «kill -s SIGKILL». Если таким образом «завис» сеанс в окне виртуального терминала, его обычно можно «убить» средствами оконного менеджера (закрыв окно). Наконец, если вы работаете на последовательном терминале, можно попытаться выключить и снова включить его. Оболочка по выключении получит сигнал SIGHUP и «убьет» подчиненные себе процессы.

---

[45] На самом деле современные ОС ставят в соответствие процессам каталоги в фиктивной файловой структуре, смонтированной в каталоге «/proc/», и управляют доступом к процессам на основании правомочий, сопоставленных этим «каталогам» и входящим в них «файлам».

[46] Хотя экранные команды («more», «vi») перехватывают нажатие Control-C.

Сложные команды и задания

Чтобы запустить задание на заднем плане, не обязательно запускать его на переднем плане, приостанавливать и возобновлять командой «bg». Можно воспользоваться *символом завершения* команды «&» (читается «амперсенд») (Рис. 1-54).

```
[alice@wonderland alice]$ cat &
[1] 2198

[1]+  остановлен              cat
[alice@wonderland alice]$
```

Рис. 1-54

Использование символа завершения «&» позволяет ввести в одной строке более одной команды (и, соответственно, запустить более одного задания), связав их этим символом (Рис. 1-55). Список заданий можно получить командой «jobs».

```
[alice@wonderland alice]$ cat & cat &
[2] 2199
[3] 2200

[2]+  Stopped                 cat

[3]   Stopped                 cat
[alice@wonderland alice]$ jobs
[1]-  Stopped                 cat
[2]+  Stopped                 cat
[3]   Stopped                 cat
```

Рис. 1-55

Знак «+» после номера задания обозначает «текущее» задание, то есть задание, которым можно управлять командами «bg» и «fg» без аргументов. Знаком «-» помечено «предыдущее» задание (которое станет текущим по завершению текущего). При переводе задания переднего плана на задний или запуске нового задания текущее задание (если оно есть) становится «предыдущим», а вновь запущенное — текущим.

Если нужно возобновить исполнение задания на переднем или заднем плане, или перевести задание на передний план, можно воспользоваться командами «fg» и «bg», задав им аргумент, состоящий из «%» («процент»), и следующим за ним слитно номером задания (Рис. 1-56).

```
[alice@wonderland alice]$ fg %3
cat
```

Рис. 1-56

Другим символом завершения команды, также позволяющим подать более одной команды и инициировать более одного задания одной строкой является «;» (точка с запятой). Команда, завершенная этим символом, бу-

дет выполняться на переднем плане, а последующие команды (задания) будут выполняться после ее завершения.

При подаче сложных команд (команд, включающих в себя более одной команды) их можно группировать с использованием круглых скобок. Сгруппированные скобками команды (вне зависимости от использованного символа завершения) образуют *группу команд*[47], выполняемых как одно задание (Рис. 1-57).

```
[alice@wonderland alice]$ (cat & cat) &
[1] 2220
[alice@wonderland alice]$
[1]+  Stopped                 ( cat & cat )

[alice@wonderland alice]$ jobs
[1]+  Stopped                 ( cat & cat )
```

Рис. 1-57

Кроме того, группировка бывает полезна, чтобы передать значение переменной сразу всем командам, или перенаправить ввод и/или вывод всех команд.

Механизм управления заданиями (являющийся позднейшей добавкой к концепции открытых систем) чрезвычайно полезен при выполнении сложных работ с алфавитно-цифрового терминала. Сегодня операторы предпочитают при возможности запускать разные программы в разных окнах виртуальных терминалов в графической среде. Тем не менее, полезно хотя бы в общих чертах представлять, что это такое.

Существуют также символы завершения команды «&&» и «||». Их действие связано с понятием *кода завершения*, возвращаемого каждой командой. Код завершения определяется программой, но обычно успешно выполнившаяся команда возвращает код «0» (ноль), а выполнившаяся с ошибкой — числовое значение кода ошибки. Явную работу с кодами завершения мы обсудим при введении элементов программирования оболочки, а здесь лишь упомянем, что символ завершения «&&» означает, что заданную за ним команду следует выполнить только в случае, если указанная перед ним команда выполнилась успешно, а символ «||» — наоборот, что «правую» команду следует выполнить только при ошибочном завершении «левой» (Рис. 1-58).

[47] В терминах стандарта «группа команд» является полным синонимом «задания», так что задание, состоящее из одной простой команды, тоже называется «группой».

```
[alice@wonderland каталог]$ ls файл && date
ls: файл: No such file or directory
[alice@wonderland каталог]$ touch файл
[alice@wonderland каталог]$ ls файл && date
файл
Пт. Июл 18 04:59:44 MSD 2003
[alice@wonderland каталог]$ ls файл || date
файл
[alice@wonderland каталог]$ rm файл
[alice@wonderland каталог]$ ls файл || date
ls: файл: No such file or directory
Пт. Июл 18 04:59:47 MSD 2003
```

Рис. 1-58

1.7 Переменные

До сих пор мы имели дело с параметрами, передававшимися команде в виде аргументов, следующих за именем команды. Команда интерпретирует аргументы исходя их их значений (так, большинство команд считает аргумент, начинающийся с дефиса, ключом) и их позиции (так, команды «cp» и «mv» последний операнд считают целевым файлом или каталогом, а предшествующие — источниками), поэтому аргументы называют еще *позиционными параметрами*.

В открытых системах существует еще один механизм передачи параметров — *переменные*. В отличие от аргументов, переменные являются *именованными параметрами* и их семантика определяется не их позицией и значением, но именем.

```
[alice@wonderland alice]$ ls нет_такого_файла
ls: нет_такого_файла: Такого файла или каталога нет
[alice@wonderland alice]$ LC_ALL=C ls нет_такого_файла
ls: нет_такого_файла: No such file or directory
```

Рис. 1-59

В примере на Рис. 1-59 Алиса сначала подает команду «ls» с несуществующим файлом в качестве аргумента и получает сообщение об ошибке на русском языке. Затем она подает ту же команду, предварив ее конструкцией «LC_ALL=C», и получает сообщение о той же ошибке на английском языке.

Конструкция, состоящая из *имени переменной* и ее *значения*, разделенных знаком равенства («=») без промежутков, и является определением параметра-переменной для вызываемой команды. В данном случае определяется переменная «LC_ALL», которой присваивается значение «C». Переменная «LC_ALL» является одной из стандартных, ее значение определяет язык и другие национально-специфические особенности интерфейса (эту и несколько других *переменных локали* мы подробнее рассмотрим ниже).

Передача переменной команде таким способом не оказывает никакого влияния на поведение последующих команд. Также она не оказывает влияния на поведение оболочки.

Если нам нужно изменить подобным образом (получать сообщения на английском языке) самой оболочки, следует установить значение переменной такой же конструкцией из имени и значения переменной, разделенных знаком равенства, за которыми не следует никакой команды. Присвоение значения переменной командой само по себе не влияет на поведение вызываемых из оболочки команд, то есть эта переменная им *не передается* (Рис. 1-60).

```
[alice@wonderland alice]$ нет_такой_команды
-bash: нет_такой_команды: команда не найдена
[alice@wonderland alice]$ LC_ALL=C
[alice@wonderland alice]$ нет_такой_команды
-bash: нет_такой_команды: command not found
[alice@wonderland alice]$ ls нет_такого_файла
ls: нет_такого_файла: Такого файла или каталога нет
```

Рис. 1-60

Чтобы значение переменной передавалась всем вызываемым командам, ее следует сделать передаваемой (*экспортировать* ее) командой «export» с именем переменной в качестве аргумента (см. Рис. 1-61).

```
[alice@wonderland alice]$ LC_ALL=C
[alice@wonderland alice]$ export LC_ALL
[alice@wonderland alice]$ нет_такой_команды
-bash: нет_такой_команды: command not found
[alice@wonderland alice]$ ls нет_такого_файла
ls: нет_такого_файла: No such file or directory
```

Рис. 1-61

Установленное значение (вне зависимости от того, экспортировано ли оно) сохраняется до конца сеанса работы с оболочкой, до его переустановки или до уничтожения переменной командой «unset» с именем переменной в качестве аргумента.

Смысл механизма переменных как минимум двояк. Во-первых, крайне удобна возможность единообразного изменения поведения определенной группы команд, команд, поданных в течение определенного сеанса, или команд в сеансах определенного пользователя (механизм ключей для этого, очевидно, слишком громоздок). Во-вторых, язык оболочки — не только язык интерактивного взаимодействия с системой, но и императивный язык программирования.

Установка значения переменной вполне соответствует оператору присвоения в большинстве интерпретируемых императивных языков программирования. Переменные в оболочке всегда имеют строчный тип,

хотя семантика некоторых команд может накладывать ограничения на значения переменных.

Запуская новый процесс (или группу процессов) при подаче команды, система передает ему копии значений всех экспортированных переменных, дополняя или заменяя их именованными параметрами, с которыми подана команда (если таковые имеются). Совокупность этих переменных называется *окружением* процесса. Обратное наследование (передача процессом переменных родительскому) в открытых системах отсутствует.

Переменная будет *раскрыта* оболочкой (подобно тому, как оболочка раскрывает значения специальных символов в именах файлов), если указать ее в любом месте любой команды в окружении фигурных скобок «{» и «}» предваренной знаком денежной единицы «$». В большинстве случаев (когда не возникает неоднозначности в интерпретации) фигурные скобки можно опустить (Рис. 1-62).

```
[alice@wonderland alice]$ echo ${LC_ALL}
C
[alice@wonderland alice]$ echo $LC_ALL
C
```

Рис. 1-62

Получить список переменных, установленных в данный момент времени, можно командой «set» без аргументов, а список экспортированных переменных — командой «env» без аргументов. В типичной системе при запуске оболочки устанавливаются значения нескольких десятков переменных, большинство из которых сразу экспортируются. В примере на Рис. 1-63 эти списки сильно сокращены. С семантикой отдельных переменных мы познакомимся ниже.

```
[alice@wonderland alice]$ set
COLUMNS=80
HOME=/home/alice
LANG=ru_RU.KOI8-R
LANGUAGE=ru_RU.KOI8-R
LINES=24
LOGNAME=alice
MAIL=/var/mail/alice
MAILCHECK=60
PATH=/home/alice/bin:/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/usr/games
PS1='[\u@\h \W]\$ '
PS2='> '
PS4='+ '
PWD=/home/alice
TMPDIR=/home/alice/tmp
USER=alice
[alice@wonderland alice]$ env
TMPDIR=/home/alice/tmp
MAIL=/var/mail/alice
PATH=/home/alice/bin:/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/usr/games
PWD=/home/alice
LANG=ru_RU.KOI8-R
LANGUAGE=ru_RU.KOI8-R
LOGNAME=alice
```

Рис. 1-63

Стандартом определен ряд переменных, имеющих фиксированную семантику, значения которых используются стандартными командами. Они представлены в таблице на Рис. 1-64. В большинстве случаев пользователю не нужно устанавливать их значения самостоятельно.

COLUMNS	количество столбцов на экране терминала
HOME	полное имя домашнего каталога пользователя
LINES	количество строк на экране терминала
LOGNAME	регистрационное имя пользователя
PATH	список каталогов, содержащих команды
PWD	полное имя текущего каталога
SHELL	полное имя оболочки по умолчанию
TMPDIR	полное имя каталога для временных файлов пользователя
TERM	тип терминала
TZ	часовой пояс

COLUMNS	количество столбцов на экране терминала
LANG, LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, LC_TIME, NLSPATH	*локаль*

Рис. 1-64

Кроме того, стандарт не рекомендует произвольно переопределять еще ряд переменных: *ARFLAGS, CC, CDPATH, CFLAGS, CHARSET, DEAD, EDITOR, ENV, EXINIT, FC, FCEDIT, FFLAGS, GET, GFLAGS, HISTFILE, HISTORY, HISTSIZE, IFS, LDFLAGS, LEX, LFLAGS, LINENO, LISTER, LPDEST, MAIL, MAILCHECK, MAILER, MAILPATH, MAILRC, MAKEFLAGS, MAKESHELL, MANPATH, MBOX, MORE, MSGVERB, PROC, OLDPWD, OPTARG, OPTERR, OPTIND, PAGER, PPID, PRINTER, PROCLANG, PROJECTDIR, RANDOM, SECONDS, TERMCAP, TERMINFO, USER, VISUAL, YACC, YFLAGS*. Некоторые из них используются самой оболочкой, некоторые — стандартными командами, а некоторые — прикладными и инструментальными программами.

Локаль, «и17я» и «л9я»

Локалью («locale» — «местонахождение») называется совокупность переменных, управляющих поведением оболочки, команд и других программ в части языковых и национально-культурных особенностей. Локалью также называются и значения, которые принимают эти переменные (кроме NLSPATH).

В любой стандартной ОС определены (совпадающие) локали «POSIX» и «C». Эта локаль называется «системной» и во всех известных нам системах она соответствует соглашениям, принятым в США[48].

В ОС может быть также определено произвольное количество локалей, именуемых следующим образом: двухбуквенное ИСО-сокращение названия страны, за которым слитно следует знак подчеркивания «_» и — слитно же — двухбуквенное ИСО-сокращение названия языка. Далее могут следовать (также слитно) точка и наименование кодовой таблицы.

[48] Строго говоря, стандарт не говорит об этом прямо, но ограничивает набор символов стандартной локали семибитной латиницей, так что кроме американского или британского английского полностью реализовать стандартную локаль, не прибегая к транслитерации, можно разве что на латыни.

Для русского языка и российских культурных особенностей значением локали будет «ru_RU.KOI8-R» или «ru_RU.ISO8859-5».

Текущую локаль можно узнать, подав команду «locale». Обычно всем переменным локали (кроме NLSPATH), перечисленным на Рис. 1-65, присваивается одно и то же значение (это можно сделать, установив значение всего лишь одной переменной, LC_ALL). Однако бывают и другие случаи: например, иностранный сотрудник или студент может предпочесть сообщения и диалоги на родном языке, а остальные национально-культурные параметры — соответствующими стране пребывания.

LC_CTYPE	классификация символов и преобразование регистра
LC_COLLATE	порядок объединения многозначных символов
LC_MONETARY	формат информации о денежных суммах
LC_NUMERIC	формат числовой неденежной информации
LC_TIME	формат времени и даты
LC_MESSAGES	формат информационных, диагностических и диалоговых сообщений
LC_ALL	переменная, перекрывающая значение остальных переменных LC_*
LANG	язык сообщений
NLSPATH	путь к локализационным ресурсам

Рис. 1-65

Интернационализованной называется программа (включая стандартные команды), корректно изменяющая свое поведение в соответствии с переменными локали.

Локализованной для определенной локали называется программа, для которой существуют (если они необходимы) специфические языковые и культурные ресурсы.

Для длинных слов «localization» и «internationalization» иногда даже в формальных документах используются сокращения «l10n» и «i18n» (цифры между первой и последней буквой образуют число пропущенных букв).

Стандартный набор команд (и многие дополнительные программы), входящие в состав популярных дистрибутивов открытых систем, в массе

своей интернационализованы, но с русской локализацией ситуация весьма неоднозначна. Вы столкнетесь с ситуациями, когда перевода того или иного ресурса (например, сообщений об ошибках и диагностики) для определенной программы не окажется — корректно интернационализованная программа «откатится» к системной локали «C»[49].

Вы также можете столкнуться и с некорректно интернационализованными программами (такое случается с прикладным кодом, перенесенным с альтернативных платформ, или разработанным без оглядки на локаль), которые игнорируют локаль, ведут себя странно или отказываются работать при локали, отличной от «C»[50].

К сожалению, при применении наиболее популярного сегодня «IBM PC-совместимого» оборудования (т.е. компьютеров на основе процессоров архитектуры IA-32) реальная локализация аппаратно-программной системы не сводится к установке локали. Для возможности ввода нелатинских символов необходимо назначить код переключения систем письменности какой-либо клавише. Кроме того, чтобы терминал отображал нелатинские символы в текстовом режиме, в видеоадаптер должен быть подгружен соответствующий шрифт (при применении эмулятора терминала в графическом режиме этой проблемы нет). Стандартного способа осуществления этих действий нет, каждая ОС решает эти задачи по-своему.

Еще одна локализационная сложность связана с ограничением восьмибитной кодировки: размер одного набора символов ограничен 256 и, соответственно, за вычетом управляющих символов и символов псевдографики места в нем хватает только на обычную и расширенную латиницу, обычную латиницу и русскую кириллицу или обычную латиницу и украинскую кириллицу.

Работать с текстами, включающими одновременно большее количество символов, можно, используя кодовые таблицы Юникод (стандарт ИСО/МЭК 10646) и кодировку UTF8. Однако не все ОС поддерживают корректную работу текстовых утилит при переменной длине символов. Кроме того, применять при этом вшитый знакогенератор видеоадаптера уже не представляется возможным, и для терминального режима нужен встроенный в драйвер программный знакогенератор. Проблемы, связан-

[49] Программы, входящие в утилиты ГНУ, составляющие большую часть системных программ в «ГНУ/Линукс», заметную — в ОС семейства «БСД» и часто используемые в других ОС, реализуют более продвинутую стратегию. Если определена переменная LANGUAGE, она будет интерпретироваться как список локалей (разделенных двоеточием) в порядке их предпочтения, в соответствии с которым программа будет искать локализационные ресурсы.

[50] В последнем случае можно явным образом передать этой конкретной программе значение переменной LC_ALL, равное «C» или «POSIX».

ные с UTF8-локалью, должны быть решены в современных ОС в течение ближайших лет.

Команда как файл и переменная $PATH

Обратите внимание на *переменную $PATH* (у Алисы ее значение оказалось равным «/home/alice/bin:/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/usr/games»). Значение переменной $PATH интерпретируется оболочкой как список имен каталогов, разделенных двоеточиями. Когда оператор вводит команду, оболочка просматривает эти каталоги в поисках исполняемого файла с именем, совпадающим с именем введенной команды.

Большинство стандартных команд ОС реализовано в виде отдельных программ (исключение составляют так называемые *встроенные команды*[51]). Чтобы оболочка нашла и запустила соответствующую программу, путь к ней (т.е. имя каталога, в котором содержится эта программа) должен содержаться в списке, составляющем переменную $PATH. Текущий каталог не исключение — чтобы поиск программы осуществлялся и в нем, он должен в явном виде присутствовать в этом списке[52].

```
[alice@wonderland alice]$ echo $PATH
/home/alice/bin:/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/usr/games
[alice@wonderland alice]$ PATH=".:$PATH"
[alice@wonderland alice]$ echo $PATH
.:/home/alice/bin:/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/usr/games
```

Рис. 1-66

В приведенном на Рис. 1-66 примере Алиса проверяет значение $PATH, затем добавляет в начало списка текущий каталог и убеждается, что значение переменной приняло искомую форму. (Этот пример демонстрирует особенности экранирования, выполняемого двойными кавычками. Они экранируют пробелы, но, в отличие от апострофов, не мешают раскрытию специальных символов и имен переменных).

Хотя стандартом не определены каталоги, в которых содержатся исполняемые файлы команд, в большинстве современных ОС основным таким

---

[51] Встроенные команды исполняются самой оболочкой, без вызова внешних программ. Пятнадцать команд («break», «:», «continue», «.», «eval», «exec», «exit», «export», «readonly», «return», «set», «shift», «times», «trap», «unset») являются *специальными встроенными*, их должна реализовать любая оболочка. Кроме того, разработчик оболочки может по каким-либо соображениям реализовать любую другую команду (стандартную или нестандартную) как встроенную. Пользователю в большинстве случаев безразлично, является ли команда встроенной или внешней, но при начале использования новой оболочки полезно ознакомиться со списком встроенных в нее команд.

[52] Это уточнение приведено специально для мигрантов с MS-DOS.

каталогом является «/usr/bin/».

Добавление в PATH текущего каталога считается весьма легкомысленным; по крайней мере, суперпользователь никогда не должен этого делать[53]. Удобным является присутствие в PATH каталога «~/bin/» (в нашем примере «/home/alice/bin/»). В него пользователь может помещать собственные сценарии и другие программы.

Команду, соответствующую программе, чей код размещен вне путей, перечисленных в PATH, можно издать, указав полное или относительное имя файла (например, команда «./моя_программа -o» запустит программу, содержащуюся в файле «моя_программа» в текущем каталоге). Файл должет быть исполняемым.

1.8 Конвейер

Помимо использования символов завершения («&», «;», «&&» и «||») и скобок «(» и «)», в открытых ОС имеется еще один механизм объединения простых команд в группу — конвейер[54].

В отличие от групп команд, объединенных упомянутыми символами завершения и независимых друг от друга, команды, входящие в конвейер, связаны передачей данных.

В следующем примере Алисе по каким-то причинам нужно получить пронумерованный список файлов в одном из каталогов. У команды «ls» нет соответствующего ключа, хотя есть ключ «-1» (единица), позволяющий осуществить вывод списка в «коротком» формате по одному файлу на строку. Но в системе имеется стандартная команда «nl», выводящая строки ввода, предваренные их номерами.

Алиса может перенаправить вывод команды «ls -1» во временный файл, затем перенаправить ввод команды «nl» из того же файла и, наконец, удалить его (Рис. 1-67).

---

[53] В текущем каталоге случайно или вследствие чьей-то «шутки» может оказаться исполняемый файл, совпадающий по имени со стандартной или административной командой системы, но выполняющий другую функцию.

[54] «Pipe». В русской литературе встречается также перевод «канал».

```
[alice@wonderland каталог]$ ls
five  four  one  three  two
[alice@wonderland каталог]$ ls -1 >временный
[alice@wonderland каталог]$ nl <временный
     1  five
     2  four
     3  one
     4  three
     5  two
     6  временный
[alice@wonderland каталог]$ rm временный
```

Рис. 1-67

Это достаточно громоздко; кроме того, в вывод попало и имя самого временного файла, что в планы Алисы не входило[55].

Конвейер — это соединение двух или более команд символом «|» («вертикальная черта», «пайп»). При связывании команд конвейером, вывод указанной слева становится вводом указанной справа, *без* каких-либо временных файлов (Рис. 1-68).

```
[alice@wonderland каталог] ls -1 | nl
     1  five
     2  four
     3  one
     4  three
     5  two
```

Рис. 1-68

В конвейер могут быть связаны и более двух команд. В примере на Рис. 1-69 Алиса передает, как и ранее, вывод команды «ls» команде «nl», а вывод «nl» передает команде «sort -r», которая (с этим ключом) выводит свой ввод, отсортированный в обратном порядке.

```
[alice@wonderland каталог]$ ls -1 | nl | sort -r
     5  two
     4  three
     3  one
     2  four
     1  five
```

Рис. 1-69

Ввод (но не вывод) первой команды в конвейере может быть перенаправлен из файла посредством символа «<», а вывод (но не ввод) последней — перенаправлен в файл или в конец файла символами «>», «>>». Перенаправлять стандартный ввод или вывод команд, окруженных символами конвейера с обеих сторон, бессмысленно[56], хотя можно (и бы-

[55] Возможно, стоит ей напомнить, что перенаправление выполняется оболочкой, которая при необходимости создает целевой файл, так что он в любом случае уже будет присутствовать в каталоге при выполнении команды «ls». Конечно, она может выкрутиться, создав файл в другом каталоге.

[56] Реальные оболочки при попытке перенаправить ввод или вывод одновременно в

вает полезно) перенаправить в файл вывод их ошибок («2>» или «2>>»).

В некоторых случаях бывает все-таки необходимо вывести «сечение» конвейера в определенной точке. Для этого служит команда «tee», копирующая ввод в вывод и параллельно записывающая его в файл, имя которого указано в качестве ее аргумента. При необходимости осуществить вывод на терминал, необходимо в явном виде указать его имя (/dev/tty) (см. Рис. 1-70).

```
[alice@wonderland каталог]$ ls -1 | nl | tee /dev/tty |
sort -r >временный
     1  five
     2  four
     3  one
     4  three
     5  two
```

Рис. 1-70

Приведенный пример повторяет уже встречавшуюся цепочку из трех команд, но итоговый вывод перенаправлен в файл «временный», а между командами «nl» и «sort» вставлена команда «tee /dev/tty», копирующая поток конвейера на текущий терминал. На терминал, соответственно, выводится последовательность строк после их нумерации, но до ее обратной сортировки.

Вне зависимости от включения конвейера в скобки, все запускаемые при этом процессы входят в одну группу процессов (одно задание).

Введение механизма конвейера и его эффективная реализация в ранних версиях ОС «Юникс» революционализировали практику программирования, в том числе, системного. Именно благодаря наличию этого механизма на уровне ОС стала возможна аккуратная декомпозиция реальных задач на относительно простые фрагменты-утилиты.

В «доюниксовых» системах пришлось бы предвидеть возможность того, что какой-нибудь Алисе понадобится нумерация строк и обратная сортировка списка файлов в каталоге, и включать соответствующие ключи в реализацию команды «ls». Что еще хуже, их реализацию пришлось бы включать в каждую программу, построчно обрабатывающую тексты (или убедить Алису в том, что «на самом деле такая возможность не нужна»)[57].

файл и другую команду ведут себя странно.

[57] И загромождение команд ключами, и произвольное ограничение возможностей пользователя свойственны и «доюниксовым» системам, и большинству сегодняшних альтернативных ОС.

Декомпозицией задач и предоставлением «тезауруса» отдельных команд вкупе с механизмами их связывания и объясняется простота и элегантность открытых ОС. Поскольку большинство команд вводят и/или выводят текст, существует практически неограниченная возможность их комбинации, отвечающей как предвиденным, так и непредвиденным разработчиками ситуациям.

Наличие таких механизмов и «дешевизна» (в терминах потребления компьютерных ресурсов) их применения обусловили складывание вокруг открытых систем культуры разработки (не только системной, но и прикладной), частью которой являются принципы:

1) минимизации функциональности отдельных программ,

2) простоты форматов ввода-вывода и

3) реализации программ в виде фильтров (то есть, преобразующих стандартный ввод в стандартный вывод) везде, где это возможно.

Наиболее очевидным примером реализации этих принципов является набор текстовых утилит, входящих в стандарт, некоторые из которых («cat», «nl», «sort») были уже бегло рассмотрены нами.

1.9 Элементы обработки текста

Обработка «плоских» (неразмеченных) текстов — одно из первых (после собственно вычислительных задач и управления приборами), очень важное и хорошо исследованное приложение компьютера.

Работа с текстами критична для многих других пользовательских приложений. Электронная почта — это текст. Форматы разметки, посредством которых представлены форматированный текст, векторная графика, ноты и т.п. (практически все данные, за исключением растровой графики, волнового представления звука и видеодорожек) — в основе своей также текст.

Даже картинки и звуковые файлы, размещенные в WWW, передаются незаметно для пользователя между машинами в закодированной текстом форме, хотя в данном случае текст и не является «собственной» формой представления данных.

Команды, с помощью которых пользователь «общается» с системой — это текст. Сколько бы не популяризовали и не навязывали графические интерфейсы, для серьезной и продуктивной работы, как правило, не обойтись без полноценного текстового диалога, так же как при серьезном и предметном разговоре сложно обойтись жестами и ответной мимикой.

Эффективная работа с текстом критична и для развития самих вычислительных и коммуникационных систем, поскольку сами программы в исходной своей форме — тексты. Для программ на интерпретируемых языках тексты являются и исполняемой формой, так что такие программы — тексты вдвойне (а типичная стандартизованная ОС почти наполовину состоит из «сценариев», т.е. программ, написанных на интерпретируемых языках).

(Хотя существуют и исключения. Например, электронные таблицы — специфические программы (определяющие порядок вычислений и способ представления их результатов) — представляют собой размеченный, а не «плоский» текст. В некоторых случаях файлы настроек — тоже программы в широком смысле этого слова — представлены не текстом, а базой данных более сложной структуры. Программы с графическим интерфейсом могут содержать значительные фрагменты, первичной формой представления которых является нетекстовая.)

Приемы работы с текстом — неотъемлемая часть компьютерной грамоты, но слишком часто она оказывается не освоенной вовремя. К сожалению, зачастую в курсе средней школы знакомство с обработкой «плоских» текстов ограничивается встроенными редакторами в среде программирования и электронно-почтовой программе, а навыки — простейшими приемами набора и исправления. Более абстрактные и всеобщие операции изучаются как часть word-процессинга, и хотя иногда при этом и демонстрируются возможности встроенных в word-процессоры языков программирования, область обработки текстов остается «вещью в себе» и никак не интегрируется с другими областями, осваиваемыми в курсе информатики.

Открытые ОС предоставляют достаточно широкий инструментарий работы с текстовыми данными, включая интерактивное редактирование и потоковую обработку. Они важны как в системном, так и в прикладном плане. В частности, администрирование операционной системы в значительной части представляет собой текстовое редактирование сценариев и файлов с данными.

С некоторыми командами потоковой обработки (такими, как «cat», «nl») мы коротко познакомились выше.

Размеченный и «плоский»

Водораздел между текстовыми редакторами и word-процессорами[58], про-

[58] Не следует, как это случается с некоторыми журналистами, путать word-процессоры («словарные процессоры») с текстовыми процессорами. Словарный процессор, подобно текстовому редактору, предназначен для интерактивной рабо-

ходит по способу отображения размеченного (имеющего некоторые атрибуты, такие, как цвет, начертание и кегль (размер) символов, выключка (выравнивание) и расположение абзацев, оформление страницы и т.п.) текста.

Word-процессор определенным образом интерпретирует разметку, визуализуя указанные атрибуты.

Текстовый редактор отображает размеченный текст «как есть» (с тегами разметки), хотя он вполне может быть «в курсе» синтаксиса языка разметки (если текст размечен, например, на языке XML, HTML или TeX) или даже самого текста (например, синтаксиса языка программирования) и каким-то образом его учитывать (например, расцвечивать теги разметки HTML или зарезервированные слова Pascal). В принципе, он может быть «в курсе» синтаксиса, грамматики и лексики даже естественного языка, хотя для работы с естественными языками «плоский», неразмеченный текст используется все реже, или, по крайней мере, неразмеченный текст все реже используется как первичная форма представления текста на естественном языке.

Отказаться вовсе от работы с «плоским» текстом затруднительно по давно известной эргономистам причине: использование визуализации «позволяет демонстрировать лишь результат форматирования, по нему невозможно определить задачи форматирования, поставленные пользователем системе. Например, если пользователь замечает, что система не делает переносов ... невозможно определить ... является ли это простым совпадением или же при форматировании данной главы перенос запрещен» (Т.Робертс, «Текстовые редакторы» // «Человеческий фактор». Т. 6. — М.: «Мир», 1992).

Редакторы

Когда компьютеры были большими и дорогими, задача редактирования программ и других текстов решалась гораздо более простыми устройствами, обычно состоявшими из телетайпа (или клавиатуры) и перфоратора, фиксирующего вводимый текст на картах или ленте. Программы и данные записывались первоначально на бумаге и тщательно проверялись вручную: синтаксическая ошибка или ошибка формата могла обойтись в лишний прогон, зачастую это означало бесплодно потраченные дорогие

ты с текстом. Примеры word-процессоров: «OpenOffice.org Writer», «Microsoft Word», «WordPerfect». Текстовые процессоры — средства программированной (неинтерактивной) обработки текста, часто связанной с изменением формата разметки. Примеры текстовых процессоров: «TeX», «troff», превращающие особым образом размеченный для типографского набора текст в данные для отображения на экране или на принтере (например, в форматах DVI или PostScript).

часы машинного времени.

С удешевлением компьютерной техники и разработкой многопользовательских систем появилась возможность посадить оператора за подключенный к машине телетайп, где он свободно вводил и исправлял текст, а компьютер тратил основную часть своих ресурсов на обслуживание других пользователей или выполнение долгих пакетных заданий. Для удобства операторов (часто ими оказывались сами программисты) разрабатывались *программы редактирования текстов* (или, попросту, текстовые редакторы), как правило, выводившие текст построчно и ожидавшие клавиатурной команды (зачастую на особом изощренном языке), сообщающей, следует ли оставить строку неизменной, либо внести в нее какие-то изменения.

Следующий шаг был сделан, когда телетайп (электрическую пишущую машинку) в качестве терминального устройства сменил дисплей с электронно-лучевой трубкой. Это превратило текст перед глазами оператора в динамический и позволило совершить революцию в редактировании текстов, внедрив так называемые «полноэкранные» (или, просто «экранные») редакторы, взаимодействуя с которыми оператор получил возможность, хотя и манипулируя клавиатурой, применять технику, скорее похожую на приемы работы с листом бумаги писателя, возвращающегося к ранее написанному, стирающего и исправляющего текст нелинейно.

«Vi» и «Emacs»

Два, по-видимому, первых экранных редактора, созданных в начале семидесятых, и явились родоначальниками «семейств» таких программ, до сих пор наиболее популярных в профессиональной среде. Это «vi» (читается «ви-ай») Билла Джоя (тогда аспиранта Университета Калифорнии в Беркли, а затем основателя Sun Microsystems) и «Emacs» (читается «имакс») Ричарда Столлмена (тогда сотрудника Лаборатории искусственного интеллекта Массачуссетского технологического института, а ныне — президента Фонда свободного программного обеспечения и лидера проекта GNU). Оба они, по сути, происходят от экранных режимов работы популярных тогда редакторов «ed» и «TECO», соответственно.

Первый ныне стандартизован и, в той или иной реализации (наиболее популярна, видимо, «vim» Брама Мооленаара) доступен в составе любой стандартной операционной системы (в том числе, свободных), а также — отдельно — для многих альтернативных ОС. Второй под названием «GNU Emacs» поддерживается Фондом свободного ПО (www.fsf.org) и выдержал уже более двадцати изданий (релизов), он обычно входит в поставки ОС «ГНУ/Линукс» и доступен для прочих (открытых и альтер-

нативных) ОС.

Исходная идеология и эргономическая модель этих двух выдающихся разработок несколько различается, что служит поводом для шутливой «священной войны» между их приверженцами. «vi(m)» относится к так называемым «многорежимным» редакторам. *В режиме редактирования* оператор вводит и исправляет текст. Перемещение по тексту, контекстный поиск и замена, более сложные операции выполняются в *командном режиме*. Между этими режимами (а также редко применяющимся *режимом построчного редактирования*) нужно явное переключение нажатием клавиатурной комбинации. Зато большинство команд привязаны к нажатиям одной клавиши, и даже перемещаться по тексту в командном режиме можно, не сбрасывая кисть руки на дополнительную клавиатуру со стрелками, а нажимая алфавитные клавиши в центре клавиатуры. Адепты «vi» — программисты и системные администраторы — очень серьезно относятся к экономии времени и энергии за счет минимизации движения пальцев.

«Emacs» — пример «безрежимного» или, если угодно, «однорежимного» редактора: пользователь *всегда* находится в режиме непосредственного редактирования текста в точке курсора, а команды издает, нажимая сложные сочетания клавиш и, при необходимости, вводя параметры команд в отдельном окне. Из-за стремления обеспечить прямую клавиатурную привязку как можно большему количеству команд и следующей из нее сложности используемых клавиатурных сочетаний был даже пущен слух о том, что «Emacs» расшифровывается как «Esc-Meta-Alt-Control-Shift» (хотя на деле, конечно, клавиатурные аккорды все же не так сложны, а «Emacs» — это просто «Editing MACroS», т.е. «макрокоманды редактирования»).

В действительности, различие это скорее идеологическое, чем прагматическое: в современных версиях «vi» в большинстве случаев также можно осуществить привязку часто употребляемых команд к клавиатурным комбинациям и выполнять их из режима редактирования, а в «Emacs» можно достаточно точно (если кому-то это потребуется) имитировать командный режим, характерный для многорежимных редакторов.

Дидактика редакторов

Реальное очень значимое отличие заключается в том, что по своей архитектуре «vi» — более или менее монолитная программа (с вытекающей отсюда компактностью), а «Emacs» — на самом деле, *расширяемая (программируемая)* коллекция макрокоманд редактирования, написанных на «Emasc Lisp» (диалекте известного языка функционального програм-

мирования). Лишь интерпретатор самого «Emacs Lisp» и небольшое количество часто выполняемых (и требовательных к ресурсам) команд встроены в саму программу и написаны на компилируемом C, большинство же команд написаны на «Lisp» и могут изменяться или дополняться пользователями (или профессиональными программистами по заказу пользователей).

За четверть века существования «Emacs», благодаря свободной модели лицензирования и открытой модели разработки, «оброс» невероятным количеством макрокоманд, «затачивающих» его под синтаксические особенности различных формальных языков (включая, но не ограничиваясь языками программирования и языками разметки), а также реализующих приложения, традиционно слабо ассоциируемые с «просто редакторами». Например, не выходя из «Emacs», можно работать с электронной почтой и службами новостей USENET (а также с гипертекстом со страничек WWW).

Или — что не менее интересно — не выходя из Emacs, можно прогнать текст программы через компилятор и подсветить синтаксические ошибки или предупреждения, воспользоваться символьным отладчиком или профилировщиком (реально, «Emacs» образует оболочку интегрированной среды разработки программ, и в этом качестве является вдохновителем и предшественником всех прочих интегрированных сред разработки (IDE)). И это лишь пара примеров.

Фактически, регулярно используемый «Emacs» позволяет реализовать (чисто в текстовом режиме, даже в системах, вообще не поддерживающих графику) метафору «рабочего стола», более известную по позднейшим графическим пакетам. Он реализует множественность окон (неперекрывающихся) на одном экране («фрейме»), а в графической среде способен работать со многими «фреймами» (окнами в терминах менеджера окон). Пакет «Emacspeak» добавляет к функциональности «Emacs» речевой вывод, предоставляя мощную поддержку для незрячих и слабовидящих пользователей[59].

Все это (доступность, расширяемость, интегрируемость) делает его серьезным претендентом на организацию «учебного» рабочего пространства программиста (и, на самом деле, есть университетские курсы, так и построенные). Можно ли это использовать в сегодняшней школе?

Однозначного ответа на этот вопрос у нас нет. Дело в том, что нам неизвестны такие (ориентированные на среду на основе «Emacs») курсы для школ вообще. А что касается России (и русскоязычного сообщества), то

[59] Функциональность «Emacspeak» пока доступна лишь англоязычной аудитории.

нам неизвестны примеры школьных курсов, вводящих на достаточно раннем этапе идеи функционального программирования. А без последнего — увы — расширяемость «Emacs» остается чисто теоретической.

Однако в качестве интегрированной среды именно для программирования (в том числе, на обычно изучаемых в школе директивных языках, например, Pascal) «Emacs» использовать, безусловно, можно. Следует только учесть, что пресловутая «кривая обучения» для него гораздо более вогнутая, чем для более простых (но и менее мощных) средств редактирования, обычно используемых в подобного рода средах. Грубо говоря, может потребоваться пара занятий до того, как учащийся будет чувствовать себя уверенно при наборе и редактировании программ, зато потом эти задачи будут решаться гораздо эффективнее. (Кривую обучения можно сгладить, создав дополнительный набор макрокоманд под конкретный курс и, наверное, это правильный способ, но он потребует от методиста незаурядного знания не только «Emacs», но и «Emacs Lisp».)

В обычных учебных курсах «vi» изучается раньше «Emacs». Такая структура заимствуется из традиционного курса подготовки администраторов и продвинутых пользователей открытых систем. Дело в том, что «vi», во-первых, стандартизован (и доступен во всех без исключения открытых системах), а во-вторых, компактен. Администратор системы может оказаться (например, при восстановлении после сбоя) в среде, где ему из экранных редакторов доступен только «vi». Поэтому для сисадминов базовые навыки работы с ним обязательны (вне зависимости от личных предпочтений).

В учебной обстановке, не ориентированной на профессиональную подготовку, такого императива, полагаем, нет, поэтому методисты и преподаватели вольны выбрать наиболее адекватный инструмент для демонстрации возможностей текстовых редакторов, если задача состоит только в знакомстве учащегося с таковыми. Выбор огромен, но остановится он, скорее всего, или на «vi», или на «Emacs».

Редактирование «без редакторов»

Далеко не всегда открывать файл и редактировать его вручную является оптимальным способом работы с содержащимся в нем текстом. Чем более формализован текст, и чем более типовым является редактирование, которое необходимо выполнить, тем больше шансов, что существует способ «малой кровью» оптимизировать этот процесс. Рассмотрим *очень формальную* задачу.

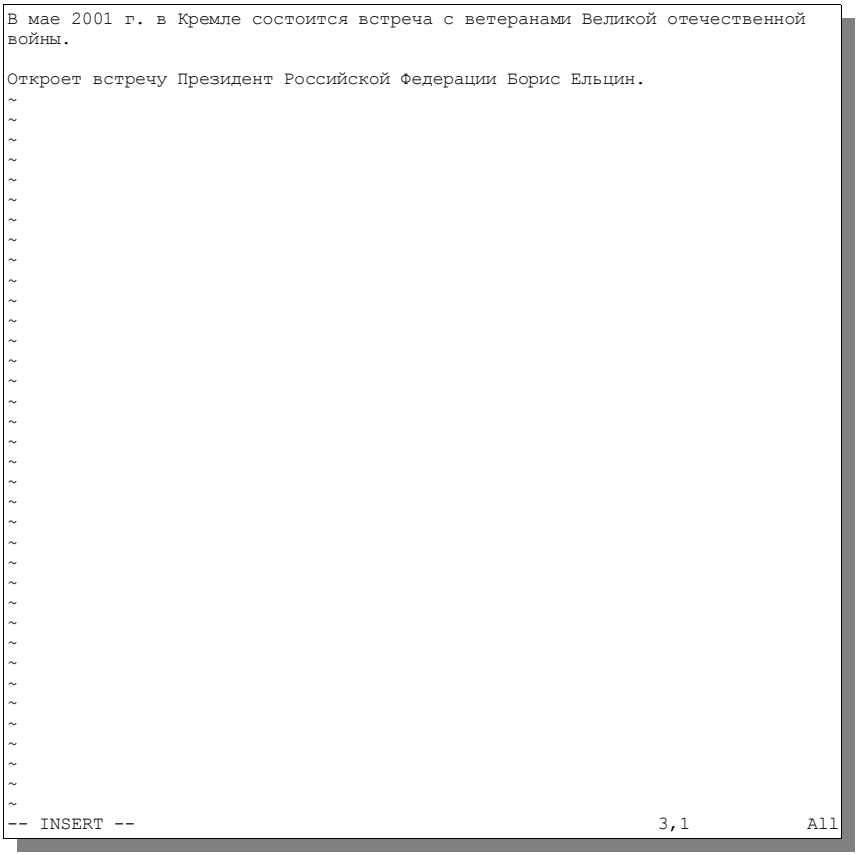

Рис. 1-71

Допустим, в файле `note` сохранен текст записки:

```
В мае 2001 г. в Кремле состоится встреча с ветеранами
Великой отечественной войны.

Откроет встречу Президент Российской Федерации Борис
Ельцин.
```

Если ситуация изменилась (и мы отдаем себе отчет, как именно), можно открыть файл с этим текстом в текстовом редакторе, например, vi (Рис. 1), и издать команду:

```
:s/Борис Ельцин/Владимир Путин/g
```

Текст (предсказуемо) приобретет вид:

```
В мае 2001 г. в Кремле состоится встреча с ветеранами
Великой отечественной войны.
```

```
Откроет встречу Президент Российской Федерации Влади-
мир Путин.
```

Однако, то же самое действие можно выполнить и «без редактора», а точнее, без интерактивного редактора, с помощью редактора *потокового*. Стандартный потоковый редактор называется sed, и синтакис его команд схож с синтаксисом командного режима стандартного интерактивного редактора vi, команда при этом издается непосредственно из командной строки:

```
$sed -n 's/Борис Ельцин/Владимир Путин/g' note
```

Если у нас подготовлен не один файл, а множество (например, `note.1, note.2, note.3`), и нужно внести в них единообразные замены (и ничего не пропустить, и нигде не ошибиться), мы обойдемся также всего одной командой.

```
$sed -n 's/Ельцин/Березовский/g' note.*
```

Если файлов будет тысяча, а требуемые изменения будут посложнее, нам, скорее всего, понадобится опять-таки всего одна команда (хотя, возможно, и потребуется серьезное изучение синтаксиса). Это называется потоковым редактированием, и оно интенсивно применяется, например, для наложения «заплаток» на исходные тексты программ (однако, как было продемонстрировано, с успехом может использоваться и для обработки текста на естественном языке).

Автоматизированная обработка текстов

ОС «Юникс» была во многом «рождена для обработки текстов» (прежде всего, это была система для программистов, а программы — это тексты). Набор служебных программ (утилит) современных стандартных ОС продолжает эту традицию, и в их составе можно найти десятки программ, ориентированных на работу с текстом. Многие из них (но не все) являются построчно-ориентированными, то есть текст понимается как последовательность строк.

Команда «grep» выводит строки, содержащие заданную подстроку, команда «sort» сортирует строки по алфавиту, «uniq» удаляет неуникальные (дублирующиеся) строки, «split» разделяет файлы, «cat» соединяет и т.п. Подробное описание команд потоковой обработки текста может занять отдельную толстую книгу.

В стандартной операционной среде отдельные утилиты могут «склеиваться» с помощью рассмотренных выше штатных средств оболочки операционной системы (перенаправление ввода-вывода, конвейер), что позволяет гибко решать самые сложные задачи обработки текстов, не прибе-

гая к программированию на специальных языках, компилированию и сборке программ.

Базовые регулярные выражения

Многими стандартными утилитами (такими как «sed», «grep», «vi») для поиска, замены, выбора текста, используются базовые регулярные выражения.

Регулярное выражение — это последовательность символов. При использовании (передаче в качестве аргумента программе или вводе в ходе сеанса редактирования) регулярное выражение (шаблон) обычно[60] окружается ограничителями — двумя одинаковыми символами, обозначающими его начало и конец, но не являющимися частью самого выражения. За исключением особых случаев в качестве ограничителей принято брать прямую косую черту (/, слэш), она окружает выражения и во всех нижеприведенных примерах.

Сами же символы могут (в зависимости от значения и, иногда, положения) иметь прямое (буквальное) значение или специальное. Символ-ограничитель не может употребляться внутри выражения в буквальном значении; также не рекомендуется использовать в этом качестве любой из перечисленных ниже специальных символов.

В буквальном значении символ автонимен, т.е. обозначает сам себя. /a/ обозначает букву «а», /слово/ означает слово «слово».

В синтаксисе базовых регулярных выражений определены следующие специальные символы.

- Любой одиночный символ обозначается точкой (.), а не вопросительным знаком, как при «глоббинге» имен файлов.
- Квадратные скобки ([и]) так же, как и при «глоббинге», используются для задания списков и диапазонов.
- Знак каретки (^) имеет специальное значение в первой позиции внутри квадратных скобок. В этом случае он означает отрицание: /[А-Яа-я]/ соответствует «любой букве русского алфавита», а /[^А-Яа-я]/ — «любому символу, кроме букв русского алфавита». Чтобы включить его в список, достаточно поместить его в любую другую позицию: /[~`^]/ — это «тильда, апостроф или знак каретки».
- Специальные значения, которые слишком сложны, чтобы их здесь рассматривать, в первой позиции внутри квадратных скобок имеют

[60] Исключением являются утилиты семейства «grep».

также: точка (.), знак равенства (=) и двоеточие (:).

- Каретка в начале выражения означает начало строки: ^T найдет заглавное «Т», начинающее строку. Подобно этому знак доллара ($) в конце выражения означает конец строки.
- Звездочка тоже используется в значении «нуля или более вхождений символа», но по-другому — для этого она должна следовать за таким символом. Шаблон /A*/ соответствует «A», «AA», «AAA» и т.д. Звездочка может следовать и за выражением, например, /[А-Яа-я]*/ означает «любую последовательность букв русского алфавита». Любая последовательность любых символов может быть обозначена /.*/.
- Обратная косая черта (\, «бэкслэш») «экранирует» следующий за нею символ, то есть отменяет его специальное значение. /\./ означает точку, /*/ — звездочку, а /\\/ — обратную косую черту. Обратная косая черта, за которой следует цифра, также имеет специальное значение, которое здесь не рассматривается.

Кроме того, регулярные выражения могут включать скобочные конструкции. В качестве скобок используются последовательности \(и \) (это совершенно нелогичное обратное (не отменяющее специальное значение следующего символа, а, наоборот, придающее ему специальное значение) значение бэкслэша обусловлено чисто историческими причинами: скобочные выражения вводились в синтаксис регулярных выражений, когда он уже устоялся). Например, шаблон /\(аб\)*/ соответствует строкам «аб», «абаб», «ababаб» и т.д. Скобочные конструкции могут быть вложенными.

1.10 Элементы программирования оболочки

В предыдущих главах мы рассматривали язык оболочки с точки зрения, в основном, *непосредственного исполнения* вводимых команд. Теперь взглянем на него под другим углом: как на универсальный *язык программирования*, а на оболочку — как на интерпретирующую реализацию этого языка.

От универсального языка программирования ожидаются: средства описания структур данных (переменные), средства вычисления выражений и присвоения их значений переменным, средства организации последовательного, условного и циклического исполнения, средства декомпозиции программы на подпрограммы.

Все эти средства присутствуют в стандартном языке оболочки[61]. Его осо-

[61] Поскольку открытые системы являются многозадачными, декомпозируя задачу,

бенностью является возможность использовать команды (стандартные и нестандартные) ОС в качестве своего рода «вызовов функций» (хотя и определение, и вызов функции также присутствуют в языке как отдельный механизм).

При попытках реализовать «простые программы из учебников» на языке оболочки результат часто оказывается не самым изящным. Однако этот язык очень хорошо приспособлен для решения административных и системных задач. В большинстве открытых ОС значительная часть самой системы написана на этом языке.

Пределы главы позволяют лишь бегло представить механизмы языка оболочки, проиллюстрировав их несколькими примерами.

Комментарии и указание оболочки

Часть любой строки, начинающаяся со знака «#» вплоть до символа новой строки является комментарием и не исполняется оболочкой. Как и в других языках программирования, комментарии предназначены для передачи какой-либо неочевидной из текста самой программы информации ее читателю.

Во многих системах (включая «ГНУ/Линукс») специальная нестандартная форма комментария может использоваться также для передачи ядру системы информации о том, какую именно оболочку использовать для интерпретации сценария. Такой комментарий имеет вид символов «#!», за которыми слитно следует имя исполняемого файла (обычно «/bin/sh», «/bin/bash» или «/usr/bin/bash»), и должен начинать файл сценария, то есть находиться в первой строке.

Информация из специального комментария востребуется только если файлу сценария придан атрибут исполняемого, а его выполнение инициировано указанием имени файла в качестве команды ОС. Если сценарий запускается на выполнение явным вызовом дополнительного экемпляра оболочки (например, «/usr/bin/bash <сценарий»), специальный комментарий игнорируется. Эта строка почти всегда присутствует при публикации сценариев, чтобы было понятно, используется ли язык стандартной оболочки («sh»), ее расширения («bash», «zsh», «ksh») или не вполне совместимые со стандартом диалекты (такие, как «tcsh»)[62].

решаемую сценарием, на несколько сценариев и используя системную многозадачность, можно реализовать определенный параллелизм в исполнении сценариев. Описание соответствующих приемов и команды «wait», позволяющей организовать синхронизацию («рандеву») потоков исполнения, выходит за рамки настоящего курса.

[62] На самом деле, в специальном комментарии может быть указан любой исполняе-

Переменные и присваивание значений

Конструкция, состоящая из *имени переменной* и ее *значения*, разделенных знаком равенства («=») без промежутков, за которой не следует никакой команды, является определением переменной оболочки. Переменная, определенная таким способом, не оказывает влияния на поведение последующих команд.

Чтобы значение переменной передавалась всем вызываемым командам, ее следует сделать передаваемой (*экспортировать* ее) командой «export» с именем переменной в качестве аргумента.

Для того, чтобы *присвоить* переменной новое значение, ее просто переопределяют; определение, таким образом, выступает и в качестве *оператора присваивания*.

Все переменные стандартной оболочки имеют строковый тип, то есть могут принимать значения, равные строкам (или цепочкам) символов переменной длины (включая пустую цепочку с нулевой длиной).

Тем не менее, в языке присутствуют арифметические и логические операции. Арифметические операции определены на множестве строк, представляющих собой запись чисел.

«Арифметические» и «логические» выражения

Выражения обычно вводятся в программу с использованием конструкции *арифметического раскрытия выражений* «$((выражение))». Заключенная в двойные круглые цепочка символов интерпретируется как арифметическое или логическое выражение, результат *вычисления* которого оболочкой подставляется на место вхождения этой конструкции в командной строке (Рис. 1-72).

```
[alice@wonderland alice]$ echo $((2+2))
4
[alice@wonderland alice]$ echo $(((2+2)-(2*2)))
0
```

Рис. 1-72

Выражение интерпретируется как если бы оно было заключено в двойные кавычки «"» и «"», т.е. раскрываются имена переменных, предваренные знаком доллара «$», но специальное значение прочих символов (например, звездочки) отменяется.

мый файл, способный интерпретировать текстовые файлы. Возможно, читатель сталкивался с ним в начале программ, например, на языке Перл.

Выражение состоит из переменных, констант и знаков операций. Стандартом определены операции, перечисленные в таблице на Рис. 1-73).

```
()              группировка выражений
unary +         одноместный (унарный) плюс
unary -         одноместный минус
~               одноместное побитное отрицание
!               логическое отрицание
*               умножение
/               целочисленное деление
%               остаток от целочисленного деления
+               сложение
-               вычитание
<<              побитный сдвиг влево
>>              побитный сдвиг вправо
<, <=           меньше, меньше или равно
>, >=           больше, больше или равно
==              равно
!=              не равно
^               побитное исключающее «ИЛИ»
|               побитное включающее «ИЛИ»
&               побитное «И»
||              логическое «ИЛИ»
expr?expr:expr условный оператор
=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=    операторы присвоения
```

Рис. 1-73

Знакомые с языком «Си» легко узнают в этом списке список стандартных операций этого языка за исключением унарных инкрементов и декрементов (префиксных и постфиксных), функции «sizeof()». В отличие от стандарта «Си», стандарт на язык оболочки требует определения этих операций лишь на длинных беззнаковых целых.

Поскольку руководства и справочники по «Си» общедоступны, разбирать операции подробно мы не будем. Они, в основном, соответствуют общепринятой математической и программистской нотации для выражений, за исключением представления символа равенства сочетанием «==», а не символом «=».

Чаще всего арифметическое раскрытие применяется в команде присваивания, но его можно использовать в любом месте (например, для задания числового операнда команды или числового параметра ключа) (Рис. 1-74).

```
[alice@wonderland alice]$ n=1
[alice@wonderland alice]$ echo $n
1
[alice@wonderland alice]$ n=$((n+1))
[alice@wonderland alice]$ echo $n
2
```

Рис. 1-74

Помимо конструкции арифметического раскрытия, существует стандартная команда «expr», также вычисляющая значение выражения (с

несколько иным синтаксисом, в частности, использующим для проверки на равенство знака «=»), переданного ей в качестве аргумента, и *выводящая* его результат. Выражение при этом следует экранировать двойными кавычками «"» и «"».

Команду «expr» лучше не применять, если ее можно заменить командой «echo $((выражение))» (с учетом отличий в синтаксисе), но в чужих сценариях она может встретиться. Кроме того, она, в отличие от арифметического раскрытия, позволяет выполнять сравнение строк на равенство. Выполнить подстановку выводимого командой «expr» результата в командную строку можно посредством механизма обратных апострофов, обсуждающегося ниже.

При настоятельной необходимости применить в сценарии численные методы, включающие работу с вещественными числами в представлении с плавающим десятичным знаком, можно воспользоваться стандартной командой вычисления выражения с произвольной точностью «bc», которая обладает также внутренними возможностями сценирования. Ее описание выходит за рамки этого курса.

Генерация кодов возврата

Обычно директивные языки для определения *условий* в операторах условного и циклического исполнения применяют механизм *выражений*. Язык оболочки в этом плане достаточно эксцентричен, и использует с этой целью механизм *кодов возврата* (переменной «$?») команды ОС. Дополнительную путаницу вводит то, что в открытых системах успешный код возврата — ноль, который, таким образом, соответствует логическому значению «истинно», в то время, как в «логических» выражениях, обсуждавшихся выше, используется соглашение «Си» (ноль, наоборот, соответствует значению «ложно», а «истинно» — любому ненулевому значению).

(Отсутствие простого механизма согласования между раскрытием арифметических выражений и условиями условного и циклического исполнения, различие в синтаксисе а) раскрываемых выражений, б) команды «expr» и в) обсуждаемой ниже команды «test» являются серьезными недостатками языка оболочки, заметно усложняющими его освоение даже опытными программистами.)

Хотя условие условного или циклического исполнения может задаваться самыми разными командами (поскольку любая команда завершается с *каким-либо* кодом возврата), чаще всего в соответствующих операторах используется команда «test». Эта команда вычисляет переданное ей в виде набора аргументов выражение и завершается с кодом возврата «0»

(ноль), если оно истинно, «1», если оно ложно и «2», если выражение содержит синтаксическую ошибку. Она настолько важна для программирования оболочки, что для нее введено особое сокращение: вместо подачи команды «test» с аргументами можно просто заключить аргументы в квадратные скобки «[» и «]», отделив их от первого и последнего аргумента промежутками (Рис. 1-75).

```
[alice@wonderland alice]$ test 1 -eq 2
[alice@wonderland alice]$ echo $?
1
[alice@wonderland alice]$ test 1 -eq 1
[alice@wonderland alice]$ echo $?
0
[alice@wonderland alice]$ [ 1 -eq 1 ]
[alice@wonderland alice]$ echo $?
0
```

Рис. 1-75

В качестве аргументов команды «test» могут выступать константы, переменные и символические обозначения операций, а также круглые скобки, позволяющие менять приоритет исполнения операций. Обратите внимание, что выражение передается команде в виде совокупности аргументов, а не в виде одного аргумента, поэтому аргументы *должны* разделяться промежутками, заключать выражение целиком в кавычки *нельзя*, а любые специальные символы *должны* экранироваться.

Командой «test» поддерживаются операции, перечисленные в таблице на Рис. 1-76. Они различаются по типу (точнее, по интерпретации) операндов, но все возвращают «логические» (в указанном выше смысле) значения.

Операции над строками	
-n строка	Длина строки ненулевая
-z строка	Длина строки нулевая
строка1 = строка2	Строки идентичны
строка1 != строка2	Строки различны
Операции над числами	
число1 -eq число2	Целые числа равны
число1 -ne число2	Целые числа не равны
число1 -gt число2	Первое целое больше второго
число1 -ge число2	Первое целое больше или равно второму

Операции над строками	
число1 -lt число2	Первое целое меньше второго
число1 -le число2	Первое целое меньше или равно второму
Операции над выражениями	
выражение1 -a выражение2	Логическое «И»
выражение1 -o выражение2	Логическое «ИЛИ»
! выражение	Логическое «НЕ»
Операции над именами и дескрипторами файлов	
-b файл	Файл существует и является блочным устройством
-c файл	Файл существует и является символьным устройством
-d файл	Файл существует и является каталогом
-e файл	Файл существует
-f файл	Файл существует и является обычным файлом
-g файл	Файл существует и обладает установленным битом SGID
-h файл	Файл существует и является символической ссылкой
-L файл	- “ -
-p файл	Файл существует и является файлом-очередью
-r файл	Файл существует и может быть прочитан текущим процессом
-S файл	Файл существует и является гнездом (сокетом)

Операции над строками	
-s файл	Файл существует и обладает ненулевой длиной
-u файл	Файл существует и обладает установленным битом SUID
-w файл	Файл существует и может быть записан текущим процессом
-x файл	Файл существует и может исполняться текущим процессом
-t дескриптор	Файл, связанный с дескриптором, открыт и ассоциирован с терминалом

Рис. 1-74

Операции над числами допускают в качестве операндов только константы и переменные, однако можно использовать и раскрываемые арифметические выражения. И символическое обозначение операций, и операнды выражений, передаваемых команде «test», в терминах командной строки являются операндами команды (хотя форма операций и похожа на ключи).

Условное исполнение

Оболочка реализует команду *условного исполнения*, доступную в трех модификациях: «if – then – fi», «if – then – else – fi» и «if – then – elif ... – fi». Простейшей является форма «if – then – fi» (Рис. 1-77).

```
if список_команд1
               then список_команд2
fi
```

Рис. 1-77

```
[alice@wonderland alice]$ if test -e контакт
> then echo "Есть контакт"
> fi
[alice@wonderland alice]$ touch контакт
[alice@wonderland alice]$ if test -e контакт; then echo "Есть контакт"; fi
Есть контакт
```

Рис. 1-78

Выполняется оператор условного исполнения так: выполняется список_команд1, затем, если код завершения истинен (равен нулю), выполняется список_команд2. Команды в каждом списке могут соединяться переводом строки или точкой с запятой «;» (Рис. 1-78).

Обратите внимание, что если оператор условного исполнения (или любая другая сложная конструкция) вводится в интерактивном режиме, и строка завершилась раньше, чем оператор, оболочка выведет *строку приглашения продолжения* (значение переменной $PS2; по умолчанию «>»), будет ожидать продолжения ввода и повторять это, пока оператор не будет завершен (в данном случае — сложным символом «fi»).

Если в первом списке более одной команды, кодом завершения списка будет код завершения последней в списке команды. Однако существуют два других символа завершения команды: «&&» и «||». Кодом завершения списка, соединенного «&&», является результат выполнения операции «И» над значениями истинности кодов завершения входящих в список команд, а кодом завершения «||»-списка — результат выполнения операции «ИЛИ».

Использование в первом списке команды «if» более одной команды является экзотическим приемом, которого по возможности следует избегать.

```
if список_команд1
                then список_команд2
                else список_команд3
fi
```

Рис. 1-79

```
[alice@wonderland alice]$ if test $((2*2)) -eq 4
> then echo "Дважды два четыре"
> else echo "Неладно что-то в Датском королевстве"
> fi
Дважды два четыре
```

Рис. 1-80

Вторая форма (Рис. 1-79) более сложна: выполняется список_команд1, затем, если код завершения истинен, выполняется список_команд2, а если ложен — список_команд3 (Рис. 1-80).

```
if список_команд1
                then список_команд2
                elif список_команд3
                                then список_команд4
                ...
                [else список_команд5]
fi
```

Рис. 1-81

И наконец, третья форма (Рис. 1-81) позволяет задавать множественные условия: если код завершения первого списка истинен, выполняется второй список команд, иначе выполняется третий список команд и, если его код завершения истинен, выполняется четвертый список команд. Конструкция «elif – then» может быть множественной, но в любом слу-

чае выполнен будет лишь один «then»-список.

В «if – then - elif ... fi» форме оператора «if» также может присутствовать конструкция «else»; следующий за ней список команд будет выполнен, если коды завершения «if»-списка и всех «elif»-списков оказались ложными.

Циклическое исполнение с предусловием

Оболочка поддерживает два оператора *цикла с предусловием*: «while – do – od» (Рис. 1-82) и «until – do – od» (Рис. 1-83).

```
while список_команд1 do
              список_команд_2 done
```

Рис. 1-82

```
until список_команд1 do
              список_команд_2 done
```

Рис. 1-83

Выполнение любого из них заключается в том, что выполняется список_команд1 и, в зависимости от кода завершения либо выполняется список_команд2 и выполнение цикла повторяется, либо выполнение цикла завершается. «While»-цикл выполняется, пока код завершения первого списка истинен, а «until»-цикл — пока он ложен.

Обратите внимание, что «until»-цикл, в отличие от использования этого ключевого слова в большинстве языков программирования, является также циклом с *пред*условием, а не с *пост*условием. Если же действительно необходимо организовать циклическое исполнение с постусловием, реальное тело цикла можно включить в список_команд_1, завершив его командой проверки условия, а номинальное тело (заключенное между ключевыми словами «do» и «done») сделать пустым.

Циклическое исполнение со списком значений

Оболочка предоставляет также возможность организации циклического исполнения с переменной, пробегающей список значений (Рис. 1-84).

```
for переменная [ in [значение ... ]]
    do
    список_команд done
```

Рис. 1-84

Список_команд будет исполнен по одному разу для каждого значения в списке значений (Рис. 1-85).

```
[alice@wonderland alice]$ for фрукт in яблоко ананас морковка
> do
> echo $фрукт
> done
яблоко
ананас
морковка
```

Рис. 1-85

Если ключевое слово «in» и список значений не указаны, переменная будет пробегать список значений специальных переменных $1 - $9, соответствующих аргументам командной строки (см. ниже) (в порядке их следования), из которой запущен сценарий (программа).

Многовариантное условное исполнение

Для многовариантного условного исполнения в зависимости от значения строковой переменной оболочка поддерживает оператор «case» с на редкость причудливым синтаксисом (Рис. 1-86).

```
case строка in
                [()шаблон) список_команд
                ;;
                [[()шаблон[ | шаблон] ... )  список_команд
                ;;] ...
                [[()шаблон[ | шаблон] ... )  список_команд]
esac
```

Рис. 1-86

Указанная строка (обычно — результат раскрытия значения переменной) поочередно сравнивается с шаблонами и при первом совпадении выполняется соответствующий список команд (до конструкции «;;»), после чего выполнение оператора «case» завершается.

Строки, указанные в качестве шаблонов, подвергаются обычному раскрытию, за исключением того, что специальные символы «*», «*», «[...] » не приводят к поиску файлов, а используются (по тем же правилам, что и при раскрытии шаблонов имен файлов) как метасимволы при сравнении.

Чаще всего многовариантное условное исполнение применяется при разборе списка параметров, с которыми сценарий был вызван для исполнения. Примеры настолько громоздки, что мы их опускаем.

Ввод-вывод

Вывод значений переменных и выражений сценарием как правило осуществляется известной нам командой «echo» или командой «printf» (предназначенной для форматированного вывода), которую мы в этом курсе не рассматриваем.

Ввести данные (т.е. присвоить значения переменным) можно с помощью команды «read». Эта команда читает из стандартного ввода строку, разделяет ее на отдельные аргументы (пробелами, табуляцией, знаком переноса или символами, содержащимися в переменной окружения $IF), и присваивает их перечисленным в команде переменным. Если аргументов оказывается больше, чем переменных, остаток строки присваивается последней переменной (Рис. 1-87).

```
[alice@wonderland alice]$ read a b c
A B C D
[alice@wonderland alice]$ echo $a
A
[alice@wonderland alice]$ echo $b
B
[alice@wonderland alice]$ echo $c
C D
```

Рис. 1-87

Однако во многих случаях вывод (и ввод) осуществляется не только командой «echo», но и прочими командами, применяемыми в сценарии. Следует понимать, что каждая команда, ввод-вывод которой не переназначен явно и не включен в конвейер, *наследует*, в числе прочих атрибутов, дескрипторы стандартных ввода-вывода от подающего ее процесса. Таким образом, переназначив, например, стандартный вывод сценария в файл, мы тем самым переназначаем стандартный вывод каждой вызываемой программы, если только ее вывод не переназначается отдельно или не передается по конвейеру.

Передача аргументов сценарию

Как и любая команда, сценарий, написанный на языке оболочки, может вызываться с позиционными параметрами (ключами и операндами).

Сценарию эти параметры доступны посредством ряда специальных переменных, перечисленных в таблице на Рис. 1-88.

$#	количество переданных параметров
$0	имя сценария
$1	первый позиционный параметр
...	
$9	девятый позиционный параметр
$*	все аргументы одной строкой

Рис. 1-88

Стандарт не предоставляет каких-либо средств поддержки грамматиче-

ского разбора командной строки; обычно для этого используется комбинация операторов «for» и операторов условного исполнения.

Функции и вызов других сценариев

В сценарии на языке оболочки можно определить подпрограммы-функции. В функции определенными являются переменные, определенные сценарием (или его окружением) на момент вызова функции. Однако сама функция должна быть объявлена до ее вызова. Кроме того, функции при вызове можно передавать параметры (как любой команде), которые она может адресовать способом, указанным выше (Рис. 1-88).

Определение функции имеет вид, указанный на Рис. 1-89.

```
имя()
{
список команд
}
имя() {
список команд
}
```

Рис. 1-89

Вызывается функция простым указанием ее имени в качестве команды (см. Рис. 1-90).

```
[alice@wonderland alice]$ функция1()
> {
> echo "Я первая функция"
> }
[alice@wonderland alice]$ функция2() { echo "Я вторая функция"; }
[alice@wonderland alice]$ функция1
Я первая функция
[alice@wonderland alice]$ функция2
Я вторая функция
[alice@wonderland alice]$ функция3
-bash: функция3: command not found
```

Рис. 1-90

Раскрытие команды

Крайне занимательной является возможность *раскрыть* целую команду, то есть включить в командную строку ее стандартный вывод. Для этого команда заключается в обратные апострофы «`» и «`», которые не следует путать ни с одиночными прямыми апострофами, ни с кавычками.

```
[alice@wonderland alice]$ echo "Временные файлы:" `ls *~ *.tmp` ". Больше
нет."
Временные файлы: vars.tmp цикл~ . Больше нет.
```

Рис. 1-91

В примере на Рис. 1-91 команда «echo» подается с тремя операндами,

первый и последний из которых являются текстовыми константами, а второй — результатом раскрытия команды «ls *~ *.tmp», то есть списком файлов, соответствующих первому и второму шаблонам, указанным в качестве операндов команды «ls».

Включенный документ

В некотором смысле обратной по отношению к раскрытию команды с помощью обратных апострофов является конструкция *включенного документа* (или, конструкция «документ здесь»).

На Рис. представлена конструкция вида «<<цепочка», внешне похожая на перенаправление ввода команды из файла. Однако цепочка представляет собой не имя файла, а ограничивающую цепочку символов, а сам ввод осуществляется непосредственно из файла сценария (или, в данном случае, с терминала) до тех пор, пока очередная строка не совпадет с ограничивающей цепочкой (Рис. 1-92).

```
[alice@wonderland alice]$ sort <<+++
> Манька
> Кошка
> Хавкет
> +++
Кошка
Манька
Хавкет
```

Рис. 1-92

Встроенный документ особенно удобен для сценариев, представляющих сильно параметризованные алгоритмы, указание данных для которых в отдельных файлах привело бы к большому количеству последних.

1.11 Справочник по наиболее употребительным стандартным командам ОС

exit - завершить исполнение оболочки

Синтаксис: exit [код_зав]

Семантика: exit завершает исполнение оболочки с кодом возврата (0-255), указанным в «код_зав». Перехват «exit» командой «trap» выполняется до завершения оболочки, если только «exit» выполняется не из этой команды «trap». Если n не указана, код возврата равен коду возврата последней выполненной команды. При выполнении «exit» из «trap» последней командой считается команда, выполненная непосредственно до выполнения «trap».

cal - вывести календарь

Синтаксис: cal [[месяц] год]

Семантика: cal выводит на стандартный вывод календарь, используя юлианское летоисчисление для дат с 1 января 1 г. по 2 сентября 1752 г. и григорианское - с 14 сентября 1752 г. по 31 декабря 9999 г.

Операнды: месяц — отображаемый месяц указывается десятичным числом с 1 (январь) по 12 (декабрь). По умолчанию – текущий месяц. *Год* — отображаемый год указывается десятичным числом с 1 до 9999. По умолчанию – текущий год.

date – вывести или установить дату и время

Синтаксис: date **[**-u**] [**+*формат***] ;** date **[**-u**]** *ммддччмм***[[***сс***]***гг***]**

Семантика: date выводит дату и время. Поданная с операндом в формате даты, date пытается установить время и дату. Операнд, начинающийся с «+» устанавливает формат вывода данных.

Ключ: -u — выполнить команду, как если бы переменная TZ (часовой пояс) была установлена в «UTC0» или "GMT0" (время по Гринвичу).

Операнды: +формат — если указан формат, каждый спецификатор формата заменяется при выводе на соответствующее значение. Вывод всегда завершается символом новой строки. Поддерживаются спецификаторы «%a» (краткое название дня недели), «%A» (название дня недели), «%b», «%h» (краткое название месяца), «%B» (название месяца), «%c» (дата и время), «%C» (век в двузначном представлении), «%d» (день месяца в двузначном представлении), «%D» (дата в формате мм/дд/гг), «%e» (день месяца в одно- или двузначном представлении), «%H» (час дня в 24-часовом двузначном представлении), «%I» (час дня в 12-часовом двузначном представлении), «%j» (номер дня в году), «%m» (номер месяца в двузначном представлении), «M» (минуты в двузначном представлении), «%n» (конец строки), «%p» (знак «утра» или «пополудни»), «%r» (час дня в двенадцатичасовом представлении со знаком «утра» или «пополудни»), «%S» (секунды в двузначном представлении), «%t» (табуляция), «%T» (время в формате ЧЧ:ММ:СС в 24-часовом представлении), «%u» (порядковый номер дня недели («Пн.»=1), «%U» (порядковый номер недели в году при неделе, начинающейся в воскресенье), «%V» (номер недели в году при неделе, начинающейся в понедельник), «%w» (номер дня недели, начинающейся с воскресенья), «%W» (номер недели, начинающейся в понедельник), «%x» (полное представление

даты), «%X» (полное представление времени), «%y» (год в двузначном представлении), «%Y» (год в одно- или двузначном представлении), «%Z» (часовой пояс), «%%» (процент).

Переменные: TZ — часовой пояс, в котором выводится или устанавливается дата.

Вывод: если формат не указан, дата выводится в формате «+%a %b %e %H:%M:%S %Z %Y».

man - вывести системную документацию

Синтаксис: man **[**-k**]** *имя*...

Семантика: man выводит информацию о каждом из перечисленных операндов.

Ключ: -k — интерпретировать операнды как ключевые слова для поиска в базе кратких описаний и вывести строки, в которых содержатся эти слова.

Операнды: имя — ключевое слово или имя команды.

Переменные: PAGER — используемый фильтр постраничного вывода.

echo — вывести аргументы

Синтаксис: echo [*строка*...]

Семантика: echo выводит свои аргументы после раскрытия специальных символов в стандартный вывод, завершая вывод символом новой строки.

Операнды: строка — строка, подлежащая выводу. В строке после раскрытия спецсимволов оболочки раскрываются следующие символы: \a — звуковой сигнал, \b — пробел, \c — подавить вывод символа новой строки, \f — перевод страницы, \n — символ конца строки, \r — символ возврата каретки, \t — табуляция, \v — вертикальная табуляция, \\ — обратная косая черта, \0*код* — символ с восьмеричным кодом «код».

Стандартный вывод: между аргументами выводятся пробелы.

touch - изменить временные атрибуты доступа и модификации файлов

Синтаксис: touch **[**-acm**][** -r *справ_файл*| -t *time***]** *файл*...

Семантика: touch изменяет атрибуты времени последней модификации

или времени последнего доступа файлов, или (по умолчанию) оба. Значение атрибута указывается аргументом ключа «-t» или заимствуется у файла, указанного в качестве аргумента ключа «-r». Если оно не указано, используется текущее время.

Если файлы не существуют, они создаются.

Ключи: -a — изменить время доступа, -c — не создавать несуществующих файлов, -m — изменить время модификации, -r справ_файл — заимствовать атрибут у файла «справ_файл»; -t время — использовать вместо текущего указанное время в формате «[[ВВ]ГГ]ММДДччмм[.сс]», где ММ — номер месяца, ДД — день месяца, чч — час дня, мм — минуты, ВВ — первые две цифры года, ГГ — последние две цифры года, сс — секунды.

Операнды: файл — имя файла.

ls — вывести содержимое каталога

Синтаксис: ls **[**-CFRacdilqrtu1**][**-H | -L **][**-fgmnopsx**][*файл*...]**

Семантика: Для каждого операнда, именующего файл типа иного, нежели каталог или ссылка на каталог, ls выводит имя и требуемую ключами информацию. Для каждого операнда, именующего каталог или ссылку на каталог, ls выводит имена и требуемую ключами информацию о каждом файле, содержащемся в этом каталоге. Если операнды не указаны, ls выводит информацию о файлах в текущем каталоге. Для ссылок на каталоги выводится информация о каталоге, если даны ключи «-d», «-F» или «-l» и не даны ключи «-H» или «-L», и информация о файлах в каталоге, если не даны ключи «-d», «-F» или «-l» или даны ключи «-H» или «-L».

Важнейшие ключи: **-R** — выводить рекурсивно информацию о подкаталогах; **-a** — включить информацию о скрытых файлах (файлах с именами, начинающимися на точку), **-l** («эль») — выводить информацию в «длинном» формате; **-p** — выводить после имен каталогов «/»; **-t** -- отсортировать в порядке времени изменения.

Операнды: файл — имя файла.

Переменные: COLUMNS — количество столбцов на терминале; *TZ* — часовой пояс.

Вывод: по умолчанию выводится по одной записи в строке. -l — выводятся тип и права файла, количество ссылок, имя владельца, имя группы, длина файла, дата и время, имя файла.

rm – удалить записи о файлах

Синтаксис: rm **[**-fiRr**]** *файл...*

Семантика: rm удаляет запись в каталоге для каждого операнда за исключением файлов «.» или «..» в любом каталоге и за исключением (если не даны ключи «-r», «-R») каталогов.

Ключи: **-f** — не запрашивать подтверждения; -i — запрашивать подтверждение; -r, **-R** — рекурсивно удалять содержимое указанных каталогов.

Операнды: файл — имя файла.

Вывод ошибок: стандартный вывод ошибок используется для вывода запросов на подтверждение удаления файлов («-i»).

mkdir – создать каталог

Синтаксис: mkdir [-p][-m права] каталог...

Семантика: mkdir создает перечисленные каталоги.

Операнды: каталог — создаваемый каталог.

rmdir — удалить каталоги

Синтаксис: rmdir **[**-p**]** *каталог...*

Семантика: rmdir удаляет записи, соответствующие указанным пустым каталогам.

Операнды: каталог — удаляемый каталог.

cp – копировать файлы

Синтаксис: cp [-fip] исх_файл цел_файл ; cp [-fip] исх_файл... каталог ; cp -R [-H | -L | -P][-fip] исх_файл... каталог ; cp -r [-H | -L | -P][-fip] исх_файл... каталог

Семантика: первая синтаксическая форма характеризуется двумя файлами, ни один из которых не должен быть существующим каталогом. cp копирует исх_файл в цел_файл. Если исх_файл — символическая ссылка, копируется целевой файл этой ссылки.

Вторая синтаксическая форма характеризуется двумя или более операндами, отсутствием ключей «-R» или «-r» и неприменимостью первой формы. Исходные файлы не должны быть каталогами, а каталог должен быть существующим каталогом. cp копирует исходные файлы в указанный каталог под именами, совпадающими с краткими именами исход-

ных файлов.

Третья и четвертая форма характеризуется двумя или более операндами и ключами «-r» или «-R». cp копирует все указанные файлы, а также рекурсивно каталоги с их содержимым в каталог.

Важнейшие ключи: **-i** — запрашивать подтверждение перед копированием в существующие файлы; **-p** — сохранять по возможности времена изменения и доступа к файлу, владельца и группу, права доступа; **-R, -r** — рекурсивно копировать содержимое каталогов.

Операнды: исх_файл — исходный файл; цел_файл — целевой файл; каталог — целевой каталог.

Стандартный вывод ошибок: стандартный вывод ошибок используется для вывода запросов на подтверждение перезаписи существующих файлов («-i»).

mv – переместить файлы

Синтаксис: mv [-fi] исх_файл цел_файл ; mv [-fi] исх_файл... каталог

Семантика: в первой синтаксической форме, характеризующейся тем, что последний операнд не является ни каталогом, ни символической ссылкой на каталог, mv перемещает исх_файл в цел_файл.

Во второй синтаксической форме cp копирует исходные файлы в указанный каталог под именами, совпадающими с краткими именами исходных файлов.

Ключи: **-f** — не запрашивать подтверждения перезаписи существующих файлов; **-i** — запрашивать подтверждение перезаписи существующих файлов.

Операнды: исх_файл — исходный файл; цел_файл — целевой файл; каталог — целевой каталог.

Стандартный вывод ошибок: стандартный вывод ошибок используется для вывода запросов на подтверждение перезаписи существующих файлов («-i»).

echo — вывести аргументы

Синтаксис: echo [*строка*...]

Семантика: echo выводит свои аргументы после раскрытия специальных символов в стандартный вывод, завершая вывод символом новой строки.

Операнды: строка — строка, подлежащая выводу. В строке после раскрытия спецсимволов оболочки раскрываются следующие символы: \a — звуковой сигнал, \b — пробел, \c — подавить вывод символа новой строки, \f — перевод страницы, \n — символ конца строки, \r — символ возврата каретки, \t — табуляция, \v — вертикальная табуляция, \\ — обратная косая черта, \0*код* — символ с восьмеричным кодом «код».

Стандартный вывод: между аргументами выводятся пробелы.

cat – вывести содержимое файлов

Синтаксис: cat [-u][файл...]

Семантика: cat последовательно выводит содержимое файлов.

Ключ: -u — читать и выводить файлы побайтно (по умолчанию — построчно).

Операнды: файл — выводимый файл. Если файл не указан, читается стандартный ввод. Если в списке файлов присутствует имя «-», вместо этого файла читается стандартный ввод.

Реализация: в большинстве систем ключ «-u» не реализован.

chmod — изменить права на файл

Синтаксис: chmod [-R] *режим файл ...*

Семантика: chmod изменяет биты режима доступа к каждому указанному файлу в соответствии с указанным режимом. Изменить режим доступа к файлу может только процесс с действующим идентификатором пользователя, совпадающим с владельцем файла, или привилегированный процесс.

Ключ: -R — рекурсивно изменять режим доступа к файлам, расположенным в указанных каталогах.

Операнды: режим — устанавливаемый режим доступа (в rwx- или числовой нотации); *файл* — имя файла.

umask — вывести или установить маску прав доступа

Синтаксис: umask [-S][*маска*]

Семантика: umask устанавливает маску прав вновь создаваемых в окружении текущей оболочки файлов в соответствии с указанным значением. Если операнд *маска* не указан, umask выводит текущую маску.

Ключ: *-S* — вывести маску в rwx-нотации.

Операнд: маска — маска прав в rwx- или числовой нотации.

ps — вывести состояние процессов

Синтаксис: ps [-aA][-defl][-G список][-o формат]...[-p список][-t список] [-U список][-g список][-n список][-u список]

Семантика: ps выводит информацию о процессах в рамках собственных привилегий. По умолчанию выводится информация о процессах с теми же действующим UID и управляющим терминалом, что и у подающего команду пользователя.

Основные ключи: *-a* — вывести информацию о процессах, ассоциированных с терминалами; *-A* — вывести информацию о всех процессах; *-f* — вывести «полный» список; *-l* — вывести «длинный» список; *-o формат* — вывести список в указанном формате; -p *список* — вывести информацию о процессах с перечисленными в списке PID; *-u список* — вывести информацию о процессах с перечисленными идентификаторами или именами пользователей.

bg — перевести задание на задний план

Синтаксис: bg [идентификатор ...]

Семантика: bg возобновляет выполнение приостановленных процессов на заднем плане.

Операнд: идентификатор — PID ведущего процесса задания или номер задания, предваренный знаком «%».

fg — перевести задание на передний план

Синтаксис: fg [идентификатор ...]

Семантика: fg возобновляет выполнение приостановленных процессов на переднем плане или переводит процессы заднего плана на передний.

Операнд: идентификатор — PID ведущего процесса задания или номер задания, предваренный знаком «%».

jobs — вывести состояние заданий в текущем сеансе

Синтаксис: jobs [-l| -p][идентификатор...]

Семантика: jobs выводит состояние заданий в окружении текущей оболочки.

Ключи: -l («эль») — вывести полную информацию (номер задания, текущее задание, идентификатор группы, состояние и команду) для каждого задания; *-p* — вывести только идентификатор ведущего процесса каждого задания.

*Операнд: идентификатор — PID ведущего процесса задания или номер задания, предваренный знаком «%».*Если операнд отсутствует, выводится информация о всех заданиях.

kill — прекратить исполнение процесса или передать ему сигнал

Синтаксис: kill -s сигнал идентификатор ... ; kill -l [статус_завершения] ; kill [-сигнал] идентификатор ... ; kill [-номер_сигнала] идентификатор ...

Семантика: kill посылает указанный сигнал указанным процессам.

Ключи: *-l* («эль») — вывести список поддерживаемых сигналов; *-s сигнал* — послать сигнал с указанным именем; *-сигнал* — эквивалент «-s сигнал»; *-номер_сигнала* — послать сигнал с указанным номером.

Операнды: идентификатор — идентификатор процесса или предваренный знаком «%» номер задания; *статус_завершения* — код завершения, возвращаемый прекращаемым процессом.

set — установить или сбросить ключи и позиционные параметры, вывести список установленных переменных

Синтаксис: set [-abCefmnuvx][-h][-o ключ][аргумент...]; set [+abCefmnuvx][+h][+o ключ][аргумент...]; set – [аргумент...]; set -o; set +o

Семантика: будучи подана без аргументов, set выводит имена и значения всех переменных оболочки, разделенные знаком «=», по одной на каждой строке.

Будучи подана с аргументами, set устанавливает или сбрасывает ключи и позиционные параметры текущей оболочки. Эта функциональность в настоящем курсе не рассматривается.

env — установить окружение для издаваемой команды

Синтаксис: env [-i][имя=значение]... [утилита [аргумент...]]

Семантика: будучи подана без аргументов, env выводит имена и значения всех экспортированных переменных, разделенные знаком «=», по одной на каждой строке.

Будучи подана с аргументами, env модифицирует текущее окружение и запускает в нем указанную команду. Эта функциональность в настоящем

курсе не рассматривается.

export — сделать переменные экспортируемыми

Синтаксис: export имя[=значение]... ; export -p

Семантика: оболочка устанавливает атрибут экспортируемости переменным, перечисленным в команде export, что включает их в окружение далее издаваемых команд. При указании ключа -p команда выводит выводит имена и значения всех экспортированных переменных, разделенные знаком «=» (если переменная установлена) или только их имена (если переменная сброшена), по одной на каждой строке, предваряя их цепочкой «export ».

Ключ: -p — вывести список экспортированных переменных и их значений.

nl — пронумеровать строки

Синтаксис: nl [-p][-b тип][-d ограничитель][-f тип][-h тип][-i приращение][-l номер][-n формат] [-s разделитель][-v нач_номер][-w ширина] [файл]

Семантика: nl читает строки из указанного файла (или стандартного ввода, если файл не указан) и выводит их в стандартный вывод, предваряя номерами.

Команда интерпретирует текст как последовательность логических страниц. Нумерация строк возобновляется на каждой странице. Страница состоит из верхнего колонтитула, тела и нижнего колонтитула (каждая из этих частей может быть пустой), строки в которых могут нумероваться независимо.

Начало каждого из разделов указывается строкой, не содержащей ничего, кроме цепочки «\:\:\:» (начало верхнего колонтитула), «\:\:» (начало тела) или «\:» (начало нижнего колонтитула). При отсутствии таких строк файл считается содержащим единственную страницу.

Ключи: ключи команды nl позволяют указать различные параметры нумерации и формата вывода, и в настоящем курсе не рассматриваются.

Операнд: файл — имя файла. При отсутствии операнда строки вводятся из стандартного ввода.

sort — сортировать, слить или проверить сортировку строк в файле

Синтаксис: sort [-m][-o вывод][-bdfinru][-t символ][-k опр_ключа]...

[файл...] ; sort -c [-bdfinru][-t символ][-k опр_ключа] [файл]

Семантика: sort выполнит одно из следующего: 1) отсортирует строки всех указанных файлов и выведет результат в указанный файл «вывод»; 2) сольет построчно (предварительно отсортированные) файлы и выведет результат в указанный файл «вывод» или 3) проверит единственный указанный файл на сортировку. Сравнения будут выполняться на основании указанных ключей или на основании строки в целом (вплоть до символа новой строки исключительно), с учетом порядка сортировки, определенного текущей локалью.

Ключи: ключи sort позволяют задавать параметры сортировки, слияния или проверки, и в настоящем курсе не рассматриваются. При отсутствии ключей строки ввода сортируются в прямом алфавитном порядке.

Операнд: файл — имя файла. Если файл не указан, сортируется стандартный ввод.

tee — дуплицировать стандартный ввод

Синтаксис: tee [-ai][file...]

Семантика: tee копирует стандартный ввод в стандартный вывод, дуплицируя его в один или более файлов.

Ключи: -a — добавить вывод в конец существующих файлов; *-i* — игнорировать сигнал SIGINT.

Операнд: файл — имя выходного файла.

vi — экранный (визуальный) текстовый редактор

Синтаксис: vi [-rR][-c команда][-t строка_тегов][-w размер][длина ...]

Семантика: vi является экранным текстовым редактором. Пользователь может переключаться между режимами vi и ex и издавать команды ex из vi.

Текущий редактируемый текст называется буфером редактирования. Все редактирование осуществляется в буфере редактирования и не затрагивает файлов до подачи команды записи. При использовании vi экран терминала представляет собой окно в буфер редактирования. Изменения в буфере редактирования отображаются на экране, а курсор отмечает позицию в буфере редактирования.

Важнейшие ключи: [[ex]]

-c *команда* — указать команду, которая будет исполнена после загрузки в

буфер первого файла; -r — восстановить состояние буфера после ошибочного завершения; -R — открыть файл только для чтения.

Важнейшие команды:

Переключение между режимами: командный режим — <Esc>; режим ввода — a (в режим вставки (вставка после курсора)), i (в режим вставки (с вставкой перед курсором)), A (в режим вставки (вставка после конца текущей строки)), I (в режим вставки (перед первым непробелом)), R (в режим замены); режим редактирования строки команды — :, /, ?, !.

Файловые команды: ZZ, :wq — сохранить текущий буфер в файл и завершить работу; Сохраняет файл и выходит из vi; :w — сохранить текущий буфер в файл; :w*имя* сохранить текущий буфер в файл с указанным именем; :q — выйти из редактора; :e имя — загрузить файл с указанным файлом в буфер редактирования; :n — загрузить следующий файл в списке файлов.

Команды перемещения: CTRL-d — на полстраницы вниз; CTRL-u — на полстраницы вверх; CTRL-f — на страницу вниз; CTRL-b — на страницу вверх; :0 — к началу файла; :*n* — к строке номер *n*; :$ — к концу файла ; 0 — к началу строки; ^ — к первому непробелу; $ — к концу строки; <Enter> — к началу следующей строки; - — к началу предыдущей строки; j — на следующую строку; k — на предыдущую строку.

Команды редактирования: CTRL-h — удалить символ слева; CTRL-w — удалить слово слева; CTRL-v — ввести непечатаемый символ; x — удалить текущий символ; D — удалить до конца строки; dd — удалить текущую строку.

Команды поиска и замены: /*текст* — искать вперед образец *текст;* ? *текст* — искать назад образец *текст;* n — повторить последний поиск в том же направлении; N — повторить последний поиск в обратном направлении; :s/*текст*/*новый_текст*/ — заменить первый образец *текст* на *новый_текст*; :s/*текст*/*новый_текст*/g — заменить все образцы *текст* на *новый_текст*.

1.12 Перечень стандартных команд ОС

Команда	*Примечания*	*Определение*
alias	МП	Определить или вывести синонимы
ar	СР *или* РАСШ	Создать или изменить библиотечные архивы
asa	ПФ	Перевести символы управления кареткой

Команда	*Примечания*	*Определение*
at	МП	Исполнить команды в указанное время
awk		Сканировать и обработать файл в соответствии с инструкциями
basename		Вывести часть полного имени файла, не относящуюся к имени каталога
batch	МП	Поставить команды в очередь на пакетное исполнение
bc		Вычислить арифметическое выражение с заданной точностью
bg	МП	Запустить задачи в фоновом режиме
break		Выйти из цикла for, while или until
c99	РС	Скомпилировать программу на стандартном Си
cal	РАСШ	Вывести календарь
cat		Объединить и вывести файлы
cd		Изменить текущий каталог
cflow	Р РАСШ	Построить блок-схему программы на Си
chgrp		Изменить группу-хозяина файла
chmod		Изменить атрибуты доступа к файлу
chown		Изменить хозяина файла
cksum		Вывести контрольные суммы и размеры файлов
cmp		Сравнить два файла
colon (:)		Ничего не делать
comm		Выбрать или отбросить совпадающие строки в двух файлах
command		Исполнить простую команду
compress	РАСШ	Сжать данные
continue		Продолжить выполнение цикла for, while или until

Команда	*Примечания*	*Определение*
cp		Копировать файлы
crontab	МП	Поставить задачу в расписание периодического исполнения
csplit	МП	Разделить файлы на контекстной основе
ctags	Р Ф МП	Построить файл меток
cut		Удалить указанные поля из каждой строки файла
cxref	Р РАСШ	Построить таблицу перекрестных ссылок программы на Си
date		Вывести или установить дату и время
dd		Преобразовать и скопировать файл
delta	Р РАСШ	Внести дельту (изменения) в файл SCCS
df	МП РАСШ	Вывести количество свободного места на диске
diff		Сравнить два файла
dirname		Вывести часть полного имени файла, относящуюся к имени каталога
dot (.)		Исполнить команду в текущем окружении
du	МП	Вывести информацию об использовании файлового пространства
echo		Вывести аргументы
ed		Редактировать текст
env		Установить окружение для запуска команды
eval		Составить команду из аргументов и исполнить ее
ex	МП	Редактировать текст
exec		Исполнить команду и открыть, закрыть или скопировать дескрипторы файлов
exit		Завершить исполнение оболочки
expand	МП	Перевести табуляции в пробелы

Команда	*Примечания*	*Определение*
export		Сделать переменную экспортируемой
expr		Вычислить аргумент как выражение
false		Вернуть значение «ложно»
fc		Обработать историю команд
fg	МП	Запустить задачи на переднем плане
file	МП	Определить тип файлов
find		Найти файлы
fold		Свернуть строки
fort77	Ф РФ	Скомпилировать программу на Фортране
fuser	РАСШ	Вывести идентификаторы процессов, имеющих открытые файлы
gencat	РАСШ	Построить каталог форматированных сообщений
get	Р РАСШ	Вывести версию файла SCCS
getconf		Вывести значения конфигурации
getopts		Разделить ключи утилиты
grep		Искать в файле образец
hash	РАСШ	Запомнить или вывести путь к утилите
head		Вывести начало файла
iconv		Преобразовать код файла
id		Вывести информацию о пользователе
ipcrm	РАСШ	Удалить очередь сообщений XSI, установки семафора или идентификатор разделяемой памяти
ipcs	РАСШ	Вывести состояние механизма межпроцессного обмена XSI
jobs	МП	Вывести состояние задач в текущем сеансе
join		Объединить базы данных
kill		Завершить процесс или передать ему сигнал

Команда	*Примечания*	*Определение*
lex	Р РС	Построить программу для лексического анализа
link	РАСШ	Вызвать системную функцию link
ln		Связать файлы
locale		Вывести информацию о локали
localedef		Определить переменные локали
logger		Занести сообщения в журнал
logname		Вывести регистрационное имя пользователя
lp		Послать файлы в печать
ls		Вывести содержимое каталога
m4	Р РАСШ	Обработать макросы
mailx		Отправить сообщения
make	Р СР	Построить, обновить или перестроить группы программ
man		Вывести системную документацию
mesg	МП	Разрешить или запретить вывод сообщений
mkdir		Создать каталоги
mkfifo		Создать специальные файлы очередей
more	МП	Вывести файлы постранично
mv		Переместить файлы
newgrp	МП	Перейти в другую группу
nice	МП	Запустить утилиту с указанным приоритетом
nl	РАСШ	Пронумеровать строки
nm	Р МП СР РАСШ	Вывести список имен, содержащихся в объектном файле
nohup		Запустить утилиту, не завершающуюся по сигналу HUP
od		Вывести дамп файла в указанных форматах

Команда	*Примечания*	*Определение*
paste		Слить соответствующие или последующие строки файлов
patch	МП	Изменить файлы на основе патча
pathchk		Проверить пути
pax		Обработать переносимый архив
pr		Вывести файлы, подготовленные для печати
printf		Вывести форматированный текст
prs	Р	Напечатать файл SCCS
ps	МП РАСШ	Вывести состояние процесса
pwd		Вывести имя текущего каталога
qalter	ПИ	Изменить пакетную задачу
qdel	ПИ	Удалить пакетные задачи
qhold	ПИ	Задержать пакетные задачи
qmove	ПИ	Переместить пакетные задачи
qmsg	ПИ	Послать сообщение пакетным задачам
qrerun	ПИ	Перезапустить пакетные задачи
qrls	ПИ	Отпустить пакетные задачи
qselect	ПИ	Выбрать пакетные задачи
qsig	ПИ	Отправить сигнал пакетным задачам
qstat	ПИ	Вывести статус пакетных задач
qsub	ПИ	Поставить сценарий в очередь на пакетное исполнение
read		Ввести строку
readonly		Защитить переменные от переустановки
renice	МП	Изменить приоритеты исполняемых процессов
return		Вернуться из функции
rm		Удалить файлы из каталогов
rmdel	Р РАСШ	Удалить дельту из файла SCCS

Команда	*Примечания*	*Определение*
rmdir		Удалить каталоги
sccs	Р РАСШ	Выполнить действия с файлами SCCS
sact	Р	Вывести текущую активность по редактированию файлов SCCS
sed		Редактировать файл поточно
set		Установить или сбросить ключи или позиционные параметры
sh		Запустить оболочку
shift		Сдвинуть список параметров
sleep		Задержать исполнение на указанный интервал времени
sort		Отсортировать, слить файлы или проверить сортировку файлов
split	МП	Разделить файлы на части
strings	МП	Найти в файлах символьные строки
strip	Р СР	Удалить из исполняемых файлов лишнюю информацию
stty		Установить параметры терминала
tabs		Установить позиции табуляции на терминале
tail		Вывести конец файла
talk	МП	Связаться с другим пользователем
tee		Дублировать стандартный ввод
test		Вычислить выражение
time	МП	Хронометрировать простую команду
times		Вывести время исполнения процессов
touch		Изменить временные атрибуты доступа и модификации файлов
tput	МП	Изменить характеристики терминала
tr		Подставить символы

Команда	*Примечания*	*Определение*
trap		Перехватывать сигналы
true		Вернуть значение «истинно»
tsort	РАСШ	Сортировать топологически
tty		Вывести имя терминала пользователя
type	РАСШ	Вывести описание типа команды
ulimit	РАСШ	Установить или вывести ограничение размера файла
umask		Установить или вывести маску прав на создаваемые файлы
unalias		Удалить определение синонима
uname		Вывести имя системы
uncompress	РАСШ	Разжать сжатые данные
unexpand	МП	Преобразовать пробелы в табуляции
unget	Р РАСШ	Откатить ранее выполненное взятие файла SCCS
uniq		Вывести или подавить вывод повторяющихся строк в файле
unlink	РАСШ	Вызвать функцию ulink
unset		Сбросить значения и атрибуты переменных и функций
uucp	РАСШ	Копировать из системы в систему
uudecode	МП	Декодировать двоичный файл
uuencode	МП	Закодировать двоичный файл
uustat	РАСШ	Вывести или изменить состояние uucp
uux	РАСШ	Выполнить команду удаленно
val	Р РАСШ	Проверить валидность файла SCCS
vi	МП	Редактировать файл в экранном режиме
wait		Ожидать завершение процесса

Команда	*Примечания*	*Определение*
wc		Сосчитать слова, строки или символы в файле
what	Р РАСШ	Идентифицировать файлы SCCS
who	МП	Вывести имена пользователей, зарегистрированных в системе
write	МП	Вывести сообщение на терминал другого пользователя
xargs	РАСШ	Составить из аргументов списки и запустить утилиту
yacc	Р РАСШ	Компилировать компилятор
zcat	РАСШ	Разжать и объединить данные

Примечания

1.Команды даны согласно ISO/IEC 9945-2001.

2. К субпрофилям стандарта относятся следующие пометы:

Помета	*Расшифровка*
МП	Среда, обеспечивающая мобильность пользователей
ПИ	Среда пакетного исполнения
Р	Среда разработки
РАСШ	Расширенная среда исполнения
РС	Среда разработки на Си
РФ	Среда разработки на Фортране
СР	Среда разработки
Ф	Фортран

Глава 2. Графический пользовательский интерфейс

До середины девяностых существовали отдельно компьютерная графика и отдельно — настольные игры в компьютерную графику. Помнящие историю отечественной школьной информатизации читатели, возможно, сталкивались с чудом техники под названием «цифровой дисплей растровый» (ЦДР), которое удавалось подключить к первому отечественному персональному компьютеру ДВК, чтобы отображать на экране телевизора несколько тысяч пикселов в четырех цветах. В то время в Лабораториях Компьютерной Графики некоторых вузов можно было встретить Графические Рабочие Станции с векторными устройствами и X-терминалами, и даже плоттерами.

Закон Мура тем временем делал свое (в данном случае, не черное, а многоцветное) дело, и к середине девяностых на компьютере с процессором Intel 486 уже запускалась та самая система, которую пятью годами раньше наблюдать можно было лишь на X-терминалах и графических станциях, стоивших каких-то немыслимых (что по тем временам, что сегодня) денег.

Игры в самодельную графику, конечно, продолжаются и сегодня, но в целом мир воссоединился, и особой нужды в таких играх давно нет. Хотя — такова диалектика массовых рынков — именно эти игры (включая игры в буквальном смысле) породили спрос на дешевые устройства (прежде всего, графические акселераторы), которые и делают настоящую компьютерную графику доступной пользователю массовой x86- и PowerPC-техники, даже устаревшие «персоналки» — сопоставимыми с X-терминалами, а более новые и мощные — соперниками и с профессиональных графических рабочих станций начального и среднего уровня.

2.1 Оконная система «Икс» и XFree86

Оконная система «Икс» — один из самых больших и успешных проектов в истории компьютерной техники — восходит к 1984 г., когда разработчики двух систем компьютерной графики, претендующих на универсальность — проектов Athena (Массачусетский технологический институт) и W Windowing (Стэнфордский университет) — решили объединить свои усилия. С тех пор практически каждая компания, серьезно занимающаяся графикой, считала своим долгом внести какие-либо разработки в систему, формальным «хозяином» которой в 1987 г. стал вновь созданный X Consortium (ныне Open Group, www.X.org).

С тех пор «Икс» прошла через одиннадцать основных релизов и множе-

ство версий.

Дальнейшее изложение ориентировано на свободную реализацию «Икс», которая называется XFree86, поддерживается одноименным партнерством (www.xfree86.org) и воплощает на сегодня версию 4.2 текущего релиза. XFree86 — самая популярная реализация «Икс», она поставляется в составе подавляющего большинства открытых систем (как свободных, так и несвободных) для x86-совместимых компьютеров, поддерживает беспрецедентно широкий спектр оборудования и, благодаря доступности исходных текстов и пользовательской аудитории в десятки миллионов человек, достаточно «вылизана», по крайней мере, насколько это возможно для такого разнообразия «железа». Несмотря на то, что исторически цифры «86» в названии пакета относятся к соответствующему семейству процессоров от Intel, современные версии XFree86 реализованы для большинства других популярных процессоров. XFree86 доступен и для некоторых альтернативных архитектур ОС, включая «Майкрософт Уиндоуз НТ».

Большинство нижесказанного справедливо для любой реализации «Икс» на любом оборудовании и под любой ОС, список которых можно найти на www.X.org.

2.2 Цветной сэндвич

То, что пользователю, сидящему за монитором, представляется сплошной графической операционной средой, реализовано как многослойный сэндвич технологий.

Непосредственно с оборудованием (видеосистемой, устройствами ввода и динамиком) работает Икс-сервер. Эта программа захватывает оборудование и предоставляет его возможности другим программам как ресурсы (собственно, именно поэтому она и называется сервером) по особому протоколу, который так и называется, Икс-протокол. Перечисленное оборудование в совокупности называется Икс-терминалом (аппаратным Икс-терминалом называется и специализированный компьютер, на котором исполняется исключительно Икс-сервер).

Здесь сразу видно отличие «Икс» от большинства самодельных систем графики, используемых в проприетарных системах: взаимодействие Икс-сервера с его многочисленной клиентурой происходит по специфицированному протоколу, который может туннелироваться через TCP/IP и, соответственно, клиенты и сервер могут исполняться на разных узлах Сети. Это означает, что одни и те же программы могут эксплуатироваться в разных топологиях, включая совокупность автономных рабочих станций («персональных компьютеров»), совокупность рабочих станций без дан-

ных или бездисковых рабочих станций («локальная сеть»), многопользовательскую систему с Икс-терминалами (или какую-либо гибридную топологию).

Еще одним ресурсом, который предоставляет Икс-сервер, являются шрифты. Оперировать шрифтами он может самостоятельно, либо с помощью другой программы, которая называется фонт-сервером и обеспечивает их масштабирование.

Большинство пользователей, установив систему, получают в свое распоряжение готовую графическую среду. Мы поступим иначе — будем разбираться с ней по слоям.

2.3 «Чистая» «Икс»

На Рис. 2-1 изображена «голая» оконная система «Икс» — то, с чем большинство пользователей никогда не сталкивается. Запустить ее обычно можно, подав команду: «X &».

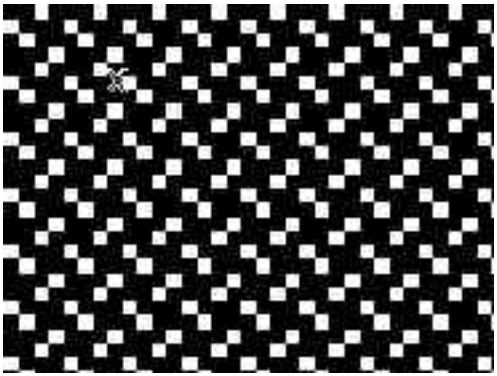

Рис. 2-1

Мы видим традиционный серый экран с не менее традиционным курсором в виде буквы «x». Используя мышь или другое координатное устройство, курсор можно перемещать по экрану. На нажатие кнопок мыши и клавиш никакой видимой реакции не следует. И невидимой тоже — сервер готов передавать эти сигналы своим клиентам, а клиенты пока не запущены. Хотя на самом деле некоторые комбинации клавиш «Икс» перехватывает и обрабатывает. Это Zap (Control-Alt-Backspace) — завершение работы сервера (если эта возможность не запрещена при конфигурации), Zoom (Control-Alt-+ и Control-Alt--) — «горячее» переключение доступных видеорежимов. В некоторых ОС (Например, «ГНУ/Линукс») Control-Alt в сочетании с функциональной клавишей освобождает оборудование и передает его на время соответствующей виртуальной консоли.

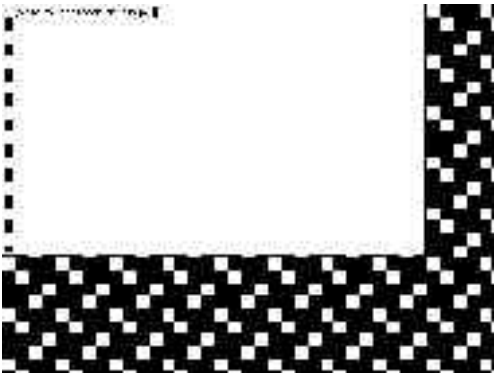

Рис. 2-2

Воспользуемся последней возможностью, перейдем на консоль и запустим первое клиентское приложение: программу «xterm» (Рис. 2-2). На экране «Икс» появилось окно, а в окне можно видеть интерфейс клиентского приложения. В данном случае интерфейс текстовый, а приложение — эмулятор терминала, на котором запущена диалоговая оболочка системы по умолчанию. С эмулятором можно делать все то же, что и с обычным терминалом: издавать команды, получать результат и запускать другие программы. Если программы текстовые (строчные или оконные), исполняться они будут в том же окне, а если графические (как и сам «xterm») — в отдельных окнах.

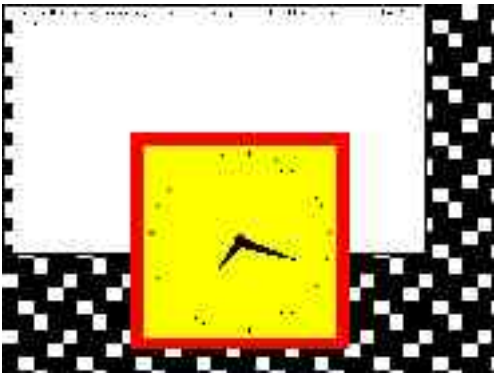

Рис. 2-3

Запустим программу «xclock» (Рис. 2-3). При ее запуске мы использовали несколько параметров, задающих геометрию (местоположение и размер) вновь порождаемого окна, цвет его фона и шрифта по умолчанию, толщину и цвет рамки. Эти (и некоторые другие) параметры типичны для программ, построенных на основе графической библиотеки «X Toolkit».

Значения параметров, заданные при вызове программы, могут быть перекрыты самим запускающимся приложением, кроме опции геометрии. Дело в том, что окно выделяется клиентскому приложению при запуске, и все доступные ему ресурсы этим окном и ограничены — это свойство X-протокола.

Запустив несколько экземпляров того же «xterm» (и почитав документацию) можно обнаружить, что и «голышом» «Икс» умеет не так мало. Например, оперирует буфером обмена текстом между приложениями и предоставляет текстовым приложениям такой ресурс, как полосу прокрутки (забавная полоска, скроллировать текст с помощью которой вверх или вниз можно, щелкая по ней разными кнопками мыши, — это наследие проекта «Athena»).

Есть ли польза от системы, работающей с фиксированными окнами? Да, если вспомнить, что «универсальный десктоп» — не единственная сфера применения компьютера. Можно запустить при загрузке «Икс» и браузер на весь экран и получить гипермедийный киоск по цене персонального компьютера. А можно посадить за тот же браузер оператора, который будет через него весь день «рулить» базу данных.

Но мы пойдем дальше. Итак, основная работа Икс-сервера — создавать окна и предоставлять клиентским приложениям возможности работы в них. Для того, чтобы работать с окнами, нужна другая программа, которая так и называется — менеджер окон (window manager).

2.4 Окноводы

Как же менеджер окон преодолевает указанное ограничение X-протокола? Никак — просто выделенным ему окном является весь экран. (На самом деле, менеджер окон — не единственная программа, способная работать с «корневым» окном; например, входящая в комплект поставки «xsetroot» позволяет установить цвет фона или поместить на него рисунок.)

Менеджеров окон существует превеликое множество — под любой набор задач, которые может решать графическая многооконная система. Их настолько много, что выбрать какой-нибудь в качестве «типичного представителя семейства» затруднительно. Поэтому выберем один из самых развитых — Enlightenment.

«Просвещение» (www.Enlightenment.org) создано Карстеном Хайцлером и Джеффом Харрисоном (Carsten Haitzler, Geoff Harrison) и его текущая версия — 0.16.5. До 2000 г. он был «штатным» менеджером окон в популярной среде «Гном» (см. раздел 2.17), затем уступив это место менее

функциональной, но более быстрой «Рыбе-пиле» (Sawfish). Он продолжает оставаться «Гном»-совместимым, и многие пользователи этого популярного десктоп-менеджера предпочитают его, хотя и без «Гном» у Enlightenment поклонников хватает.

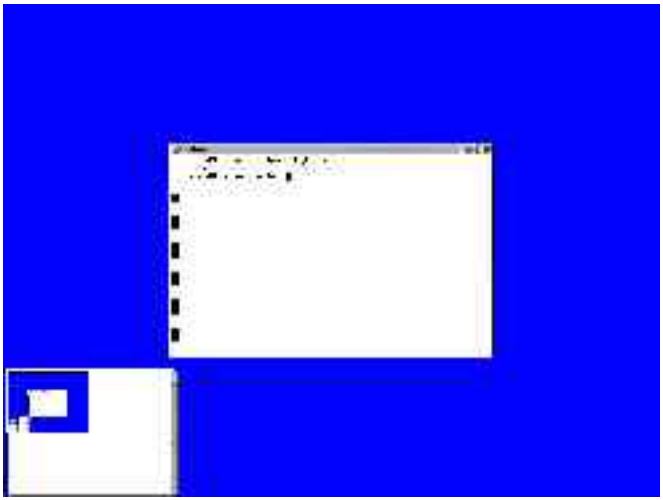

Рис. 2-4

Запустим «Просвещение» (Рис. 2-4). Как резко изменилась картина!

Первое, что мы видим — это появившиеся вокруг окна нашего xterm «виджеты» (элементы окон) — строка заголовка с кнопками и рамка. Окно теперь можно перемещать по экрану, «ухватив» за заголовок, масштабировать, «взяв» за бок или за угол, максимизировать, минимизировать или закрыть, нажав соответствующую кнопку. Спрашивается, что еще можно делать с окном?

Рис. 2-5

Вопрос не праздный. Нажав на левую кнопку в заголовке, получаем неожиданно разнообразное меню (меню — это тоже «виджет») таких действий (Рис. 2-5). Оказывается, его можно еще уничтожить (Annihilate), поднять/опустить (Raise/Lower), оттенить/растенить (Shade/Unshade) приклеить/отклеить (Stick/Unstick) и выполнить еще массу действий, для которых потребовались отдельные меню! Набор

этих действий зависит от конкретного менеджера окон (и Enlightenment — один из самых богатых возможностями), а то, какие из них выведены в строку заголовка отдельными кнопками — вообще от его настройки.

Собственно, управление окнами — основная функция оконного менеджера, и на этом его функциональность может и заканчиваться. Однако большинство из них выполняют по крайней мере еще одну функцию.

Рис. 2-6

Вы уже обратили внимание на то, что при запуске «Просвещения» на экране появилось еще одно окно. Это так называемый пейджер (pager), на Рис. 2-6 он изображен крупным планом. На пейджере представлена миниатюрная копия экрана, обновляющаяся в режиме реального времени, причем, если подвести курсор к изображению отдельного окна, оно увеличивается и рядом высвечивается название приложения, запущенного в нем. Но почему экран занимает только четверть окна пейджера? Потому что оконный менеджер позволяет оперировать «виртуальным столом, по размеру превышающим физический экран, а пейджер — одно из средств перемещения физического экрана по рабочему столу. Enlightenment позволяет создавать до 64 экранов на рабочем столе.

Рис. 2-7

Еще один важный компонент Enlightenment мы не увидели сразу: это меню настройки самого менеджера, которое можно «достать», щелкнув правой кнопкой мыши на фоне экрана (Рис. 2-7). Порывшись в настройках, можно обнаружить, что вышесказанное о способах оперирования с этим менеджером весьма условно, потому что поменять можно буквально все, от декора виджетов до количества и функций элементов оформления окон и их реакции на различные действия.

Лишь один пример: сколько способов визуализировать перемещение окна вы знаете? Разработчики «Просвещения» придумали целых шесть,

включая фантастический «полупрозрачный». Настройки и расширения Enlightenment можно объединять в «темы» (themes) и обмениваться ими.

Собственно, на этом функции оконного менеджера как такового и заканчиваются, а дальше Enlightenment вторгается во владения другого класса программ — менеджеров рабочего стола...

2.5 Столоначальники

...Что демонстрирует отсутствие резкой границы между ними. Существует два подхода к тому, чтобы достроить оконную систему до полнофункциональной среды. Первый — добавить в «графический сэндвич» еще один слой — менеджер рабочего стола — работающий «поверх» оконного менеджера и использующий функциональность последнего. Этим путем идут команды разработчиков «ГНОМ» (см. раздел 2.17) и «КДЕ» (см. раздел 2.18).

Другой путь — «дотянуть» до полнофункциональной среды функциональность самого оконного менеджера, и им идет «Enlightenment» и ряд других проектов.

Чего нам не хватает до полнофункциональной среды? Менеджера программ и утилит. Так вот, в «Просвещении» есть и такая функциональность, доступная (по умолчанию) по щелчку на фоне левой кнопкой.

Комментировать здесь особо нечего: пункты меню позволяют запустить множество различных приложений, причем, кроме независимо разработанных, и целую пачку «эпплетов», поставляемых вместе с «Enlightenment». Альтернативный способ запуска — через «панель» — встроен в некоторые темы «Просвещения».

Откуда берутся такие ресурсы, как «виджеты» с их декором и способом поведения? Конечно, менеджер окон может содержать их в себе. Но такой подход не очень характерен для открытых систем, одним из принципов разработки которых является компонентность. Большинство развитых оконных менеджеров, менеджеров рабочего стола и «заточенных» под них приложений можно сгруппировать по библиотекам (toolkits), с опорой на которые они разработаны.

2.6 Триумф интерфейса над пользователем?

Косметических улучшений за тридцать лет существования парадигмы WIMP была придумана масса, а вот более или менее серьезных, при внимательном анализе, обнаруживается только два: интеграция звука (и превращение графической (визуальной) среды в сенсуальную) и начало эксплуатации концепции гиперссылок, в терминах которых можно пере-

формулировать почти весь интерфейс.

Фредерик Брукс еще в 1995 г., обсуждая основные процессы, произошедшие в программной отрасли за 20 предшествовавших лет, назвал в числе «наиболее впечатляющих явлений» «триумф интерфейса WIMP» [17, сс. 239-243]. В этом ставшем классическим четырехстраничном анализе (всем, интересующимся темой, крайне рекомендуется прочитать эти четыре страницы, а заодно — и всю книгу), Брукс:

- производит декомпозицию самой идеи («диалог» с системой: объекты-«существительные» и действия-«глаголы»),
- выделяет факторы, способствовавшие ее «триумфу» («концептуальная целостность через метафору» «рабочего стола»; эквивалентность клавиатурных команд пунктам меню, обеспечивающая постепенный переход от новичка к опытному пользователю; навязывание архитектуры через средства разработки),
- называет ограничения метафоры «рабочего стола» («проблема двух курсоров»), а также
- предрекает устаревание WIMP при внедрении речевого интерфейса («WIMP через поколение станет достоянием истории. Указание курсором останется способом задания существительных при управлении нашими компьютерами. Для выражения глаголов станет использоваться речь»).

Прошло еще пять лет, и мы можем отметить, что:

- Проф. Брукс не заметил решения «проблемы двух курсоров» (а заодно — и непротиворечивой интеграции командной строки в графико-интерфейсное окружение) в конце восьмидесятых в Norton Commander (и сонме последователей этой замечательной программы на разных платформах (обзор см. в [15]. Проф. Безруков предложил для реализованного в Norton Commander интерфейса термин «ортодоксальный менеджер файлов» (OFM));
- WIMP не думает устаревать, и скорее сам абсорбирует новые интерфейсные возможности (включая распознавание речи), чем будет вытеснен ими;
- и, наконец, самое серьезное — это то, что «триумф WIMP» на сегодня выглядит не то чтобы менее бесспорным, а менее однозначным, все более походя на пресловутое «триумфальное шествие советской власти» по обессиленным Первой мировой войной частям Российской империи и ее окрестностей. Во многих прикладных областях попытки внедрения WIMP стали скорее частью проблемы пользовательского интерфейса, чем частью ее решения.

«Сплошной» же WIMP-среды и вовсе нет нигде, кроме встроенных/специализированных систем: в любом окружении, претендующем даже не на универсальность, а просто на широкую сферу применения, элементы WIMP сочетаются с элементами другой интерфейсной модели — командно-строчной — иногда более органично (OFM, AppleScript и т.п.), а чаще эклектично, противоречиво и с фатальным для производительности исходом (фрагменты «рваной» командной строки в «диалоговых окнах», разнообразные Wizards и «окна установки предпочтений»).

Если перечитать текст доклада, в котором идеи WIMP впервые были представлены широкой публике [16], станет понятно, почему: модель WIMP предлагалась как средство непосредственного манипулирования конкретными объектами («взять это и положить туда», «изменить такое-то свойство того-то объекта», а не как средство формулирования абстрактных положений и команд («все ли файлы, лежащие в каталоге X, имеют формат Y?», «удалить все файлы, созданные до 01.01.2000 в которых упоминается Борис Ельцин» и т.п.). Соответственно, сделать WIMP-рабочее место для выполнения технических процедур, «рабского», неквалифицированного труда можно, а вот систему поддержки полноценной свободной практики — затруднительно.

2.7 От какого наследства нам не стоит отказываться?

Виктор Вагнер [18] противопоставляет «рыхлости» модели WIMP, пусть и целостной метафорически, концептуальную целостность командно-строчного интерфейса, основывающуюся на четырех принципах:

- универсальности формы представления информации (текстовый файл, понимаемый как последовательность символов, некоторые из которых разделяют строки (записи), поля и слова);
- возможности переназначения ввода-вывода и соединения программ каналами;
- философии «набора инструментов» (одна утилита — одна функция);
- наличия в оболочке механизма регулярных выражений.

По Вагнеру, по-настоящему успешным графическим интерфейсом («True UNIX GUI») будет интерфейс, предлагающий не менее целостную и последовательно реализованную концептуальную основу. Причем, предлагающий ее не только и не столько конечному пользователю, сколько разработчику, т.е. реализованный начиная с системы быстрой разработки (СБР, RAD). В упомянутой статье Вагнер рассматривает несколько кандидатов на роль универсальной формы представления информации в графической среде и рассуждает о том, какие принципы могли бы стать ана-

логами другим «китам», на которых покоится командно-строчный интерфейс.

На самом деле, существует целый ряд систем, в той или иной степени закладывающих основу «интерфейсов следующего поколения». К сожалению, ни одну из них нельзя назвать на сегодня массовой, кроме, возможно, языка описания интерфейса XUL, использованного в «Мозилла» (см. гл. 3), но и для XUL пока нет СБР.

2.8 Зачем нужны «легкие» среды?

В то время, как сама оконная система «Икс» много лет является фактическим отраслевым стандартом, лежащие «над» нею слои графической среды не стандартизованы.

Какую-либо классификацию графических сред дать затруднительно, однако самым грубым образом их можно разделить на «интегрированные» и «легкие».

Оборотной стороной интегрированности является достаточно высокая их требовательность к ресурсам. Комфортная работа с «КДЕ» или «Гном» «свежего разлива» начинается примерно от производительности, эквивалентной производительности 800 МГц процессора Celeron, отказ от некоторых ресурсоемких свойств (анимация изменений в среде и т.п.) позволяет «снизить планку» примерно до 500 МГц при объеме оперативной памяти от 128 МБ. Разумеется, эти цифры даже ниже характерных для компьютеров «стартового уровня», поставляемых сегодня производителями, однако парк машин, находящихся в эксплуатации, как в офисах, так дома и в школе, включает и компьютеры с более низкими характеристиками.

Так, в школе весьма желательно предоставить возможность работы в графической среде на менее мощных машинах. Здесь помогут «легкие» графические среды, представляющие собой оконные менеджеры с несколько расширенными возможностями (базовая и расширенная функциональность типичных оконных менеджеров описана ниже).

Обсуждаемые сегодня IceWB, BlackBox и fluxBox (а также чуть более требовательный к ресурсам WindowMaker)[63] позволяют достаточно комфортно работать с графикой на машинах производительностью (в эквиваленте Intel Pentium) примерно от 100 МГц и с памятью от 32M (текст одного из предыдущих разделов набирался в поезде, с одновременной

[63] Возможностью представить их обзор в компактном виде автор обязан прежде всего своим соавторам по [3] Егору Гребневу, Сергею Иванову, Михаилу Шигорину.

«съемкой» изображений с его экрана графическим редактором «ГИМП» (см. главу 5) на ноутбуке с процессором Intel Pentium MMX 166 и ОЗУ объемом 64МБ).

Следует оговориться, что отказ от интегрированных графических сред не является «волшебной палочкой»: конкретные прикладные программы могут быть сами по себе достаточно требовательными к ресурсам. Так, на упомянутом ноутбуке запуск word-процессора «OpenWriter» занимает более минуты (хотя дальнейшая работа не сопряжена с существенными задержками). Кроме того, если прикладная программа изначально создана в ориентации на определенную интегрированную среду, она может интенсивно использовать соответствующие библиотеки, даже если запускается в «легкой» среде. Например, запуск программ из пакета KOffice в «легкой» среде, на самом деле, дает небольшой выигрыш по сравнению с его запуском из «родной» для него среды «КДЕ».

(Если необходимо задействовать имеющийся парк «слабой» техники для таких задач, а также, если необходимо сохранять в эксплуатации еще менее производительные машины (например, старшие модели IBM PC-совместимых компьютеров на базе процессоров Intel 486 или AMD 586 или «Макинтоши» на процессорах Motorola 68K), следует подумать об использовании такой техники в режиме графических терминалов или, по крайней мере, варианте запуска наиболее «тяжелых» прикладных программ на сервере.)

Следует оговорить также, что ограниченность аппаратных ресурсов не является единственным мотивом применения «легких» графических сред. Каждая графическая среда, интегрированная или легкая, обладает собственными уникальными особенностями, собственным стилем, и уместность пользования конкретной средой в значительной степени зависит от набора задач, решаемых на компьютере конкретным пользователям, и от его личных предпочтений.

2.9 Базовая функциональность оконного менеджера

Как уже говорилось, ключевой компонент графической платформы — Икс-сервер:

- захватывает оборудование,
- создает по запросу других программ (которые в этой терминологии называются X-клиентами) окна и
- предоставляет другим программам возможность работы в окнах, то есть вывода информации в эти окна и обработки сигналов от устройств ввода (клавиатуры и «мыши» или другого координатного устройства), когда окно, назначенное программе, является

активным. Предоставление ресурсов возможно в том числе и через сеть, когда клиент и сервер работают на разных компьютерах (узлах).

В «голой» среде, образуемой Икс-сервером без оконного менеджера, окно, выделяемое клиенту, является фиксированным: его геометрия (местоположение на экране и размер) задается при запуске клиента и сохраняется в течение всего сеанса работы с этим клиентом. Это вполне соответствует цели создания специализированных систем с графическим интерфейсом пользователя (таких, как мультимедийные киоски и т.п.), но совершенно недостаточно для универсального «настольного» применения.

При универсальном применении компьютера характерна поочередная работа с различными программами (иногда достаточно большим их количеством), причем пользователь может отрываться, допустим, от редактирования текста, чтобы поработать другой программой с иллюстрацией, прочитать почту или заглянуть на WWW-страницу, затем возвращаться к редактированию текста и т.д. Графическая операционная среда должна быть достаточно гибкой, чтобы допускать и поддерживать такой, «субъектно-ориентированный», а не ориентированный на строго последовательное выполнение предзаданных процедур, стиль работы.

В частности, должно поддерживаться управление (с помощью клавиатуры или «мыши») окнами, т.е. возможность изменять «на лету» их геометрию (положение и размеры), а также (обычно не относимое к геометрии) положение в воображаемой «стопке» окон, т.е. определение того, какое из окон будет «верхним» (видимым полностью), если окна перекрывают друг друга на плоскости экрана.

Управление окнами и составляет базовую функциональность оконного менеджера (устоявшийся термин window manager, относящийся к этому классу программ, будет передаваться далее словосочетанием-калькой «оконный менеджер», которое, впрочем, не представляется особенно удачным, так же, как и встречающиеся в литературе «менеджер окон» и «администратор окон»).

Технически ограничение на изменение геометрии однажды выделенного окна преодолевается оконным менеджером за счет того, что ему в качестве окна выделяется весь экран.

Прикладным программам, таким образом, выделяются далее уже не окна собственно X, а окна оконного менеджера. Для них это совершенно прозрачно, хотя желательно, чтобы программа была достаточно «сообразительной», чтобы изменить свое поведение при изменении размеров вы-

деленного ей окна «на лету» (изменение положения окна в подавляющем большинстве случаев ничего от клиента не требует), это справедливо для большинства, но не для всех программ (в частности, этого не «умеют» многие старые программы и некоторые компьютерные игры).

В свою очередь, и оконный менеджер может быть достаточно «умен», чтобы понять, что программа не реагирует на изменение геометрии, и заблокировать возможность изменения размеров окна пользователем (чтобы он не оказался в ситуации, когда ему видна лишь часть области вывода программы или наоборот, часть окна прикладной программы пуста). Однако такое решение может привести к весьма дискомфортным ситуациям (например, если при запуске программы ее окно оказывается больше экрана)[64].

2.10 «Виджеты»

Базовая (а также расширенная) функциональность оконных менеджеров доступна пользователю прежде всего за счет введения в интерфейс так называемых «виджетов» (widgets = window gadgets, «оконные приспособления») — таких визуальных элементов, как рамки, кнопки, меню и пр., которые служат «органами управления» окна. Технически виджеты представляют собой отдельные окна (в терминах оконной системы «Икс»), примыкающие к окну прикладной программы и, как правило, перемещающиеся вместе с ним.

В пользовательской перспективе виджеты, составляющие обрамление окна, часто воспринимаются как его часть. Однако не следует забывать, что внутри окна (содержимым которого управляет прикладная программа) зачастую тоже есть свои виджеты: кнопки, полосы прокрутки, переключатели, меню и т.п. В общем случае, используемые оконным менеджером и прикладной программой библиотеки виджетов могут и не совпадать.

---

[64] Следует отметить, что большинство базовых функций оконных менеджеров при исполнении опирается на поддержку оконной системой X функций двумерной графической акселерации (ускорения), реализованных практически во всех современных графических адаптерах. В отличие от трехмерной акселерации, полезной лишь для достаточно узкого круга приложений (программ трехмерного моделирования, компьютерных игр), двумерная акселерация — действительно универсальна полезна для графического пользовательского интерфейса. При использовании карты без двумерного ускорения или карты, чья акселераторная функциональность не поддерживается системой X, можно рекомендовать настройку среды для исключения, например, визуализации перемещения окна со всем его содержимым, дабы избежать неоправданного роста нагрузки на процессор и драматического падения производительности.)

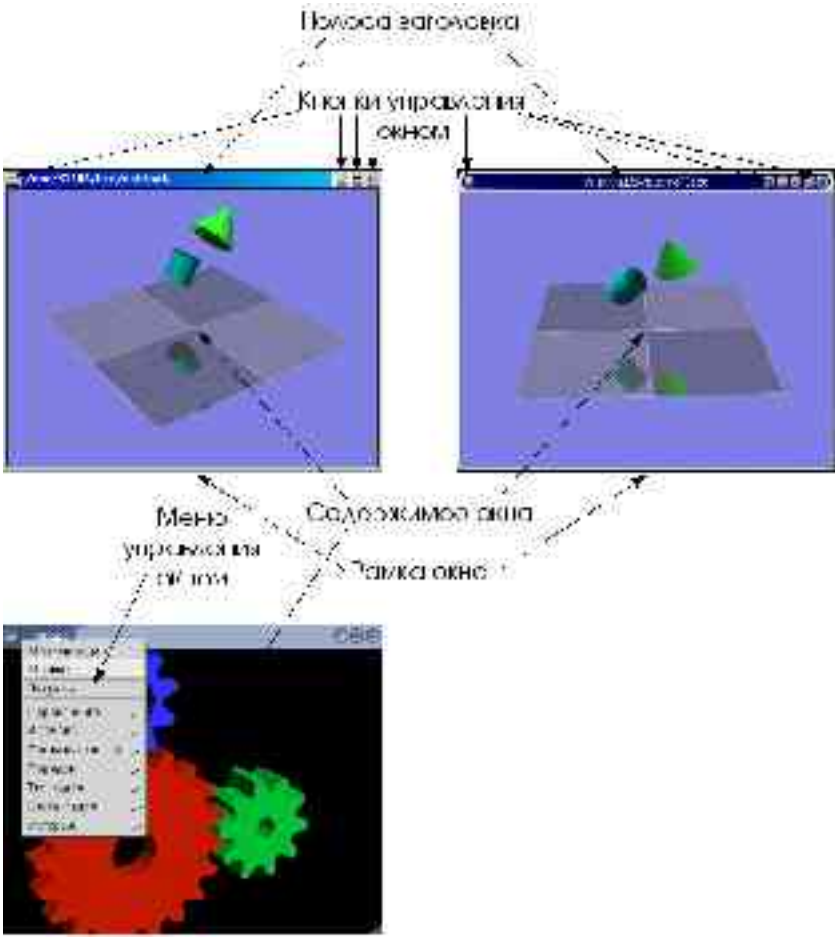

Рис. 2-8

(Зачастую при проектировании выдвигается требование единства стиля органов управления и согласованного управления изменением этого стиля (например, для настройки среды для пользователя с ограниченными возможностями: со слабым зрением, нарушением моторики и т.п.), и в этом сильно выигрывают интегрированные графические среды «Гном» и «КДЕ», используемые совместно с прикладными программами, основанными на тех же графических библиотеках и наследующими те же настройки. Однако на практике ограничиться набором программ, основанных на одной библиотеке графических примитивов, бывает трудно, поэтому разумно познакомить учеников с особенностями по крайней мере самых распространенных из них.)

Обрамление окна обычно включает:

- рамку, окружающую окно. При «буксировке» рамки мышью окно изменяет свой размер. Иногда для изменения размера окна предназначены только выделенные «уголки» рамки, представляющие собою отдельные виджеты;
- полосу заголовка, часто совпадающую с одной из (обычно, верхней) сторон рамки. Полоса заголовка может содержать название программы, команду, ее запустившую, или другую информацию, специфичную для окна. При «буксировке» полосы заголовка перемещается все окно. Со «щелчками» различными кнопками мыши на полосе заголовка также могут быть связаны различные действия по управлению окнами;
- кнопки управления окном. Часто вынесенные на полосу заголовка или в другое место рамки кнопки позволяют выполнить с ним такие действия, как закрытие (часто сопровождающееся выходом из программы, открывшей окно), максимизация (разворачивание окна на весь экран), минимизация/сворачивание (см. ниже расширенную функциональность), вызов меню управления окном, которое может содержать весьма обширный репертуар других действий.

Детали реализации обрамления окна могут быть весьма различными в зависимости от конкретного оконного менеджера и его настроек.

2.11 Расширенная функциональность оконного менеджера

Собственно, перечисленными функциями оконный менеджер, предназначенный для работы в составе интегрированной операционной среды, может и ограничиться. При использовании же в качестве операционной графической среды самого оконного менеджера, крайне полезной может оказаться его расширенная функциональность. К ней можно отнести:

- минимизацию/сворачивание окон и управление свернутыми окнами. Работа на «столе», «захламленном» десятком различных окон, может быть дискомфортной, и крайне полезна возможность «свернуть» или «минимизировать» окно со временно неиспользуемой программой. Для того, чтобы средствами графической среды можно было окно затем развернуть, оно и в свернутом состоянии должно каким-то образом визуализироваться. Существует несколько относительно распространенных способов визуализации свернутых окон. Например, «на столе» может оставаться полоса заголовка свернутого окна, по щелчку на которой оно вновь разворачивается. Свернутым окнам могут соответство-

вать пиктограммы («иконки», «значки») на поверхности «рабочего стола» или в специально отведенном для этого окне («панели управления»). Свернутые окна могут визуализироваться как пункты общего или специального меню (см. ниже);

- управление несколькими «столами». Практика показывает, что для многих продвинутых пользователей, для которых освоение стандартных систем следует за освоением специфически персонально-компьютерных, именно возможность работать на нескольких «столах» оказывается первым «убойным приложением» оконной системы «Икс». Действительно, переключение между виртуальными «столами» позволяет организовать комфортную работу со множеством программ даже на мониторах с относительно низким разрешением (1024x728, 800x600) и физическими размерами (17, 15-дюймовыми). В иных условиях комфортность работы существенно снизилась бы, или настоятельной необходимостью стало бы приобретение более крупного и емкого монитора (что зачастую влечет за собой необходимость смены графической карты и прочих недешевых мероприятий). Все современные оконные менеджеры поддерживают виртуальные столы, правда называются эти сущности в них по-разному: «столами», «рабочими областями» или «экранами». До предела (чтобы не сказать, до абсурда) эта функциональность развита в оконном менеджере Enlightenment, упоминавшемся в одном из предшествующих разделов: E позволяет организовать до 64 «экранов» на «рабочем столе», при этом «рабочих столов» также может быть более одного (точнее, до 32). Трудно представить, зачем может понадобиться две тысячи с лишним отдельных экранов (как правило, четырех экранов хватает с избытком для любых практических задач), однако возможности приема демонстрируются этим «в полный рост»;
- быстрый запуск команд. Возможность быстрого запуска предуготовленных команд обычно ассоциируется с общим меню, вызываемым «щелчком» на особом виджете, не связанном с прикладными окнами, или в свободной от прикладных окон области экрана;
- возможности настройки «поведения» и внешего вида среды. «Поведение» (реакция отдельных виджетов на операции с ними, модель фокусировки (связывание ввода с клавиатуры и «мыши» с теми или иными программами) и т.п.) и внешний вид оформления окон, а также наличие на экране «общих» виджетов, не связанных с конкретными прикладными окнами, «обои» и т.п. могут варьировать в очень широких пределах. Иногда возможности та-

кой настройки считают некими «архитектурными излишествами», однако более взвешенной является точка зрения, согласно которой в хорошем визуальном дизайне (так же, как и в хорошей архитектуре) ничто не является излишеством. В частности, в школе программы эксплуатируются на широком спектре оборудования с весьма разными характеристиками и разного качества, причем используются они широким кругом людей с различными психофизическими особенностями, как в пределах нормы, так и связанных со здоровьем, и игнорировать возможности настройки нельзя даже уже исходя из гигиенических соображений.

* * *

Выше при характеристике тех или иных (предположительно, общих) характеристик оконных менеджеров чаще обычного употреблялись слова «обычно», «как правило», «может» и т.п. Это связано с чрезвычайным разнообразием решений на базе распространенных оконных менеджеров. Ниже самые распространенные из них характеризуются более подробно и определенно.

2.12 Оконные менеджеры «BlackBox» и «FluxBox»

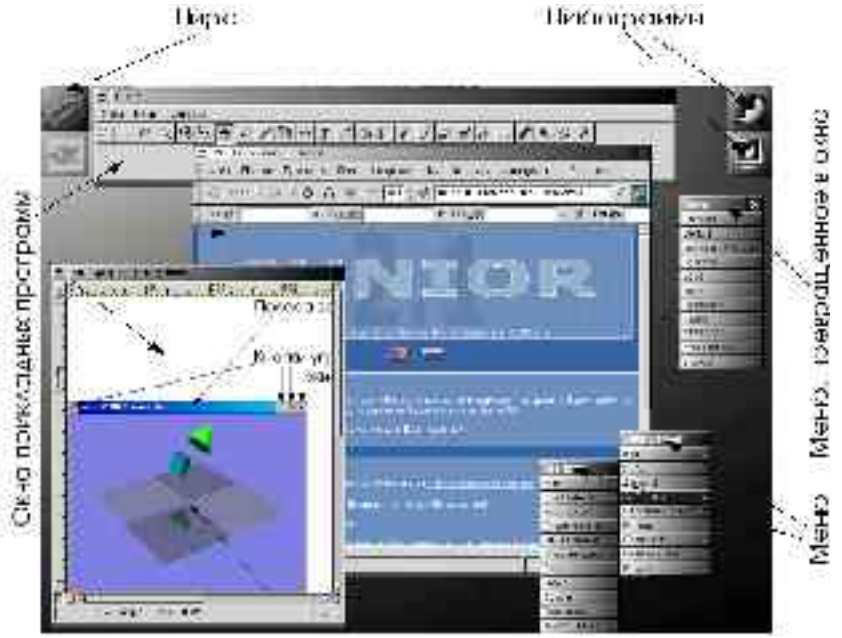

Рис. 2-9

«BlackBox» («BB») — один из самых компактных, «минималистичных» и быстродействующих оконных менеджеров. Он позволяет эффективно организовать работу на «рабочем столе», не «захламляя» его ненужными ссылками и не расходуя экранное пространство на отображение громоздких элементов оформления.

Наряду с базовой функциональностью, «BB» предоставляет (факультативно) панель, содержащую кнопки переключения между «рабочими столами» (по умолчанию их четыре) и заголовки открытых окон. Общее меню вызывается «щелчком» правой кнопкой «мыши» на свободном от окон месте «стола». Меню (или любое из вложенных в него меню) «щелчком» по заголовку может быть превращено в окно, остающееся на экране до явного его закрытия щелчком на соответствующей кнопке.

По умолчанию на полосе заголовка каждого окна присутствуют кнопки сворачивания (сворачивание можно выполнить также двойным «щелчком» на самом заголовке), максимизации и закрытия окна. Свернутое окно присутствует на экране в виде полосы заголовка, развернуть его можно повторным двойным «щелчком» на полосе заголовка или из меню Workspaces («рабочие пространства»), доступного по «щелчку» средней

кнопкой мыши на свободном от окон месте «стола». Это же меню позволяет перейти на другой «стол», добавить или удалить «стол» из рабочего пространства.

«BB» поддерживает различные модели фокусировки ввода. «Click to focus» («фокусировка по щелчку») позволяет реализовать стиль работы, привычный для пользователей «Гном», «КДЕ» или «Майкрософт Уиндоуз»: окно становится активным (принимающим текущий ввод с клавиатуры и от «мыши») после «щелчка» на нем. Активное окно автоматически становится «верхним» (видимым полностью, даже если оно частично перекрывается с другими окнами). Модель «Sloppy focus» («небрежная фокусировка») предполагает активизацию окна при попадании на него курсора мыши (окно при этом не «всплывает» автоматически наверх).

Наряду с панелью и конвертируемыми в дополнительные окна-«панели» меню, «BB» реализует еще один автономный виджет — так называемую «щель» (Slit). «Щель» располагается на краю видимого экрана и может содержать маленькие (без обрамлений) окна специализированных программок (их существует около десяти), индицирующих какие-либо состояния среды или позволяющих быстро выполнить часто исполняемые действия.

На основе «BB» созданы два более развитых оконных менеджера — «OpenBox» и более популярный «FluxBox».

«Наиболее характерная особенность «Fluxbox» — реализация закладок (tabs) в контексте рабочего стола. Если закладки в браузере позволяют одновременно открыть несколько страниц в одном окне, то закладки fluxbox позволяют удобно сгруппировать несколько окон на столе. Все окна в группе имеют одинаковые размеры и расположены строго одно под другим. Для переключения на какое-либо из них достаточно навести курсором мыши или щелкнуть (в зависимости от настроек) по соответствующей закладке. К примеру, мне приходится работать с несколькими различными почтовыми клиентами. Совместив их в одну группу, я могу легко переключаться между ними и при этом я всегда знаю, где расположено каждое окно. На словах объяснить преимущества этого оригинального подхода не очень легко, но после нескольких дней практического использования, становится трудно без него обходиться: к хорошему привыкаешь быстро.» [3]

Внешний вид «BB», «FluxBox» и «OpenBox» легко настраивается с помощью механизма «тем» рабочих столов.

2.13 Оконный менеджер «WindowMaker»

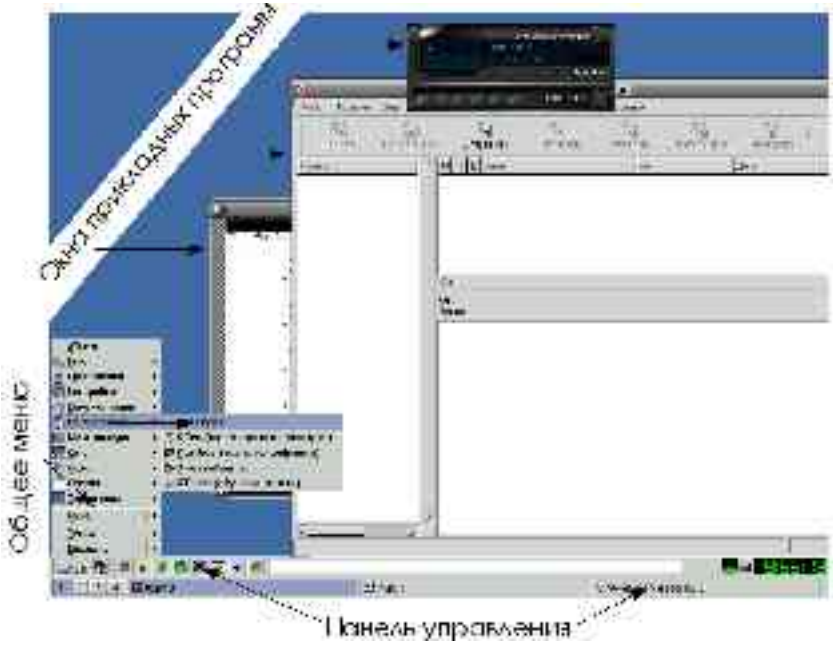

Рис. 2-9

«WindowMaker» («WM») — это свободная реализация (в рамках проекта «GNUStep») концепций «NextSTEP» — первой получившей более или менее широкую известность универсальной графической среды пользователя. За недоступностью оригинальной «NextSTEP» для современных платформ, познакомиться с «WM» полезно и поучительно вне зависимости от того, собираетесь ли вы с ним работать — это позволит увидеть исходную точку развития графических сред и оценить продуктивность (или контрпродуктивность) того, чем эти идеи «обросли» со временем.

Основным автономным виджетом WM, как и NextSTEP, является «пирс» прикладных программ, представленный при запуске пиктограммой со скрепкой. При запуске любой «корректной» (с точки зрения WM), а также некоторых «некорректных» программ, кроме ее окна на экране появляется ее пиктограмма. Если «пришвартовать» эту пиктограмму к «пирсу», она там и останется, позволяя запускать эту программу вновь и вновь простым щелчком по ней — это «родной» некстстеповский интегрирующий интерфейс.

WM позволяет работать с несколькими «столами» (переключение по умолчанию по Alt-n или через меню, доступное по «щелчку» правой

кнопкой на свободном месте «стола»). «WM» очень гибко настраивается, как в части внешнего вида, так и в части «поведения», причем большая часть настроек доступна из программы «Wprefs.app», запускаемой по щелчку на пиктограмке «со ступенькой».

Рис. 2-10

2.14 Оконный менеджер «IceWM»

«IceWM» — простой оконный менеджер, очень часто выбираемый пользователями, приходящими из-под «Майкрософт Уиндоуз» или «OC/2», поскольку он способен достаточно точно имитировать их основные черты.

Из автономных виджетов прежде всего стоит отметить панель с кнопкой, вызывающей главное меню (подобно тому, как это делает кнопка в «Майкрософт Уиндоуз», «Гном» или «КДЕ»). С помощью панели можно также управлять текущим сеансом и настраивать «IceWM». Впрочем, основное меню также доступно и по «щелчку» правой кнопкой на свободном месте «стола», что более привычно для пользователей «WindowMaker», «Sawfish», «Blackbox» или «Enlightenment».

А еще панель содержит список запущенных программ (включая те, окна которых минимизированы), на нее можно вывести и «мини-терминал», позволяющий оперировать командной строкой. Любые действия могут выполняться с помощью ассоциированных клавиатурных комбинаций.

«IceWM» также позволяет работать с множеством «столов» («рабочих мест»), которые нумеруются или именуются пользователем.

2.15 Интегрированные графические среды

Запуск графической среды (точнее, «бутерброда» из оконной среды «Икс», оконного менеджера и графической среды) в открытой операционной системе можно сравнить с запуском «Майкрософт Уиндоуз» в «МС-ДОС»[65].

Однако, сходство заканчивается, не успев начаться. «МС-ДОС» — это однозадачная и однопользовательская система, и запущенная оболочка захватывает все ее ресурсы. Из-за неполноценности ОС оболочке приходится брать на себя несвойственные ей функции (например, имитацию многозадачности), с которой она справляется плохо (так, «зависание» одной программы вполне может привести к неработоспособности всей системы).

При запуске графической среды под полноценной ОС, она, с точки зрения последней, представляет группу обычных процессов, управление которыми производится общесистемными средствами. Точно так же, общесистемными средствами производится и управление процессами, запускаемыми «из-под» этой графической среды. Более того, поскольку платформой для запуска конкретной среды является изначально сетевая среда «Икс», прикладная программа даже может запускаться на другом компьютере.

Среда отнюдь не монополизирует использование компьютера; параллельно с ее работой продолжает исполняться множество служебных системных процессов; с других терминалов (если система многотерминальная) могут запускаться другие программы или даже другие графические среды (или дополнительные экземпляры той же среды).

2.16 Плюсы и минусы интегрированных сред

Однородность опыта и связанная с нею привычность (иногда ошибочно называемая «интуитивностью», хотя она не имеет отношения к философ-

[65] Имеются в виду графические оболочки («Майкрософт Уиндоуз» 1.x, 2.x, 3x, 9x и Me), а не семейство «Майкрософт Уиндоуз НТ» (сегодняшними версиями которых являются «Майкрософт Уиндоуз 2000, XP и .NET Server»).

скому и психологическому понятиям интуиции) позволяют при освоении нового инструмента-программы сосредоточиться на ее прикладной логике (на аспектах, связанных с конкретным приложением ИТ, которое она реализует), не задумываясь и специально не фокусируя внимания на аспектах, общих для разного инструментария. Это делает более «крутой» пресловутую «кривую обучения» нового пользователя.

(Разумеется, это сильно идеализированная картина. Иногда прикладная логика диктует некоторые элементы эргономики; например, интерфейсы большинства систем автоматизированного конструирования и проектирования (CAD, САПР) весьма сходны, вне зависимости от среды, в которой работают эти программы.)

Как ни парадоксально, основной недостаток работы в интегрированной среде является оборотной стороной основного достоинства: жестко закрепленные навыки мешают при выходе за ее пределы. Конечному пользователю, ограниченному опытом работы в одной среде, недостает «стереоскопичности» видения, глубины понимания; элементы эргономической логики могут напрямую ассоциироваться с определенными визуальными элементами и «жестами», с помощью которых подаются команды.

Общеизвестны сложности, с которыми сталкиваются люди, несколько лет пользовавшиеся «МакОС» или «Майкрософт Уиндоуз» при необходимости поработать в другой (пусть даже и весьма схожей) среде. Подобная «ригидность» опыта может формироваться и при работе в любой из свободных сред, хотя как правило пользователь в них не ограничен (в отличие от специфически персонально-компьютерных сред) прикладными программами, специально разработанными для данной среды и делящими с ней наборы элементов интерфейса («виджетсеты»), поэтому его опыт изначально более разнообразен.

В этом смысле, крайне полезным представляется знакомство учащихся с *разными* средами уже на начальном этапе освоения графических интерфейсов. Это не обязательно должны быть разные интегрированные среды, но само представление о том, что один и тот же результат может достигаться с помощью разных интерфейсных средств, должно быть передано обязательно. В общем случае это возможно и в рамках одной интегрированной среды из числа рассматриваемых ниже — и «КДЕ», и «Гном» в высшей степени гибки в отношении настройки внешнего вида и «поведения». Эта гибкость также весьма полезна для обеспечения доступности и максимально достижимого комфорта учащимся с физиологическими особенностями и физическими недостатками (дальтонизмом, слабым зрением, ограниченной подвижностью или расстроенной координацией движений и т.п.).

2.17 Общие черты интегрированных сред

Сколько-нибудь последовательной теории интегрированных графических сред не существует. Изучая отдельные среды в динамике их развития, можно, тем не менее, выделить несколько общих черт:

- они опираются на определенный интерфейс разработчика (API), состоящий из библиотек, доступных также разработчикам прикладных программ (будь то MS Windows API для «Майкрософт Уиндоуз», «Мотиф» для «CDE», «Qt» для «КДЕ» или GTK+ для «Гном»);
- они реализуют элементы объектной метафоры (одельные экземпляры сущностей, таких как файлы, процессы (их потоки ввода-вывода) визуализируются определенным образом, на них можно фокусироваться и выполнять с ними различные действия, их состояния и изменения этих состояний также могут визуализироваться или озвучиваться). Целостная объектная метафора своей реализации не нашла (и, видимо, последовательно «объектная» среда была бы крайне неудобной в использовании);
- они реализуют единообразные элементы управления («виджеты»), зачастую не только в оформлении отдельных окон, но и в их «начинке»;
- они содержат те или иные элементы управления, не привязанные к отдельным окнам прикладных программ («общие» меню, «панели управления», «поверхность стола» и т.п.);
- они позволяют согласованно изменять свойства интерфейса образующих среду программ;
- они реализуют «буфер (буферы) обмена», позволяющий передавать типизованные данные от программы программе (оконная

система «Икс» содержит буфер, позволяющий передавать данные лишь простого текстового типа);
- они реализуют возможность «перетаскивания» (drag'n'drop) объектов или данных между окнами одной программы или разных программ.

(За ограниченностью объема «за бортом» остаются более сложные вопросы, такие, как компонентная объектная модель и модели сетевого взаимодействия, так или иначе «втягиваемые» в проекты интегрированных сред.)

На сегодня существуют и развиваются две свободные интегрированные графические среды общего назначения: «КДЕ» и «Гном». Они входят в поставку большинства стандартных (открытых) ОС, как свободных, так и несвободных. Хотя «Гном» на полгода моложе «КДЕ», мы начнем обсуждение именно с «Гном».

2.18 «Гном» (Модельная среда сетевых объектов GNU)

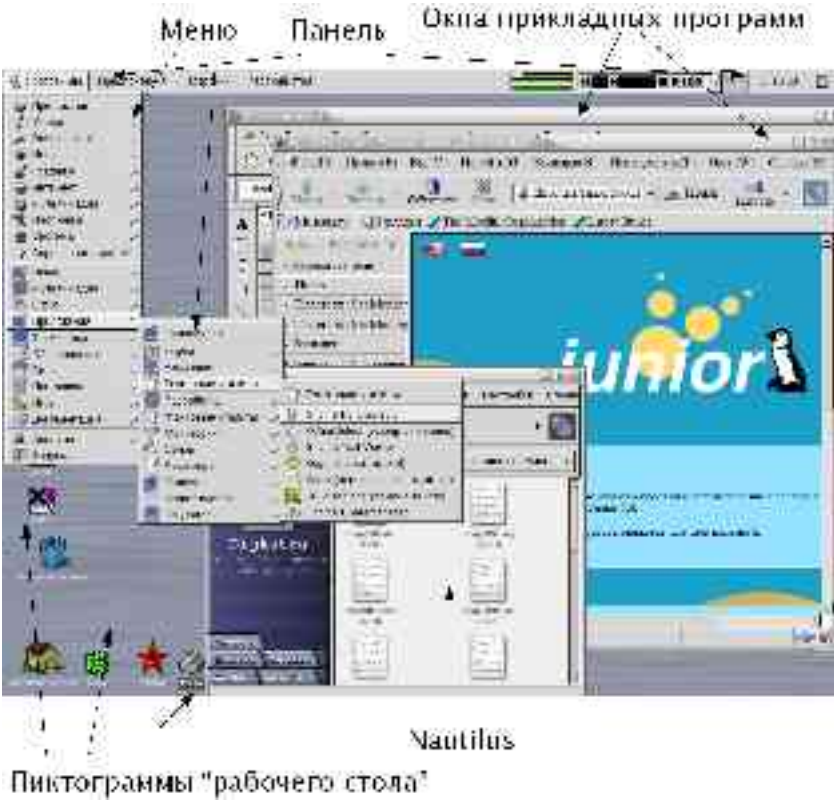

Рис. 2-11

«Гном» (GNOME, GNU Network Object Model Environment — «Среда GNU, основанная на модели сетевых объектов», но также и «Образцовая среда для сетевых объектов GNU») — один из самых амбициозных и масштабных проектов в программистском сообществе.

Кроме реализации функционально полной графической среды (возможно, уместнее говорить о сенсуальных средах, учитывая то, что звук стал их полноправной частью), «Гном» претендует на то, чтобы полностью реализовать спецификации промышленной платформы сетевого взаимодействия CORBA и полностью абстрагировать слой «менеджера рабочего стола» (или «графической среды») от низлежащего слоя управления окнами («оконного менеджера»).

«Гном» поддерживает ряд оконных менеджеров, среди которых: Sawfish («штатный» оконный менеджер по умолчанию), Enlightenment, IceWM,

WindowMaker, AfterStep и FVWM2, совместимые с «Гном», впрочем, в разной степени.

Сегодняшняя версия «Гнома» («Гном» 2.4) — полноценная интегрированная среда, включающая реализацию повседневно необходимых функций и позволяющая использовать сторонние решения для реализации функциональности, которая в ней отсутствует.

«Гном» использует один из самых развитых интерфейсных пакетов GTK+, реализованный для разных платформ. Над ним надстраивается масса компонентов и библиотек, обеспечивающих сетевую функциональность, интерфейсы к различным языкам программирования, работу со звуком через механизмы ОС и пр. Сам «Гном» стремится оставаться мобильным и доступным во всех открытых системах. Он стабильно работает в «ГНУЛинукс», «БСД», AIX и «Солярис»; последнее обстоятельство способствовало поддержке разработки «Гном», которую оказывает Sun Microsystems через созданный в 2001 г. году «Фонд “Гном”», среди учредителей которого также крупнейшие поставщики свободных ОС.

С пользовательской точки зрения «Гном» предстает как набор базовых компонентов интерфейса и «апплетов», утилит и прикладных программ. К базовым компонентам относятся менеджер файлов и поверхности стола Nautilus, панели управления и меню GNOME Panel и центр управления (Gnome Control Center).

«Наутилус». Менеджер файлов Nautilus позволяет отображать содержимое файлов и каталогов в окнах и выполнять над файлами обычные действия (удаление, переименование, копирование и перемещение и т.п.), а также осуществлять предварительный просмотр многих типов данных. «Наутилус» эффектен, но работа с ним не более эффективна, чем с прочими браузерами файлов, включаемыми обычно в графические среды (менеджер файлов CDE или «Майкрософт Уиндоуз Эксплорер»).

Помимо отображения содержимого каталогов в окнах, «Нау» также может отображать один из каталогов на поверхности «рабочего стола»: размещенные на нем иконки как бы приклеены к монитору, и при смене текущего экрана остаются на том же месте относительно наблюдателя (так же, кстати, ведут себя и открытые окна, если их «приклеить»).

Поддерживается широкий спектр операций переноса мышью (drag'n'drop), причем «перетаскиванию» подвержены не только объекты (файлы, пункты меню и т.п.), но и некоторые их свойства: так, можно «взять цвет» в окне выбора цвета и перенести его на панель, которая воспримет его. Есть даже операции, позволяющие назначить один объект свойством другого: например, если на панель «перетащить» не цвет, а

файл с картинкой, последняя станет ее фоном. «Таскать» файлы между окнами «Нау», на рабочий стол и панели можно практически без ограничений.

Панели и меню. Уже упомянутые панели являются, наряду с менеджером файлов, важнейшей составной частью интерфейса «Гном». Панелей может быть неограниченное количество. Панель может относиться к одному из пяти типов, но на самом деле их два: панель-меню (menu panel) и объектная панель. Первая из них содержит пункты меню и может содержать пиктограммы, а вторая — только пиктограммы. Последняя может быть краевой (edge), выравненной (aligned), скользящей (sliding) или плавающей (floating), но это скорее свойство панели (которое можно менять «на ходу»), определяющее особенности ее поведения, чем тип.

Внешний вид и поведение панелей является в высшей степени конфигурируемым. Пользователь может задавать как глобальные предпочтения (анимация движения панелей, отображение панельных объектов и пр.), так и индивидуальные предпочтения для каждой из них (ее тип и положение на экране, ширина, возможность автосокрытия и принудительной минимизации, цвет и фоновое изображение и т.п.) Ну и, разумеется, пользователь может «набивать» панели теми объектами, которые ему нужны.

На панелях могут присутствовать объекты пяти типов:

- пускатель (launcher) ассоциирован с приложением или командой, которые исполняются по щелчку на его пиктограмме в панели;
- выдвижной ящик (drawer) — это кнопка, открывающая другую панель, перпендикулярно первой — некий аналог подменю в меню, который можно «набить» всевозможными апплетами;
- апплет (applet, «приложеньице») — интересный тип панельного объекта, демонстрирующий то, что он не обязан быть представлен статической картинкой. Это программа, места в панели которой достаточно, чтобы отображать какую-нибудь полезную (или забавную) информацию или даже принимать клавиатурный и/или координатный ввод. С «Гном» поставляется масса апплетов, отображающих всякую полезную информацию (состояние ресурсов и статус сети, например) или позволяющих осуществить нетривиальные действия (например, mini commander, позволяющее набрать команду, не запуская терминала). Важными «приложеньицами» являются «путеводитель по столу» (Desktop Guide) и «список задач» (Task List), позволяющие переключаться между виртуальными экранами и активизировать окна запущенных программ, соответственно;

- специальные объекты — это те же апплеты, но выполняющие функции, которые другими средствами «достать» почему-либо нельзя (запереть экран, выйти из «Гном» или запустить программу «вручную»). В качестве «специальных объектов» исполняются и программы, которые не были написаны специально для «Гном», но могут, тем не менее, осуществлять вывод в панель — «поглощенные программы» (swallowed applications);
- наконец, объект-меню раскрывает меню.

За работу системы меню, как и за работу панелей, отвечает компонент GNOME Panel, и это не случайно: разница между панелью и меню более декоративная, чем сущностная: любое меню можно зафиксировать на экране, и оно превратится в подобие панели-меню (только вертикальное, а не горизонтальное, и с меньшими возможностями настройки).

У «Гнома» нет единой иерархии меню: кроме главного, вызывающегося объектом-меню с гномьей лапой (оно же, когда вызывается щелчком правой кнопки на фоне или нажатием клавиши, почему-то называется глобальным (global)), пользователь может создавать «обычные» (normal) меню, связанные с объектами-меню на панелях.

Меню настраиваются примерно так же, как и панели: пользователь может добавлять, менять и удалять пункты, создавать подменю и т.п. При этом создаваемые «обычные» меню изначально пусты, а главное/глобальное «набивается» при установке всем, что «Гном» найдет в системе, и пользователю остается только убрать лишнее и переставить пункты в соответствии со своими предпочтениями.

Утилиты, апплеты и «капплеты». Для настройки различных аспектов функционирования системы предназначен Центр управления, представляющий собой набор «управляющих апплетов» (capplets), связанных с разными компонентами и прикладными программами.

Одни из них позволяют менять параметры рабочего стола и облик приложений (включая использование «тем»), другие — настраивать мультимедиа, третьи — управлять свойствами клавиатуры и мыши, и т.д.

Важным «капплетом» является менеджер т.н. «драйверов документов» (Document Handlers), устанавливающий соответствие между типом файла или протокола и программой, выполняющей различные операции с ними. Набор «капплетов» является расширяемым, их можно писать не только для программ, написанных специально для «Гнома», но и для внешних программ.

Еще более открыт набор утилит, прикладных программ и апплетов, поставляемых с «Гном» — вместе с программами, входящими в

большинство дистрибутивов ОС, о которых «Гном» «в курсе», их число превышает сотню.

Перечислить их здесь нет никакой возможности, но среди них есть интерфейсы для администрирования системы, средства звукозаписи и воспроизведения, сетевые утилиты, игры и многое другое.

«Гном» снабжен встроенной системой помощи; кроме того, его разработчиками совместно с Sun Microsystems подготовлено компактное (130 с.) руководство, доступное в разных форматах на сайте проекта. В его поставку входит система разработки графических приложений под GTK+, которая называется Glade и включает в себя специфические для «Гном» элементы.

«Гном» и большинство его компонент соответствуют соглашениям об интернационализации и, соответственно, поддерживают работу с кириллицей и локализацию и перевод интерфейса. К сожалению, локализация «Гнома» сильно отстает от разработки.

2.19 «КДЕ» (Настольная среда K)

Само название «КДЕ» — явная пародия на CDE — Common Desktop Environment («Общая настольная среда») — последнюю попытку отрасли стандартизовать графическую среду на несвободной основе, предпринятую в конце девяностых годов. «K» в «КДЕ» ничего не означает.

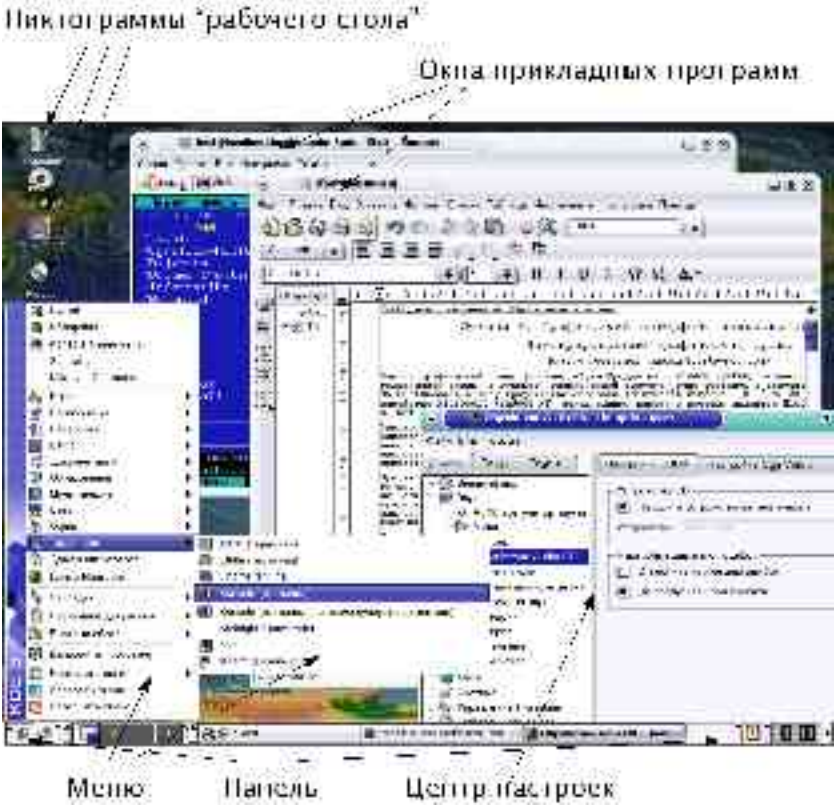

Рис. 2-12

Несмотря на явно игривый тон, начинающийся с названия среды и продолжающийся в названии компонент (в «КДЕ» любят играть со словами; например, универсальный браузер, входящий в среду, называется Konqueror (от Conqueror — «завоеватель», «покоритель»), терминал — Konsole (от Console — «консоль), а система помощи — вообще Kandalf (от имени Гэндальфа, мага из фантазийных произведений Дж.Р.Р.Толкиена)), «КДЕ» — очень серьезный проект.

Если единообразие и «сплошность» среды считать достоинством, то «КДЕ» — несомненный лидер среди всех (как свободных, так и несвободных) интегрированных графических сред. Основное «видимое» средство интеграции — это универсальный браузер Konqueror. Функция Konqueror близка к той, которую приобрел «Майкрософт Интернет Эксплорер» в «Майкрософт Уиндоуз» — он совмещает функции гипермедийного браузера WWW и браузера локальных ресурсов.

Разработчики «КДЕ» пошли даже дальше своих коллег из Microsoft и определили ряд дополнительных протоколов, что позволило, в частности, просматривать с помощью браузера в единообразном формате все разнообразие справочной информации, представленное в сегодняшних открытых система (традиционные страницы руководства man, гипертекстовую систему Info из проекта GNU, разрозненные файлы документации в текстовом и гипертекстовом формате). В Konqueror интегрирована также возможность предварительного просмотра содержимого большого количества типов файлов.

«КДЕ» включает также настраиваемую систему панелей и меню и интегрированный «центр управления», позволяющий согласованно изменять параметры среды. «КДЕ» менее гибка в настройке, чем «Гном», однако ее гибкости вполне достаточно для решения любых практических задач (в том числе, имитации вида и поведения других сред). «КДЕ» работает только с собственным оконным менеджером KWin.

В поставку «КДЕ» входит множество «аксессуаров» и прикладных программ, к тому же «рядом» с проектом выросла целая «грибница» сопутствующих, ориентированных на те или иные предметные приложения, из которых самым развитым является KOffice.

Глава 3. Пакет «Мозилла»

Среди массы свободных клиентских программ, связанных с сетевой функциональностью, за ограниченностью печатного объема остановимся на проекте «Мозилла».

«Мозилла» представляет собой свободный пользовательский прикладной пакет, реализующий интерфейсы просмотра WWW (браузер), электронной почты и новостей USENET, многопользовательских диалогов в реальном времени («чатов») IRC и редактирования страниц WWW (компоновщик). Входящие в пакет программы, таким образом, открывают доступ ко всем наиболее популярным приложениям Интернет (кроме, на сегодня, «быстрых сообщений»). Существенной сильной стороной «Мозилла» является практически неизменная функциональность и эргономика на широком спектре платформ, включая как стандартные («МакОС X», «ГНУ/Линукс», «Солярис», «Айрикс», HPUX и пр.), так и альтернативные («МакОС 9», «Майкрософт Уиндоуз», «БиОС», «ОС/2») операционные системы на большинстве аппаратных платформ.

Слегка модифицированная версия «Мозилла» 1.x также распространяется организатором разработки — компанией AOL Time Warner.— под названием Netscape Navigator 6.x и 7.x.

Русская локализация «Мозилла» (www.mozilla.ru) выполнена Валентиной Ванеевой, Вадимом Виниченко и Сергеем Дегтярёвым. Текущая стабильная версия — 1.5.

Помимо текущей ветви разработки, команда также работает над следующим поколением приложений «Мозилла». Уже доступны предварительные версии браузера «Жар-Птица» («Mozilla Firebird») и пакета работы с электронной почтой и новостями «Гром-птица» («Mozilla Thunderbird»), они также оперативно русифицируются.

3.1 Базовая функциональность «Мозилла»[66]

Браузер. Для просмотра страниц WWW и «хождения» по FTP-серверам предназначен компонент «Навигатор». На панели компонент «Мозилла» он изображается морским штурвалом (см. рис. 3-1).

[66] Раздел написан Андреем Добровольским <dobr@iop.kiev.ua>. Текст ранее публиковался в [19] и в [21].

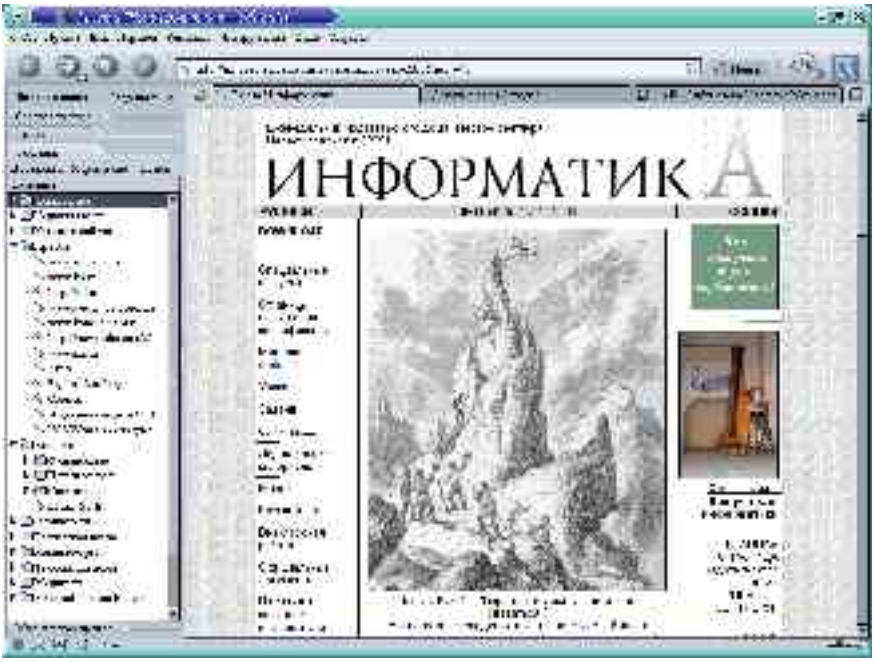

Рис. 3-1

Все возможности программы доступны из меню, наиболее часто употребляемым соответствуют также «горячие клавиши» и кнопки панели навигации. Под меню — еще одна панель, которую можно настроить по своему желанию (чаще всего, на нее выносятся ссылки на десяток самых посещаемых страниц), еще ниже — окно для отображения гипертекстовой информации с посещаемых страниц. Замыкает окно строка состояния с панелью компонент слева и служебными пиктограммами справа.

С каким бы компонентом вы не работали в даный момент, всегда можно запустить любой другой кнопкой с соответствующим значком из панели компонент или пункт меню «Окно». Из пункта меню «Создать...» вы можете, кроме того, открыть еще одно окно просмотра или вкладку (вкладка — это окно в окне) уже открытого окна, на разных вкладках вы можете просматривать разные страницы с одного сайта или разные сайты. Это позволяет не захламлять экран и панель задач окнами одного типа и сразу видеть, где закачка страницы уже завершилась, а где еще нет.

Из пункта меню «Файл» можно также открыть любой локальный html-файл или каталог (или файл другого известного Навигатору формата, например, графический или текстовый), одной командой переслать по почте адрес понравившейся страницы или даже всю страницу, сохранить

страницу на жестком диске или напечатать на принтере.

Программа ведет учет страниц, которые вы посетили («Перейти... Журнал посещений»). Это удобно для быстрого возврата на уже посещенную страницу. Сколько просмотренных адресов (и на какое время) программа будет их сохранять, можно указать в общем окне настроек.

Как и в других браузерах, вы можете вести структурированную базу закладок на понравившиеся сайты и редактировать параметры каждой закладки, в частности, менять названия и делать пометки на память: а чем же зацепила вас именно эта страница.

Навигатор может передать запрос на поиск информации на указанные поисковые машины, дает возможности управления заполнением форм и блокирования изображений (это удобно для отсечения рекламы из многочисленных баннерных сетей) и cookies конкретных сайтов, запрета появления выскакивающих окон.

Наверное, самая интересная особенность программы — «боковая панель» (Sidebar). Размещенная в левой части всех окон, она значительно облегчает работу. На нее можно вывести множество полезной информации. Например, вы можете настроить ее так, чтоб сразу при запуске программы, видеть свежие новости, новости с любимых новостных сайтов, все свои закладки в виде дерева, иметь доступ к своим любимым поисковикам или еще что-нибудь, что сами придумаете.

Почта. Электронная почта соревнуется с WWW за почетное звание главного коммуникационного приложения. Существует масса свободных программ для работы с почтой, и среди них — модуль Мозиллы «Почта и новости».

Открыть его можно, нажав кнопочку с изображением конверта, выбрав соответствующий пункт меню «Окна».

Этот модуль позволяет обмениваться письмами с друзьями, писать и получать почту из почтовых рассылок и групп новостей. Всю свою почту можно разложить по папочкам и настроить систему фильтров, чтобы новая почта попадала, куда нужно, а надоедливый спам — в мусор.

Вы можете отвечать на письма, переадресовывать полученное кому-то еще, отсылать и получать нужные файлы. Когда нужно быть уверенным в личности отправителя и неприкосновенности письма, воспользуйтесь интеграцией с GPG, свободной реализацией популярного формата OpenPGP, обеспечивающих электронную подпись и шифрование сообщений.

В почтовом ящике возможен поиск по теме, отправителю, словам в

самом письме, дате, приоритету, адресатам, датам или комбинации этих параметров — этого хватит, чтоб не потеряться в своем почтовом ящике. Можно иметь несколько почтовых ящиков на разных серверах и работать с ними в одном окне.

Главное окно программы поделено на четыре части. Слева вверху — список почтовых ящиков, под ним боковая панель. Справа вверху — список писем в выбранной папке, под ним — текст выбранного сообщения. Многие из них можно временно спрятать, чтоб освободить место для других окошек. И здесь доступна боковая панель со всеми ее возможностями.

Выбрав в правой части название почтового ящика, вы увидите список действий, которые можете выполнить в нем и над ним. Например, настроить сам почтовый ящик, если есть необходимость, и параметры работы с почтовым сервером, параметры для исходящих и входящих сообщений. Возможности гибкой настройки и мощная система фильтрации сообщений, вместе с возможностями проверки правописания для многих языков (включая английский, русский, украинский) сделают вашу переписку удобной и приятной.

«Компоновка» собственных страниц. Немного побродив по Сети, каждый испытывает желание сообщить о чем-то своем всему мировому сообществу. Препятствием к этому оказывается необходимость изучать стандарт HTML. Не очень-то просто найти для этого время, особенно если речь идет о создании странички на пару абзацев о домашнем любимце.

В этой ситуации вам на помощь придет «Компоновщик», спрятавшийся за кнопкой с листком и пером для письма. Вызвав его, вы сможете быстро и без погружения в изучение стандарта создать простую страницу в режиме визуального редактирования. Как в word-процессоре, просто пишете текст, вставляете таблицы и рисунки и сразу можете видеть, что у вас получается. Более того, заглянув на другие вкладки окна Компоновщика, можно еще и подучить HTML. В этом помогут вкладки «Все теги» и «<HTML> Код». Проверить, как ваша работа будет смотреться в окне просмотра «Навигатор», можно на последней вкладке. Главное преимущество этого инструмента перед многими похожими состоит в создании правильного HTML кода без лишних и нестандартных тегов, поэтому рекомендую его как первый инструмент начинающего (разумеется, для создания серьезных проектов изучение стандартов W3C обязательно).

В окне программы сперва линейка меню, под ней инструментальные панели, далее главное рабочее окно с вкладками и панель состояния под ним. Для подробного описания всех возможностей модуля потребуется не одна страница. Просто запустите его и попробуйте что-нибудь со-

здать. Большинство пиктограмм и команд никакого объяснения не требуют.

Адресная книга. По мере того, как увеличивается круг ваших знакомых, помнить все необходимые адреса становится все тяжелее, но на помощь приходит адресная книга программы. Каждый раз, когда вы пишете письмо новому адресату или получаете письмо от нового отправителя, этот компонент программы запоминает его адрес. Дальше, когда вы начинаете заполнять поле «кому» нового письма, он пытается завершить набор за вас. Если адрес угадан правильно, можно просто переходить к заполнению остальных полей. Когда программа ошиблась, можно выбрать подходящий адрес из выпадающего списка или завершить набор самостоятельно.

Внешний вид окна «Адресной книги» достаточно типичен: снова, сперва меню, затем панель инструментов и основное окно с панелью состояния. Основное окно поделено на четыре части, некоторые из которых можно прятать. Слева — названия коллекций адресов вверху и боковая панель под ними. Справа — список карточек выбранной коллекции вместе с панелью поиска по ним вверху и окно для отображения информации с выбранной карточки внизу. Записи для каждой карточки редактируются по необходимости.

Настройки. Мозилла — пакет не только мощный, но и очень гибкий в настройке. Все доступные параметры настроек собраны в одном окне — дерево доступных настроек поделено на категории в соответствии с имеющимися программами плюс несколько общих.

В общей категории «Внешний вид» настраивается использование шрифтов для каждой кодовой страницы, отображаемые при запуске Мозиллы окна, использование цветов, язык интерфейса.

Вторая общая категория — «Дополнительно» — позволяет отрегулировать использование cookies, Java, размер и время сохранения буфера на диске для быстрого возврата к уже просмотренным страницам, использование прокси-серверов и еще некоторые полезные параметры.

Название группы «Приватность и безопасность» говорит само за себя. Количество доступных в ней настроек, говорит о том, насколько ответственно разработчики отнеслись к этому вопросу. Предлагаю вам разобраться с ней самостоятельно.

Регулировать внешний вид окна программы и степень «свежести» страницы в окне просмотра, вам позволят пункты меню «Вид». Отсюда можно запретить или разрешить показ каждого из компонентов окна программы, подобрать размер шрифта на странице, изменить, если нужно,

кодовую страницу для показа текста, увидеть исходный код страницы и получить информацию про страницу на которой сейчас находитесь.

3.2 «Мозилла»: как это сделано

Возможно, более интересным будет общий взгляд на «начинку» «Мозилла», поскольку эта тема остается пока практически нераскрытой в публикациях на русском. А именно интересными внутренними архитектурными решениями и возможностями, которые они открывают для программистов, обусловлена перспективность и востребованность этого проекта.

«Движок» «Геккон». Собственно интерпретацию («рендеринг») гипертекста и гипермедиа в браузере и компоновщике осуществляет «движок» под названием Gecko («геккон», «ящерка»). Сам по себе «движок» компактен и быстр. Он используется также рядом альтернативных браузеров, таких, как «Галеон», «Афродита», «Камино», «Феникс»; первый из них, достачно развитый и «шустрый», может рекомендоваться пользователям старых или маломощных (например, карманных) машин.

Язык XUL. Видимо, самой интересной особенностью «Мозилла» является реализация интерфейса пользователя на языке XUL, представляющем собой XML-приложение, т.е. набор определений вида и поведения визуальных объектов, свойственных современным графическим интерфейсам пользователя, на расширяемом языке разметки XML (об XML и его интегрирующей роли в современных приложениях компьютера подробнее говорится в разделе 4.1).

В большей своей части программы «Мозилла» «написаны для XUL» и интерпретируются «на лету». Обратной и неприятной для пользователя стороной этого остается значительная ресурсоемкость «Мозилла» и размер пакета. Базовая установка занимает порядка 30 Мб и комфортная работа в большинстве сред начинается от 128 Мб ОЗУ и 500 МГц процессора.

«Светлой» стороной XUL'ности «Мозилла» является его чрезвычайная гибкость, что, видимо, позволит уже в ближайшие годы «малой кровью» существенно расширить набор входящих в пакет программ и практически полностью покрыть ими клиентские приложения Интернет.

На Рис. 3-2 видно, что по специальному протоколу chrome можно просто «открыть» в браузере еще один браузер (или любой другой компонент пакета), причем это не просто изображение его интерфейса, а полноценная работающая программа.

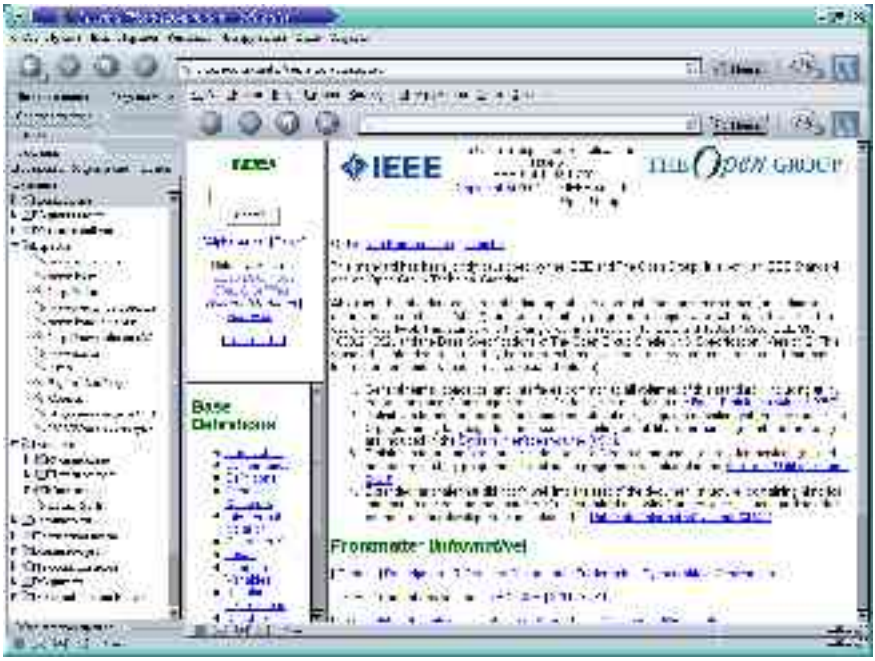

Рис. 3-2

Разумеется, суть использования XUL не в этом, а в том, что на этом относительно простом языке разметки могут быть легко реализованы различные интерактивные прикладные программы. На сайтах www.mozdev.org и www.xulplanet.com можно найти десятки программ на XUL, а по http://books.mozdev.org/chapters/ можно найти текст книги «Создание прикладных программ в “Мозилла”» (к сожалению, пока только на английском).

Например, так выглядит традиционная первая программа «Привет, мир!» на XUL (ее текст заимствован из упомянутой книги) — см. Рис. 3.3.

```
<?xml version="1.0"?>
<!-- Sample XUL file -->
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<box align="center">
 <button label="hello xFly" onclick="alert('Hello World');" />
</box>
</window>
```

Рис. 3-3

Знакомым с JavaScript-сценариями в HTML или XML этот пример должен быть кристально понятным. Однако внимание следует обратить на то, что элементы разметки, определяемые XUL, свойственны графиче-

ским интерфейсам общего назначения («окно» (window), «контейнер» (box), «кнопка» (button) и т.п.), и весь XUL-файл определяет интерфейс, а не страницу[67].

Перспективы, открываемые универсальным языком описания графического интерфейса перед отраслью быстрой прикладной разработки (RAD) и практическим программированием вообще — очевидны. Но здесь уместнее будет указать на методические перспективы: включение в один и тот же вводный курс программирования на XUL наряду с изучением языков разметки контента (например, HTML, особенно в его нынешней, XML-версии) позволит осуществить значительную понятийную экономию. Возможно, наш кругозор ограничен, но нам не известны другие полноценные языки программирования интерфейса, кроме XUL, являющиеся корректными и исполнимыми XML-приложениями.

[67] Строго говоря, «Мозилла» поддерживает и смешанные контексты, например, XUL-компоненты в HTML или наоборот, но XUL-интерпретатор как таковой не обязан этого делать, и XUL-интерпретатором, соответственно, вовсе не обязательно должен быть браузер.

Глава 4. «Открытый Офис»

Хотя «офисные» программы с распространением сетей и коммуникационных программ и перестали быть основным приложением персональных компьютеров, они все еще достаточно популярны, и в учебных планах им отводится значительное (быть может, даже слишком большое) место. К «офисным» программам традиционно относят словарные процессоры, редакторы электронных таблиц и компоновщики мультимедийных презентаций.

В пакет «OpenOffice.org» (далее — «OO.o»), которому посвящена эта глава, входят:

- словарный процессор «OpenWriter»;
- редактор электронных таблиц «OpenCalc»;
- компоновщик мультимедийных презентаций «OpenImpress» (в этой книге не рассматривается).

Кроме того, в пакет включен:

- векторный графический редактор «OpenDraw».

У «OpenOffice.org» есть масса недостатков — он (версия 1.0) вышел сыроватым (хотя версия 1.1, уже выпущенная в англоязычном и готовящаяся к выпуску в русскоязычном варианте, гораздо более «вылизана»), он достаточно «тяжел» для слабой техники, он неидеально отдокументирован и локализован) — которые постепенно исправляются. У него есть четыре очень серьезных достоинства:

1) он свободен, поставляется конкурентно (с соответствующими ценовыми последствиями) и может быть использован как база для дальнейших разработок (в том числе, специфически учебных);

2) он изначально платформенно-независим, и пользование им не диктует практически никаких ограничений на выбор операционной системы и операционной среды. Более того, навыки работы с ним также в большой степени переносимы. Фактически, «OO.o» выглядит и управляется одинаково под любой стандартной ОС (разработчики говорят о поддержке «ГНУ/Линукс» и «Солярис»; известно об устойчивой работе пакета под FreeBSD, «Дарвин»/«МакОС X»), а также под «Майкрософт Уиндоуз 9x/Ми» и «Майкрософт Уиндоуз NT/00/XP»;

3) он использует в качестве языка разметки XML (не только для размеченного текста, но также и для математических формул, т.е. по сути, все его форматы суть приложения международного стандарта разметки тек-

ста SGML), соответствующие определения типов документа доступны и отдокументированы. В качестве кодировки используется Unicode, что позволяет забыть про третью (после классических «дураков и дорог») российскую проблему с разнобоем в кодовых таблицах, используемых для представления кириллицы в разных системах,

4) в команде разработчиков есть российские программисты и фирмы, которым небезразличны перспективы его применения в школе.

4.1 Словарный процессор «OpenWriter»

И «Обязательный минимум...», и большинство конкретных учебных планов предусматривает знакомство лишь с базовой функциональностью программ манипуляции текстами, и любой из свободных словарных процессоров («AbiWord», «Kword», «OpenWriter») гарантированно и с избытком перекроет требования (так же, как и масса присутствующих на рынке несвободных программ, таких, как «Майкрософт Уорд», «Лексикон», «СтарОфис» или «УордПёрфект»). Поскольку большинство из них так или иначе развивают основные интерфейсные подходы «УордПёрфект», существенного (и могущего быть выявленным в пределах тех немногих часов, что учитель в состоянии уделить этой теме) эргономического различия между ними нет.

Значит основания для рационального выбора программы нужно искать не внутри темы манипулирования текстами, а в ее связи с другими учебными темами. У программистов есть такой эмпирический принцип: когда начинаешь путаться в программах, отложи их в сторону и попытайся разобраться со структурами данных — сам удивишься, насколько очевидными и простыми окажутся после этого решения, касающиеся программ. Возможно, этот принцип разумно применять и пользователям.

Давайте попробуем отложить в сторону программный инструментарий и разобраться с тем, какого рода данные мы им обрабатываем.

«Плоский» и размеченный

Водораздел между текстовыми редакторами и word-процессорами проходит по критерию способа отображения размеченного (имеющего некоторые атрибуты, такие, как цвет, начертание и кегль (размер) символов, выключка (выравнивание) и расположение абзацев, оформление страницы и т.п.) текста. Текстовый редактор отображает его как есть, например:

<курсив>Предложение, набранное курсивом.</курсив>

а word-процессор визуализирует эти атрибуты, например:

Предложение, набранное курсивом.

Визуализацию иногда путают с так называемым WYSIWYG-принципом (сокращение от «What you see is what you get» — «Что видишь, то и получишь»).

Однако в общем случае это неверно: WYSIWYG-идеология унаследована от эпохи персональных компьютеров, когда в ходе «малой компьютеризации» отдельные офисы и рабочие места становились «островами» безбумажных технологий в море бумажной коммуникации, и компьютерное представление мыслилось лишь промежуточной или предварительной формой существования текста или документа.

Сегодня большинство документов никогда не попадает на принтер, и нет нужды подстраиваться под архаичный бумажный документооборот. Вполне возможно, что автору или редактору удобнее выделение не курсивом, а подчеркиванием:

Предложение, набранное курсивом.

Более того, современные технологии (например, HTML или XML-оформление текстов) изначально предполагают, что читатель документа может устанавливать собственные предпочтения в визуализации, зависящие от особенностей используемого им оборудования (размеров и разрешающей способности монитора и т.п.) или от своих биологических особенностей (слабого зрения, дальтонизма и т.п.), и в общем случае они могут не совпадать с предпочтениями автора или редактора.

Таким образом, один и тот же документ может быть отображен с разметкой, с визуализацией, а иногда — и с разметкой и с визуализацией (см. Рис. 4-1).

Рис 4-1

Хотя на рисунке нам удалось показать три типа отображения одного и того же документа, не выходя из одного прикладного пакета, это не такой частый случай. На самом деле текстовые редакторы, как правило, не имеют способности непосредственной визуализации вообще или обладают ею лишь в зачатке (как, например, Emacs, способный визуализировать формат Enriched text, но Emacs это не просто редактор, а целая операционная среда), а word-процессоры, в свою очередь, крайне неудобны для редактирования «плоского» текста: слишком разнятся базовая операторика и ожидаемая эргономика этих двух типов прикладных программ.

Учащийся сталкивается с задачей манипулирования «плоским» текстом как минимум два раза (при знакомстве с электронной почтой и при изучении основ программирования), соответственно, успевает познакомиться, как правило, с двумя разными текстовыми редакторами (встроенными в почтовую программу и среду программирования, соответственно). Как минимум два раза он сталкивается и с задачей манипулирования размеченным текстом: один раз его знакомят с word-процессором (как правило, в России под руку подворачивается «пиратский» «Майкрософт Уорд», либо бесплатно распространяемый «СтарОфис 5», либо дешевый «Лексикон», исключения единичны), а затем ему преподносят основы HTML.

Знакомство с манипулированием текстом, таким образом, оказывается бессистемным и фрагментарным, и, в лучшем случае, автор учебника или учитель сумеют рассказать о том, что сфера это, в общем-то единая, а показать это оказывается весьма затруднительно.

Значительным шагом к систематизации опыта, вырабатываемого школьным курсом, на наш взгляд, является использование инструментария, позволяющего демонстрировать возможность работы с размеченным текстом разными средствами. Это значит, что к очевидным требованиям, предъявляемым к «учебному» word-процессору (достаточность функций, локализованность, мультиплатформенность, ценовая доступность), добавляется серьезное пожелание: стандартность формата разметки.

Это сильно облегчает выбор. На самом деле, на сегодня всем перечисленным требованиям удовлетворяет, по сути, лишь одна программа. Но сначала — немного о стандартах.

Стандарты разметки текста

Существуют и доказали свою устойчивость два основных типа языков разметки.

Первый из них, это семейство, называемое *ML-языками: на эти две буквы заканчиваются аббревиатуры их названий — GML, SGML, HTML,

XML, — а сами по себе эти буквы означают просто «markup language» — «язык разметки».

Второй — разработанный выдающимся американским теоретиком и практиком программирования Дональдом Кнутом язык программирования верстки TeX[68] и его расширения (например, LaTeX). Не будучи официальным стандартом, TeX постепенно вытесняет и замещает прочие языки разметки, предназначенные для набора и верстки текстов (TeX и системы на его основе плохо приспособлены для верстки т.н. «иллюстрированных изданий» с характерным для них богатым насыщением текста графикой, сложными обводами и наложениями текста на графику и пр., и этот сегмент рынка остается пока не стандартизованным).

За пределами этих типов — огромное множество нестандартных (и даже неопубликованных) форматов, зачастую использующих не текстовую, а двоичную форму представления данных (например, файлы «Майкрософт Уорд», «Лексикона» и т.п.). Это исключает возможность применения для работы с такими данными обычных текстовых редакторов и обработку их стандартными текстовыми утилитами, а также сильно затрудняет обратную разработку формата с целью обеспечения импорта и экспорта из независимо написанных программ[69].

Наверное, TeX имеет потенциал к использованию в качестве примера языка разметки (или, точнее, языка генерации разметки), однако вряд ли в средней школе — отчасти потому, что ориентирован на печатную форму в качестве окончательной формы представления содержания, что представляет на сегодня если не экзотическую, то, во всяком случае, достаточно специальную область применения компьютеров, в отличие от *ML-языков, в равной степени ориентированных и на «экран», и на «бумагу».

Судьба *ML-языков

SGML достаточно давно (с 1986 г.) является стандартом на разметку документов, принятым Международной организацией стандартизации (серия ISO 8879). Парадокс заключается в том, что до недавнего времени даже частичные реализации SGML были сравнительно немногочислен-

[68] Это греческий корень, он читается как русское «тех», а не как «текс».

[69] Возможно, использование двоичных форматов и было оправданно во времена, когда позволяло экономить байты памяти и носителей на «персоналках» с крайне ограниченными аппаратными ресурсами. В то же время, современные компьютеры (даже относимые к классу персональных, «стартового уровня») обладают ресурсами, позволяющими организовать гораздо более удобную схему: сжатие «на лету» по стандартному алгоритму стандартно размеченного текста или иных данных.

ными, и его использование ограничивалось рамками государственных организаций (в массе своей оборонных и научных) и крупных корпораций. Гораздо более широкое распространение получили «похожие на SGML» языки, а именно, HTML различных версий, являющийся одним из технологических столпов WWW.

HTML был сознательно создан как «игрушечный SGML»: он не обладал всей гибкостью и мощью последнего, но был очень компактен и легок в реализации и изучении. Одна из сторон «игрушечности» HTML заключается в том, что он подталкивает пользователя к использованию физической, а не логической разметки, и именно поэтому, на наш взгляд, его не стоит изучать в школе.

Однако добавление все новых и новых возможностей и конструкций в HTML в ходе его развития привело к тому, что сложность его существенно выросла и приблизилась к сложности SGML-приложений, при сохраняющейся несовместимости с SGML.

Параллельное развитие двух близких по назначению языков было очевидно нецелесообразным, поэтому дальнейшее развитие WWW предполагает переход на XML — «расширяемый язык разметки», который превосходит по мощности, гибкости и согласованности HTML и является полноценным SGML-приложением. Уже сегодня наиболее развитые WWW-серверы генерируют HTML именно из XML; непосредственно «понимать» последний постепенно учатся и браузеры.

«Молодое поколение выбирает *ML!»

На наш взгляд, принципы расширяемой разметки, реализованные в XML, могут и должны стать одной из базовых составляющих компьютерной грамотности и обязательно должны найти свой путь в школьные учебные планы. Это позволит:

- подвести единую основу и логически связать такие темы, как манипуляция размеченным текстом, гипертекстом и гипермедиа, векторной графикой, электронными таблицами и т.п.,
- приблизить школьную информатику к реальным тенденциям развития информатики и информационной отрасли вообще, вывести ее из закутка «персонального компьютинга»,
- упростить за счет стандартизации задачу выбора (разработки) учебных программ и пакетов.

Задача доступного изложения основ XML и приемов работы с ним сама по себе непроста, как дидактически, так и технически (в частности, нужны определения типов документов (DTD) для учебных задач, достаточно

развитые для демонстрации возможностей языка, но в то же время достаточно простые для понимания XML-документов «с листа» и низкоуровневого редактирования).

Однако одно из основных препятствий на пути использования XML в школе — неразвитость визуализирующих редакторов — уже отпало с появлением офисного пакета «OpenOffice.org». Он сочетает привычные пользователям ПК пользовательские интерфейсы с поддержкой стандартных XML-приложений, таких, как «текстовый документ» (программа «OpenWriter»), «электронная таблица» («OpenCalc»), «презентация» (OpenImpess), «формула» (OpenMath), «гипертекст» (OpenWeb) и, что уже совсем не характерно для «офисного» софта, «векторный рисунок» («OpenDraw»), их взаимного внедрения и связывания.

По сути дела, «OO.o» — это «троянский конь», заброшенный в мир «малой компьютеризации»: «снаружи» он похож на «офис», а «изнутри» (или «с изнанки») представляет собой набор XML-инструментов. «Офисной» стороной он обращен к опыту пользователей персональных компьютеров, инструментальной — к современным, постперсональным вычислительно-коммуникационным системам (включая локальные сети и сети Интернет с возможностями безбумажного документооборота и совместной работы над документами).

«OpenWriter»

«OpenWriter» (далее — OW) — это неофициальное, но уже закрепляющееся название word-процессора из комплекта свободных офисных прикладных программ OO.o (официальным названием, видимо, следует считать Ooowriter)[70].

[70] «В девичестве» (до того, как американская корпорация Sun Microsystems приобрела немецкую компанию StarDivision и свободно лицензировала код, права на который принадлежали ранее последней) пакет назывался StarOffice, а word-процессор — StarWriter; под таким названием он получил известность и обрел достаточную популярность, в том числе, и в России (особенно версии 5.1 и 5.2).

Рис. 4-2

Как уже говорилось выше, все словарные процессоры внешне (по функциональности и интерфейсу) похожи друг на друга, и OW (см. рисунок) — не исключение. Он предназначен для набора, редактирования и оформления текстов на естественных языках (включая многоязычные) и поддерживает:

- физическое и логическое (через механизм стилей) форматирование документа в целом, отдельных страниц, разделов, абзацев и символов;
- шаблоны (наборы стилей и формы документов);
- лингвистическую поддержку (корректные переносы, проверку орфографии и грамматики, тезаурус (русского грамматического и тезаурус-модулей пока нет));
- внедрение и связывание объектов — как из XML-приложений, так и чужеродных (включая растровую графику и результаты выполнения запросов к базам данных);
- импорт/экспорт унаследованных нестандартных форматов (в базовую поставку входит модуль только для Microsoft Office), а также плоско-текстовых и гипертекстовых форматов;
- встроенный макроязык;
- автоматическую нумерацию элементов, оглавления и указатели;

- ... (назовите сами).

За подробностями отсылаю к [6-9].

Интересное, однако, начинается, когда мы посмотрим на OW «с изнанки». Файлы с расширением имени «.sfx», создаваемые им — это PKZIP-архивы, содержащие (в простейшем случае) набор XML-файлов, соответствующих (в терминах XML) манифесту, содержанию документов, определению стилей и значениям текущих настроек.
Заглянем в файл с содержанием (content.xml). Даже не зная XML, и лишь ориентируясь в синтаксисе языка разметки, можно понять, что файл содержит сначала определения стилей, использованных в документе (даже «жесткое» форматирование имитируется в OW путем создания неявных стилей), а затем размеченного указаниями на эти стили текста. Смотрите, заголовок статьи размечен так (Рис. 4-3).

```
<text:p text:style-name="P2">
 <text:span text:style-name="T1">
  Лекция 0.
 </text:span>
 <text:span text:style-name="T2">
  OpenWriter —
 </text:span>
 <text:span text:style-name="T3">
  свободный
 </text:span>
 <text:span text:style-name="T2">
  word-
 </text:span>
 <text:span text:style-name="T3">
  процессор
 </text:span>
</text:p>
```

Рис. 4-3

Понятно, что для форматирования использован один стиль абзаца «P2» и три стиля символов «T1», «T2» и «T3». Выше, в определениях стилей можно найти, что, допустим, «T2» — это стиль, определенный на Рис. 4-4.

```
<style:style style:name="T2» style:family="text">
<style:properties fo:font-weight="bold» style:font-weight-asian="bold»
style:font-weight-complex="bold"/>
</style:style>
```

Рис. 4-4

То есть «текстовый» (символьный) стиль, предполагающий набор и отображение полужирным шрифтом.

Теперь content.xml может обрабатываться любым XML-инструментом уже без использования «OO.o». Его можно преобразовать в HTML или проиндексировать, вывести на печать, просмотреть браузером, поддер-

живающим XML. Произвольные определения документов напрямую пока браузерами не поддерживаются, однако текст (неформатированный) можно уже сегодня просмотреть, просто открыв content.xml в «Мозилла» или другом браузере, поддерживающем XML.

4.2 Редактор электронных таблиц «OpenCalc»

Слово «компьютер», означающее буквально «вычислитель» и восходящее к лат. computare, сегодня не нуждается в переводе: повсеместно им обозначают электронные вычислительные машины, и оно понятно даже носителям языков, в которых для компьютера есть собственное слово. Однако в английском, из которого оно и начало свое распространение, это слово имеет и более ранее значение: человек, занятый вычислениями.

Парадоксально, но сколько-нибудь систематического исследования вопроса о динамике совокупной «вычислительной мощи», которыми располагало человечество до появления автоматических вычислителей, в мировой литературе не существует (по крайней мере, с ходу не находится), хотя само содержание общематематического образования и профессиональной подготовки до сих пор наполнено, наряду с теоретическими сведениями о природе и свойствах математических объектов, вполне прагматическими приемами, способами, методами эффективного ручного счета.

Такое исследование будет непростым предприятием, учитывая, что в большинстве случаев вычисления были не отдельной профессией, а частью других профессиональных занятий. Бухгалтер, инженер, техник затрачивали существенные усилия на проведение расчетов, являющихся частью их повседневной работы. Компьютер меняет все это: от профессионала по-прежнему требуется умение применить способ расчета, но сами «вычислительные объемы» выполняются все более и более автоматизированно.

Лишь в относительно небольшом количестве случаев такие вычисления можно сделать полностью автоматическими и централизованными, свалить их на «числомолотилки», а в большинстве случаев расчеты должны проводиться уместно, в том месте и в тот момент, когда это необходимо. Иными словами, от все большего количества профессионалов ожидается умение программировать вычисления.

«Программирование» здесь употреблено не в узком значении, связанном с определенной профессиональной деятельностью, использованием специальных систем нотаций и интеллектуальной дисциплины, а в самом широком, совпадающем с этимологией слова. «Программировать» озна-

чает буквально всего лишь «прописывать наперед» те действия, которые должны быть выполнены.

«Убойное приложение» ПК

У маркетологов есть такое сленговое выражение: «убойное применение», или «убойное приложение» (killer application). Оно означает то применение какой-либо многофункциональной в своей основе вещи, которое формирует основную долю спроса на нее и превращает ее из модной новинки в массовый товар.

Появившийся на рубеже семидесятых и восьмидесятых, в сравнении с «настоящим» дорогим компьютером дешевый ПК был всем плох, кроме одного: он быстро выводил данные на экран.

Нужно вспомнить, что в те времена относительно дешевые алфавитно-цифровые терминалы соединялись с машиной последовательными интерфейсами (RS-232), скорость передачи данных по которым обычно измерялась в сотнях знаков в секунду, иногда в тысячах. Гораздо быстрее обменивались данными X-терминалы, включенные в сеть, но это оборудование другого класса, стоившее совсем других денег.

«Терминал» персонального компьютера — это (не считая клавиатуры и «мыши») адаптер, интегрированный на системной плате или вставленный в разъем шины с параллельной передачей сигнала со скоростью, как минимум в сотни раз превышающей пропускную способность последовательного интерфейса, а на монитор, находящийся всего в десятках сантиметров от компьютера, сигнал передается по аналоговому кабелю.

Даже на ранних ПК вывод данных на экран был, в масштабах человеческого восприятия, мгновенным (если не «тормозила», конечно, сама программа). Это позволило относительно дешево реализовать различные приложения, сама идея которых в том, чтобы представить пользователю «живую» презентацию каких-либо данных. К числу таких приложений относится полноэкранное редактирование текстов и, в особенности, работа с электронными таблицами. Электронные таблицы — по большому счету, единственное приложение компьютера, которое было придумано для ПК и впервые реализовано на ПК. Сегодня, разумеется, благодаря многократно возросшей скорости передачи данных, редакторами электронных таблиц можно пользоваться в компьютерных системах и сетях практически любой топологии (упомянутый ниже KSpread без проблем заработал на карманном компьютере Sharp Zaurus).

Программирование особого рода

Мы придерживаемся отнюдь не общепризнанной точки зрения, согласно которой популярность электронных таблиц как делового приложения компьютера, обусловлена именно простотой решения задач, требующих программирования.

Электронная таблица — это двумерный массив, каждый элемент (ячейка) которого может содержать либо значение, либо выражение (формулу), причем выражения в качестве связанных переменных могут содержать ссылки на другие ячейки. (Можно считать значение (константу) частным случаем формулы, однако по историческим и эргономическим соображениям синтаксис этих сущностей различен. Значения, к которым приводятся ячейки, содержащие формулу «=100» (если, как во всех известных нам системах управления электронными таблицами, синтаксис формулы предполагает «=» в первой позиции) и константу 100, равны.)

По сути дела, электронная таблица предполагает использование простого функционального языка программирования (точнее, современные системы управления электронными таблицами как правило реализуют язык формул, функциональный в своей основе, но с элементами инфиксной нотации, т.е. с возможностью вместо «=функция1(функция2((сумма(a; произведение(b;c)))))» написать чуть короче: «=функция1(функция2 (a+b*c))»).

Синтаксис этого языка очевиден для всех, кому ясно, что такое формула в обычном математическом понимании. Кроме того, форма электронной таблицы снимает с пользователя-«программиста» заботу об организации данных (их организует сама таблица, и вместо имен переменных можно использовать координаты ячеек), о вводе-выводе и о связывании отдельных конструкций в программу (вычисление формул происходит по мере необходимости).

Таким образом, с помощью электронных таблиц в учебный курс информатики можно вводить «нулевую степень программирования», объясняя, что такое выражение и переменная, но откладывая на потом то, от чего можно абстрагироваться (сущности, перечисленные в предыдущем абзаце, и другие, более сложные).

Свободные редакторы электронных таблиц

Пробежавшись по каталогам свободных программ (таким, как «кузница кода» Sourceforge, содержащая тысячи проектов), можно обнаружить более двух десятков программ в категории «электронные таблицы».

Большинство из них — незавершенные или более или менее законченные учебные проекты. Работу с электронными таблицами можно, видимо, считать зрелым персонально-компьютерным приложением: оказывается, за полгода-год один программист в состоянии реализовать (разумеется, опираясь на существующие библиотеки) до 90% функциональности, свойственной лидирующим программам в этой категории.

Однако знакомство с содержанием коммуникации на форумах поддержки позволяет предположить, что реальную широкую пользовательскую аудиторию получили три свободных проекта:

- «OpenCalc» — электронно-табличный компонент уже знакомого нам по предыдущим разделам интегрированного прикладного делового пакета «OpenOffice.org»;
- KSpread — компонент еще одного конкурирующего пакета под названием KOffice, который мы пока обходим вниманием. KSpread сегодня также не будет нами рассматриваться, но по чисто техническим причинам. Ничего плохого мы о нем сказать не можем. И, наконец,
- Gnumeric — компонент слабоинтегрированного пакета (или, скорее, собрания программ) «Гном» Office, не слишком популярного в России из-за хронических сложностей с кириллической письменностью, свойственных word-процессорному его компоненту, Abiword. Сразу отметим, что, в отличие от последнего, Gnumeric «русофобией» не страдает.

Упомянутое выше слово — зрелость самого приложения — ключевая характеристика. Набор ожиданий пользователя, в общем-то, известен, причем не только в части функциональности, но и в части основных эргономических характеристик программы: помимо богатых выразительных возможностей самих функций, для работы с электронными таблицами важна т.н. «остенсивная» операторика, иными словами, возможность «показать пальцем» на объект, с которым нужно произвести те или иные действия. Например, «суммировать значения *вот этих* ячеек», а не «... ячеек с A5 по D5». В большей, чем в других приложениях, мере очевидны эвристики, которые должны реализовываться программой в качестве «подсказок»; например, если в ячейку, завершающую длинный столбец чисел, пользователь намерен ввести формулу, скорее всего он суммирует значения, а если он начал ряд «1 2 3» или «январь, февраль, март», скорее всего, он продолжит его достаточно прямолинейно.

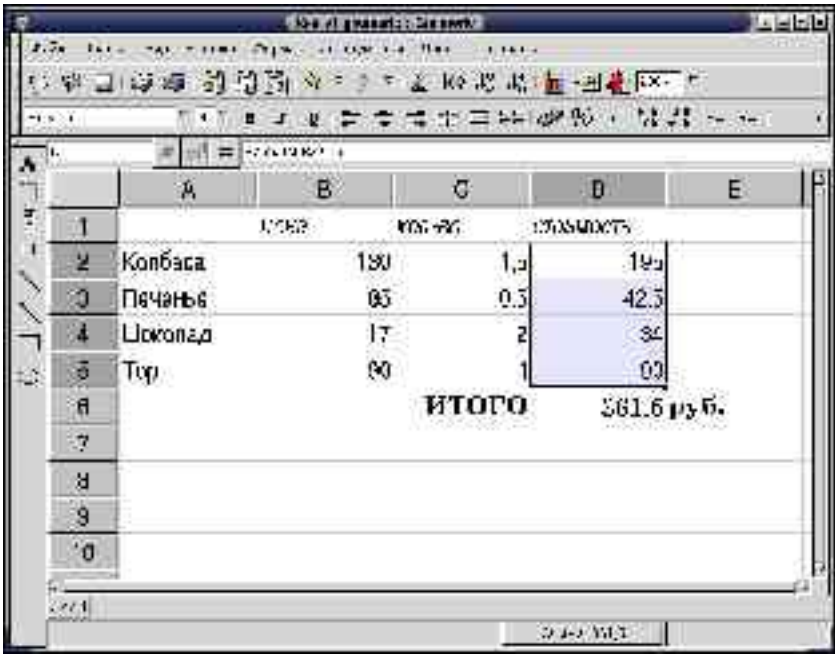
A
B
C
D
E
1
2 Колбаса 180 1,5 195
3 Печенье 85 0,5 42,5
4 Шоколад 17 2 34
5 Тор 90 1 90
6 ИТОГО 361,6 руб.
7
8
9
10

Рис. 4-5

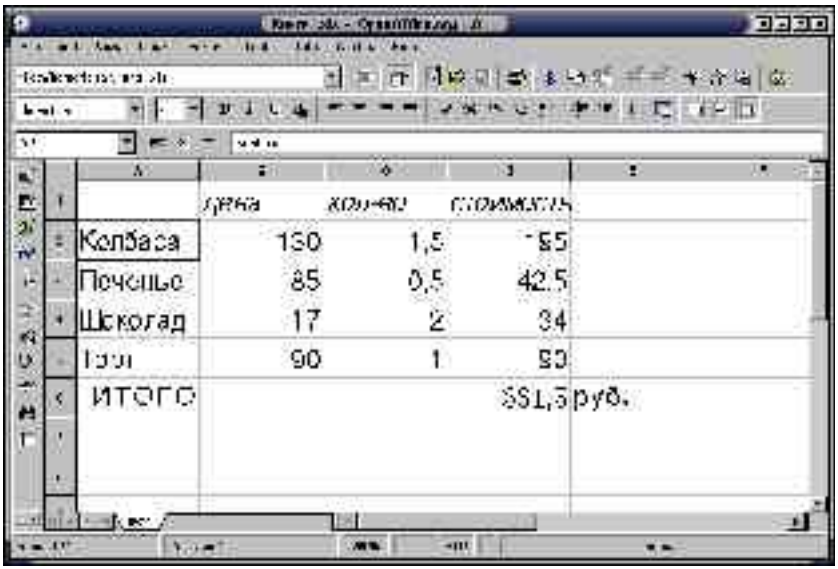
Колбаса 180 1,5 195
Печенье 85 0,5 42,5
Шоколад 17 2 34
Тор 90 1 90
ИТОГО 361,5 руб.

Рис. 4-6

Соответственно, различий можно ожидать лишь в деталях реализации функциональности и эргономики. И действительно, большинство редакторов электронных таблиц, включая и перечисленные, очень похожи.

За две недели автор (в обычной жизни не пользующийся этим классом программ), проверяя свои впечатления, «играл» с четырьмя подобными системами, пытаясь решать несложные задачки, которые обычно он решает (ввиду специфики личного профиля навыков) с помощью СУБД, включая 1) элементарные инженерные расчеты (расход материалов и жесткость корпусной мебели), 2) бюджетирование небольшого проекта, 3) бюджетирование личных расходов. Под горячую руку попали и 4) три задачки из учебника алгебры за 11 класс, две из которых даже удалось с ходу решить.

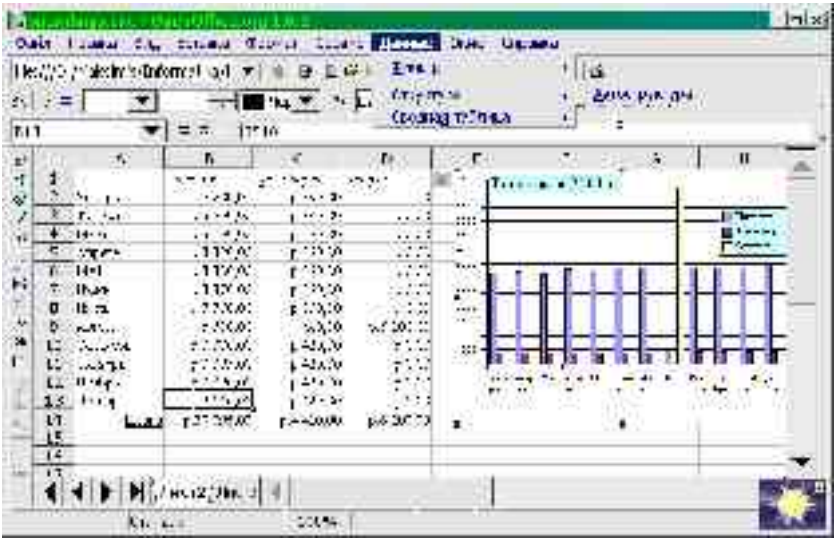

Рис. 4-7

В число этих систем вошли три упомянутые свободные программы и, в качестве контрольного образца, очень популярная несвободная «Майкрософт Эксел» (из офисного пакета «Майкрософт Офис 2000»). Вывод: существенной разницы в возможностях и способах их реализации 99% пользователей не обнаружат. Некоторые тонкости: самой «интуитивной» показалась «Ньюмерик», лучше всего документирована «Майкрософт Эксел», у последней также наиболее развиты средства визуализации (построение графиков и диаграмм).

Тем не менее, если бы автору предстояло регулярно работать с электронными таблицами, скорее всего, его выбор пал бы на сравнительно «се-

ренькую» «OpenCalc» из-за единства интерфейса и интегрированности с «OpenWriter» и «OpenDraw», которыми он пользуется регулярно.

Обзор возможностей «OpenCalc»

Как и остальные упомянутые программы (за исключением Microsoft Excel), «OpenCalc» определяет особое приложение языка разметки XML, которое и используется для хранения рабочих книг (почему-то workbook переводится как «рабочая книга», хотя вообще-то это обычная «тетрадь») с подшитыми в них листами электронных таблиц. Как и остальные компоненты «OpenOffice.org», «OpenCalc» упаковывает XML-файл с содержимым (а также ряд вспомогательных файлов) в PKZIP-архив, который и является единицей хранения документа. (О важности стандартизации языков представления данных в «офисных» приложениях мы подробно говорили в разделе 4.1, к каковому и отсылаем читателя.)

Кроме «родного» формата, «OpenCalc» «понимает» распространенный формат, используемый Microsoft Excel разных версий, экспортирует данные в DIF (Data Interchange Format), форматы ранних версий StarCalc, потомком которых она является, SYLK, импортирует — также из форматов dBase и Lotus 1-2-3. Книгу (workbook) можно с очень приличным качеством экспортировать в гипертекст (html 3.2).

Текущая версия (1.0) «OpenCalc» позволяет работать с отдельными таблицами (листами) размером до 255 столбцов (пронумерованных буквами и двухбуквенными сочетаниями, от A до IV) на 32000 строк (пронумерованных числами), чего вполне достаточно для большинства офисных применений и уж, во всяком случае, для любых разумных учебных задач. «OpenCalc» допускает абсолютную и относительную адресацию ячеек и их диапазонов.

В «OpenCalc» поддерживается типизация данных с возможностью их интерпретации как чисел, денежных сумм, дат, времени, логических значений и, наконец, просто текста. Возможны и определяемые пользователем типы. Для некоторых типов определены различные форматы представления, задающие способ их отображения или печати. В случае, если ячейка содержит формулу, ее результат также может быть типизован.

Библиотека функций OCalc достаточно компактна — их около трех с половиной сотен. Она разбита на ряд категорий: управление БД, работа с датами и временем, финансы, статистика и т.п. Имеются средства расширения этого набора.

«OpenCalc» реализует такие средства, как:

- автозаполнение однородных рядов данных;

- именование ячеек и их групп;
- сортировка и фильтрация;
- построение графиков и диаграмм.

Мощный механизм стилей оформления, свойственный всем компонентам пакета «OO.o», доступен и в «OpenCalc». Стили оформления могут определяться для отдельных ячеек, их совокупностей, листов и рабочих книг в целом, а также для включаемых элементов, таких как текст или иллюстрации (в том числе, графики и диаграммы).

4.3 Редактор векторной графики «OpenDraw»

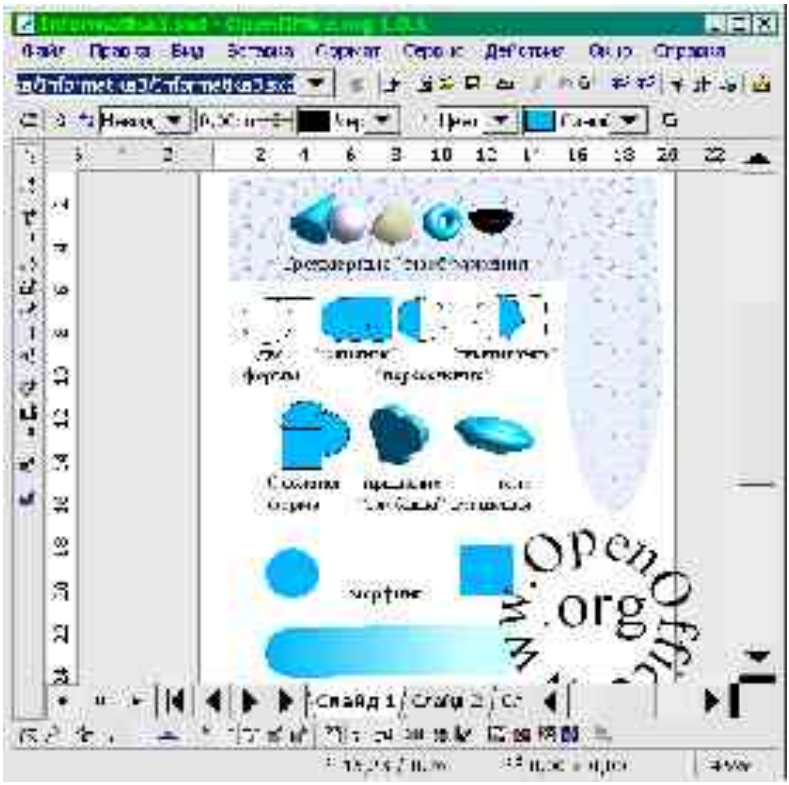

Рис. 4-8

Компьютерная графика, чуть более десяти лет назад бывшая если не экзотическим, то довольно специальным приложением компьютера, сегодня повсеместна.

Всякий, впервые сталкивающийся с компьютером, уже в самом начале

обучения (если курс специально не построен так, чтобы ограничиться на первых порах программами с текстовым или псевдографическим интерфейсом), скорее всего, встретится с графическим терминалом, отображающим различные графические элементы (включая и текстовую информацию, также отображающуюся в графическом режиме средствами компьютерной графики).

Элементарная обработка (создание и редактирование) изображений (как векторных, так и растровых) входит в число приложений, перечисленных в «Обязательном минимуме содержания образования по информатике» [1] в качестве обязательных к освоению в курсе средней школы.

Предметность обучения предполагает, что учащийся «узнает» в создаваемых или редактируемых им картинках при изучении обработки изображений подобие уже знакомых ему элементов графических интерфейсов. Чтобы это счастливое узнавание произошло, представляется целесообразным включить некоторые первоначальные сведения о компьютерной графике уже во введение в предмет, которым начинается его изучение. К сожалению, не все учебники это реализуют, поэтому преподавателю могут потребоваться некоторые дополнительные усилия, но, на самом деле, небольшие, поскольку наглядный материал «под рукой» в любом компьютерном классе.

Векторное и растровое кодирование изображений

Способы кодирования графических данных делятся на две категории: растровый (точки равномерно размеченного прямоугольника) и векторный (описания линий и фигур, составляющих изображение).

Например, растровое изображение окружности может быть таким: «заполняем квадрат 5x5: белая точка (Б), черная точка (Ч), Ч, Ч, Б, Ч, Б, Б, Б, Ч, Ч, Б, Б, Б, Ч, Ч, Б, Б, Б, Ч, Б, Ч, Ч, Ч, Б». (Более умные «форматы со сжатием», возможно, позволят сократить запись до чего-нибудь, подобного: «заполняем квадрат 5x5: Б, 3Ч, Б, 3(Ч, 3Б, Ч), Б, 3Ч, Б», а то и еще более компактно, но суть не в этом).

Векторное ее изображение совсем другое: «черная окружность с центром в (3,3), радиусом 2 и толщиной линии 1».

Важно понимать, что большинство компьютерных графических устройств сегодня — и терминалы, и сканеры, и принтеры — по своей природе растровые. Исключение составляют только графопостроители (плотерры) и редко доступные в учебных условиях их промышленные «кузены» (такие, как фрезерный станок с числовым управлением).

Соответственно, изображение, вне зависимости от его собственной при-

роды (векторной или растровой), будет при отображении преобразовано в растр и доступно человеческим чувствам в растровой форме (даже если растр почти незаметен, как на дисплеях с высоким разрешением, или вообще имеет сетку ниже порога восприятия, как при печати на качественных струйных или лазерных принтерах). И указанные два способа вполне могут привести к неотличимому результату.

Применение векторной и растровой графики

Разница может проявиться тогда, когда мы начнем обрабатывать элементы изображения.

Векторное изображение может без ущерба масштабироваться (увеличиваться или уменьшаться), причем эта операция обратима. В приведенном примере мы можем без труда увеличить векторную окружность в пять раз: «черная окружность с центром в (15,15), радиусом 10 и толщиной линии 5», и она останется окружностью.

А вот растровое изображение при увеличении обнаружит лишь свой растр в увеличенном виде (и на окружность наш пример похож уже будет мало). Более того, операция уменьшения (или увеличения в нецелое число раз) растрового изображения уже не является обратимой: информация при осуществлении такой операции безвозвратно теряется.

Кроме того, векторное изображение, содержащее более одного элемента (например, изображение двух пересекающихся окружностей), может быть разъято на элементы без каких-либо потерь. С растровым изображением такое, в общем случае, невозможно: программа «не знает» об элементах изображения и о принадлежности тех или иных точек отдельным элементам.

В общем случае, векторное кодирование хорошо подходит для работы с чертежами, схемами, картами, диаграммами, графиками и т.п., а растровое — для обработки фотографий и рисунков. Векторное изображение может быть без ущерба для восприятия преобразовано в растр, в то время, как обратное преобразование в общем случае проблематично (хотя есть программы, позволяющие с той или иной точностью распознавать графические образы). Векторное кодирование часто компактнее растрового, но его отображение требует больших вычислительных ресурсов[71].

Пример, который всегда перед глазами пользователя компьютера —

[71] Мы не касаемся здесь специального вопроса о сложных форматах кодирования, включающих векторные элементы и фрагменты растра, так же, как и — растрового и векторного — кодирования движущихся изображений и вопросов сжатия растровых изображений.

шрифты. Когда точно известен масштаб, в котором будет отображаться текст, часто используются растровые («фиксированные») шрифты, представляющие собой набор растровых изображений отдельных букв и других символов используемого алфавита (например, растровые шрифты используются видеокартой, работающей в текстовом режиме, но также часто и как элемент графического интерфейса). Их применение позволяет отображать символы с недостижимой иным способом четкостью, но, если нужно более одного размера шрифта, потребуется трудоемкая работа по разработке нескольких таких шрифтов.

Когда необходимо свободное масштабирование шрифтов, используются векторные шрифты (на самом деле, большинство масштабируемых шрифтов — это библиотеки программ, «рисующих» соответствующие буквы, часто - с учетом важности их отдельных элементов для восприятия; но разница в данном случае несущественна). Качество отображения мелкого кегля или на экране с низкой разрешающей способностью у них ниже, но они универсальнее.

Прочие элементы графических пользовательских интерфейсов также используют как векторную (большинство органов управления и большинство элементов оформления окон), так и растровую (пиктограммы-«значки») графику.

Свободные программы для работы с векторной графикой

Из множества свободных программ, предназначенных для редактирования векторно-графических файлов, разумные потребности большинства учебных курсов может удовлетворить любая из следующих:

- очень простой векторный редактор Xfig для оконной системы XFree86,
- sodipodi (обычно поставляется со всеми операционными системами, включающими среду «Гном»),
- Kontour (компонент популярного «офисного» пакета KOffice)

и, наконец,

- «OpenDraw» (входящая в пакет «OpenOffice.org»).

Рекомендуется при возможности хотя бы вкратце ознакомиться с каждой из них, чтобы понять, какая лучше подходит для конкретного учебного курса. В этом разделе далее будет обсуждаться «OpenDraw», исходя из уже описанных преимуществ, но это ни в коей мере не означает непригодность для учебных целей прочего из перечисленного.

Общие характеристики «OpenDraw»

Как и остальные компоненты пакета «OO.o», «OpenDraw» использует в качестве «родного» формата специально разработанное XML-приложение (об XML и о том, почему ориентация на этот стремительно набирающий популярность формат данных важна при выборе учебных программ, см. раздел 4.1).

Этот пакет доступен для популярных стандартных («ГНУ/Линукс», «Солярис») и нестандартных («Майкрософт Уиндоуз») операционных систем и прилично (хотя и не идеально) локализован.

Самым же существенным недостатком «OpenDraw» являются относительно высокие требования к аппаратным ресурсам, поэтому его использование затруднительно на старых или маломощных компьютерах (для комфортной работы над несложными учебными упражнениями должно быть достаточно Intel P-II, Celeron или K6-2 с частотой от 500 МГц или G3 с частотой от 350 МГц при памяти от 64 МБ). Если нужна демонстрация векторно-графических возможностей на таких компьютерах, мы рекомендуем sodipodi или еще более «легкую» XFig.

Сохранение в «чужих» векторных форматах (экспорт) на сегодня реализовано только для ранних версий предшественника «OpenOffice.org» пакета StarOffice (и StarDraw как отдельной программы). Зато импорт (чтение «чужих» форматов) возможен не только из универсальных векторных форматов, но и из DXF, используемого в популярных системах автоматизированного проектирования (САПР).

«OpenDraw» также позволяет экспортировать рисунок во многие растровые форматы или в гипертекстовую страницу.

Следует понимать, что StarDraw, на основе которого разработана программа, задумывалась как «офисный» графический редактор, прежде всего предназначенный для создания и редактирования графических элементов оформления документооборота (сопровождающих документы рисунков, карт, диаграмм, графиков и пр.). Поэтому действия, типичные для такой работы, максимально облегчены и «вынесены на первый план», и «OpenDraw» содержит массу готовых деталей, широко употребимых в «офисной» графике (например, готовых стрелок и множества соединительных линий, часто используемых в таких случаях). Для технического черчения и схемотехники «OpenDraw» не приспособлена (хотя при случае в ней можно создать простой чертеж или электронную схему).

Документ «OpenDraw» называется «рисунком», что несколько дезориентирует, поскольку на самом деле он может содержать целую «пачку»

отдельных изображений, называемых слайдами. Каждый слайд, в свою очередь, может содержать один или более слоев .

Рисунок сохраняется в файле, который технически представляет собой PKZip-архив, включающий стилевые определения и собственно содержание документа на языке XML. Какой-либо инструментарий, специально предназначенный для обработки «OpenDraw» XML, нам не известен.

Как и остальные компоненты «OpenOffice.org», «OpenDraw» в высшей степени конфигурируемая и настраиваемая под конкретное применение программа. Для использования в учебном контексте может оказаться полезным предварительно настроить ее для некоторого сокращения обилия непосредственно доступных пользователю функций, чтобы ученик в них не затерялся.

Основные возможности «OpenDraw»

Функциональность «OpenDraw» сопоставима с большинством других редакторов векторной графики и включает:

- создание, форматирование и преобразование объектов-графических примитивов: отрезков и стрелок; квадратов и прямоугольников; окружностей, эллипсов, дуг, сегментов и секторов; кривых Безье, «свободных» кривых, ломаных и многоугольников;
- создание, форматирование и редактирование объектов-текстовых надписей (при вводе или редактировании текстов доступна большая часть функциональности word-процессора OpenWord (см. раздел 4.1)). На самом деле, текст может содержаться в любом замкнутом контуре. В состав «OpenDraw» также входит декоративная функция FontWork, позволяющая расположить текст «фигурно» (вдоль дуги или окружности);
- импорт растрово-графических объектов (в том числе, и в качестве текстур заливки векторно-графических примитивов).

Преобразование объектов включает:

- изменение характеристик линий контура (они могут быть разного цвета, сплошными, пунктирными, двойными и т.п.) и заливки (заполнения; оно возможно отдельным цветом, цветовым градиентом (переходом) и даже растровым изображением) замкнутого контура;
- перемещение, изменение размеров и поворот;
- группирование (превращение нескольких в один) и разгруппирование объектов; комбинирование и раскомбинирование объектов (разница между группированием и комбинированием слиш-

ком тонка для нашего обзора);
- вращение и зеркальное отображение;
- преобразование примитивов — линий и контуров — в произвольные кривые или ломаные;
- выравнивание и равномерное распределение (по вертикали или горизонтали, относительно края или центра);
- специальные операции, некоторые из которых описаны ниже.

Большинство описанных операций преобразования доступны как посредством меню (к пунктам которого могут быть привязаны «горячие» клавиши), так и на пиктографических палитрах инструментов, наряду с палитрой цвета присутствующих (по желанию) в главном окне программы.

Точное позиционирование объектов облегчает возможность отобразить сетку разметки слайда, а также команды выравнивания объектов относительно узлов этой сетки, края или центра листа (слайда), а также относительно друг друга. Особенно полезно это при создании достаточно формализованных объектов.

Механизм стилей

Важной особенностью «OpenDraw» является использование механизма стилей для форматирования графических элементов. Использование стилей в векторной графике вполне сопоставимо с использованием того же механизма при оформлении размеченного текста (см. раздел 4.1) и имеет те же преимущества перед «жестким» форматированием/оформлением.

Этот прием позволяет серьезно облегчить вариантное оформление рисунков, отказавшись от «жесткого» (такого, при котором для каждого объекта или группы объектов характеристики задаются вручную) форматирования объектов.

Например, при подготовке схемы оргструктуры учреждения можно создать стиль, которым оформляются изображения подразделений, допустим, с голубой заливкой в черном контуре в 3 пункта толщиной. Если позднее потребуется изобразить те же подразделения желтыми прямоугольниками без рамки, не нужно будет помечать их вручную (представьте, насколько это может быть трудоемко даже для относительно простой организации типа средней школы), достаточно будет лишь изменить параметры стиля. Более того, можно определить стиль и для стрелок, показывающих связи между подразделениями, с «наследованием» цвета от цвета стиля подразделений, если предполагается, что их цвет должен совпадать.

Стиль можно определить и для слайда в целом, что позволяет единооб-

разно оформлять серию рисунков. Набор стилей можно объединить в шаблон (сохраняемый в отдельном файле), если предполагается регулярное его использование.

«Логические операции» над объектами, «3D» и морфинг

Интересной функцией «OpenDraw» является наличие так называемых логических операций над объектами. Два или более заполненных контура могут быть скомбинированы особым образом, что порождает новый объект, являющийся объединением форм, их пересечением или «вычитанием» одной формы из другой.

Еще одной интересной особенностью программы является встроенная в нее базовая функциональность имитации трехмерной графики (3D-функции). Она ограничена 1) конструированием тел вращения, 2) преобразованием в тела вращения произвольных двумерных фигур, а также 3) экструзией (приданием последним «глубины»). Трудно понять необходимость рутинного конструирования тел вращения в «офисной» графике, однако для школы возможность такой демонстрации возможностей компьютерной графики может быть весьма полезна.

Двумерный морфинг (плавное перетекание фигур) реализован в «OpenDraw» не лучшим с точки зрения эстетического результата образом, однако интересным дидактически: на самом деле, программа просто группирует заданное (в качестве «количества шагов» морфинга) количество «промежуточных» (по форме и цвету) фигур между морфируемыми фигурами. Созданную таким образом группу можно разгруппировать и посмотреть, что и как, собственно, программа сделала.

Векторизация растровых изображений

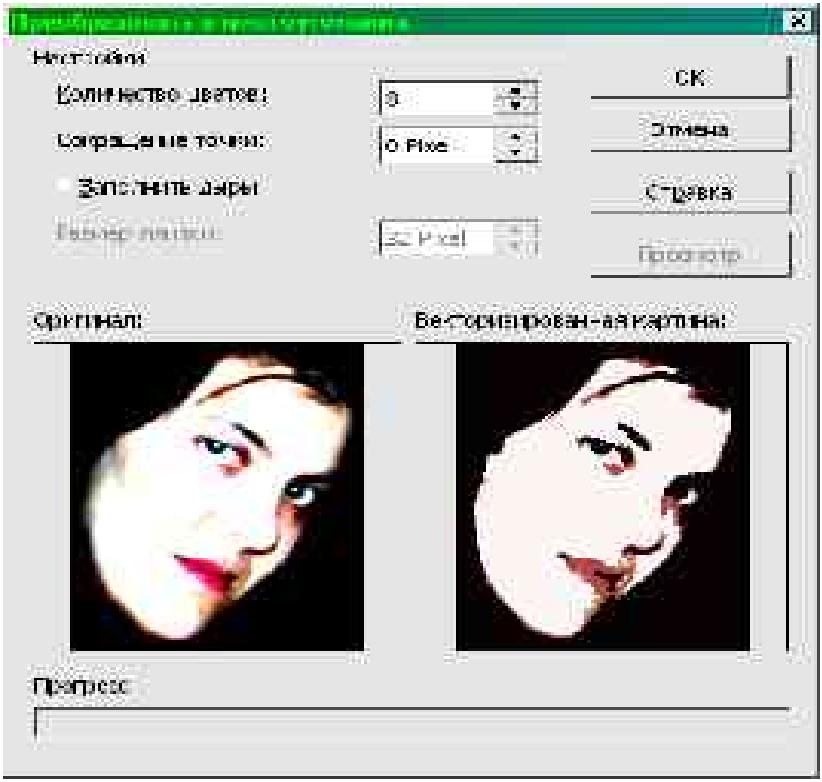

Рис. 4-9

Выше уже было замечено, что, хотя преобразование векторного изображения в растр — элементарная техническая задача, гарантированное обратное преобразование в общем случае невозможно. Тем не менее, существуют программы, которые пытаются это сделать.

«OpenDraw» обладает функцией векторизации, «спрятанной» в меню «Преобразовать» под кличкой «В многоугольник». Если применить данный пункт меню к импортированному растровому изображению, появится окно предпросмотра с возможностью задать некоторые параметры и увидеть результат.

Ничего мистического в этой функции нет: программа ищет связные области, залитые одним или близкими цветами, и описывает их контур как многоугольник. Затем она группирует полученные фигуры одного цвета и переходит к следующему цвету или группе цветов. Результирующий

векторный объект представляет собой группу, в свою очередь состоящую из цветовых групп.

Более глубокого анализа программа не проводит (в принципе, можно было бы каждый выделенный объект пытаться аппроксимировать с заданной точностью отрезком, коническим сечением или кривой Безье; это позволило бы распознать элементарные геометрические фигуры, например, просканировав простой рисунок).

Эта функция предусматривает слишком мало параметров, чтобы быть особенно полезной на практике, однако для демонстрации того, в чем состоит процесс векторизации, она вполне подходит.

Глава 5. Редактор растровой графики «ГИМП»

5.1 Источники и параметры растровой графики

Обычным источником растровых изображений является сканер — устройство, «проходящее» (сканирующее) лист бумаги или кадр фотопленки точка за точкой и передающее компьютеру значения, соответствующее интенсивности базовых цветов в каждой точке. Все большей популярностью пользуются цифровые фотокамеры — аппараты, вместо фотопленки фокусирующие изображение на светочувствительной матрице, передающей цифровую запись изображения на энергонезависимый носитель информации (гибкий диск или т.н. флэш-карту), который затем может читаться компьютером.

Растровые изображения могут также создаваться человеком на компьютере с помощью устройств координатного ввода («мыши» или более подходящего для этой цели графического планшета) или синтезироваться различными программами.

Важнейшими параметрами растрового изображения являются его растровый размер (в точках) и «глубина цвета» (количество бит, используемых для представления цвета каждой точки). Эти параметры часто записывают в виде 1024x728x24, что означает 1024 точки по горизонтали, 728 точек по вертикали и 24 бит на цвет (двадцати четырех бит достаточно для получения «фотореалистичных» изображений, дальнейшее повышение разрядности не приводит к увеличению качества отображаемых или печатаемых современными средствами изображений, хотя в промежуточной обработке или при синтезе изображений иногда используется большие значения глубины цвета). Еще одним параметром, предусмотренным некоторыми форматами хранения растровой графики, является его «масштаб», который принято измерять в точках на дюйм (DPI); это чисто информационный параметр, который может учитываться при печати изображений, но не влияет на возможность их обработки.

5.2 Источники и параметры и форматы представления растровой графики

За тридцатилетнюю историю компьютерной графики разработано великое множество (сотни) форматов хранения изображений. Большинство из них является плодом несогласованности «технического творчества» отдельных групп исследователей и компаний, а также отражает особенности давно вышедших из употребления специфических устройств. Важными свойствами форматов являются присущие им внутренние ограничения, из которых наиболее значимы ограничения на глубину цвета, под-

держиваемые цветовые модели («мониторная» RGB, «полиграфическая» CMYK и т.д.) возможность сохранения нескольких слоев изображения (понятия слоев, контуров и масок здесь не обсуждаются), наличие т.н. «альфа-канала» (фиктивного «цвета», соответствующего степени прозрачности изображения при наложении его на другое изображение) и поддерживаемые алгоритмы сжатия.

Применяемые в растровой графике алгоритмы сжатия подразделяются на неразрушающие и форматы с потерей качества (последние используют психофизиологическую модель человеческого зрения для того, чтобы избавиться от незначимых деталей изображения, что несколько снижает его качество, но позволяет добиться гораздо большей компактности кода).

Практически универсальными являются форматы TIFF (обычно применяемый в полиграфии, допускает лишь неразрушающее сжатие), PNG (наиболее удобный для представления графических данных в мультимедийных системах и WWW, допускающий неразрушающее сжатие), а также медленно вытесняемые последним JPEG (допускающий сжатие с потерями) и GIF (ограниченный 8 бит (256) цветами). Кроме того, многие графические редакторы (включая описанный ниже «ГИМП») обладают собственным форматом, позволяющим сохранять в том же файле массу вспомогательной информации, полезной при продолжающейся более одного сеанса работе с файлами.

5.3 Общие сведения о «ГИМП»

GNU Image Manipulation Program (Программа ГНУ для манипуляции изображениями), или сокращенно GIMP — потомок курсового проекта двух студентов, Питера Маттаса и Спенсера Кимболла (1985-86 гг.).

За восемь лет и при участии десятков программистов «ГИМП» вырос в один из самых насыщенных функциональностью графических редакторов, уступающих лишь «под завязку» набитому сторонними модулями редактору Photoshop[72]. Текущая стабильная версия — 1.2 (как и в некоторых других проектах нечетным «малым» номерам соответствуют экспериментальные (разработческие) версии, а четным — стабильные).

Интересно, что в рамках проекта «ГИМП» была создана библиотека работы с экранными примитивами GTK (ныне GTK+), являющаяся на сегодня одной из самых развитых и широко используемых в своем классе (в частности, на GTK+ основана популярная графическая операционная сре-

[72] Стоит отметить, что стоимость одной копии последнего, самого по себе не дешевого, вместе со всеми такими модулями составляет несколько тысяч долларов.

да «Гном»).

«ГИМП», доступный для всех популярных настольных платформ (включая стандартные, а также «Майкрософт Уиндоуз», «МакОС» и др.), широко используется для работы над WWW и мультимедийной графикой, обработки любительского, репортажного и даже художественного фото, ретуширования кинокадров. Правда, для последней цели чаще используется модификация, известная как FilmGIMP, ее, в частности, активно эксплуатировали при монтаже первого фильма о Гарри Поттере.

«ГИМП» редко используется для предпечатной подготовки графики: в нем пока нет поддержки «полиграфических» цветовых моделей и системы цветоделения. Еще одним ограничением текущих версий «ГИМП» является относительно низкая производительность, затрудняющая работу с действительно большими (сотни тысяч точек) и сложными (десятки слоев) изображениями. Для создания и обработки достаточно компактных и простых изображений, а также для их отображения на мониторе и печати на оборудовании потребительского класса эти ограничения не важны.

«ГИМП» способен работать с продвинутыми координатными устройствами — графическими планшетами, в том числе, моделями, распознающими силу нажатия на перо.

5.4 «ГИМП» — программируемый графический редактор

Вероятно, успехом «ГИМП» не в малой степени обязан своей изначально модульной и программируемой архитектуре. Сам по себе этот редактор — достаточно компактная и простая программа, однако его возможности приумножаются за счет открытости архитектуры и наличия множества модулей (в текущей поставке — около трехсот), реализующих те или иные дополнительные функции, такие как импорт-экспорт сторонних форматов или обработку изображения или его фрагмента по тому или иному алгоритму.

Такие модули можно разрабатывать как отдельные программы с использованием библиотеки GDK, а можно пользоваться одним из встроенных в «ГИМП» интерпретаторов языков программирования. Именно наличие таких интерпретаторов и делает «ГИМП» программируемым графическим редактором. Можно сказать, что его архитектура подобна архитектуре текстового редактора Emacs.

Таких интерпретаторов на сегодня два. Забавное название Script-fu, видимо, следует понимать как английско-китайское выражение, означающее «мастерство сценирования» (напомним, что «сценариями» («скрип-

тами») называют программы, написанные на интерпретируемых языках). Script-fu — это первый встроенный в «ГИМП» интерпретатор функционального языка Схема (Schema), являющегося потомком первого языка функционального программирования Лисп. Разработчик скрипта имеет доступ к многочисленным базовым функциям-графическим примитивам.

Схема — весьма продуманный и стройный язык, однако его методический потенциал не может в полной мере быть раскрыт в сегодняшних курсах информатики в средней школе, программирование в которых вводится в директивной (сентенциональной) парадигме. На Схеме, так же, как и на Лиспе, можно писать в директивном стиле, однако изящества и простоты, столь необходимых в обучении, достичь при этом возможным не представляется.

Директивен другой язык, интерпретатор которого также встроен в «ГИМП». Это Перл (Perl), его «ГИМП»-овская реализация называется Perl-fu. Хороший программист может писать на Перле чисто и аккуратно, однако синтаксис языка сам по себе настолько гибок (чтобы не сказать «жидок»), что, по нашему мнению, Перл совершенно не годится на роль изучаемых в числе первых (хотя существуют и другие мнения, весьма авторитетные).

Добавление в «ГИМП» еще одного интерпретатора (например, алголо- или паскалеподобного языка) не представляется особо сложной задачей, однако о таких проектах ничего не известно, и, наверное, от программирования обработки изображений «внутри» «ГИМП» как от простой в методическом освоении темы большинству педагогов стоит пока отказаться.

5.4 Интерактивная функциональность и эргономика

Тем не менее, базовой интерактивной функциональности «ГИМП» (включая доступные модули) вполне достаточно, чтобы покрыть обычно изучаемые в школе вопросы обработки графики.

Рис. 5-1

При запуске «ГИМП» на экране открывается ряд окон (Рис. 5-1). Главное окно содержит меню основных функций, панель пиктографически обозначенных «инструментов» и области, в которых отображаются текущие значения основного и фонового цветов, формы кисти, текущего градиента. Окна изображения соответствуют отдельным открытым графическим файлам (или слоям в них).

Дополнительные инструментальные окна (по какой-то причине названные «диалогами») могут открываться из меню главного окна. Однако их всего десяток («Слои, каналы и контуры», «Параметры инструментов», «Кисти», «Шаблоны», «Градиенты», «Палитра», «Устройства ввода», «Индекс документов», «Консоль ошибок»), а основной массив функций, применяемых к текущему или вновь создаваемому изображению или выбранному участку изображения (включая функции, реализованные внешними модулями-фильтрами) «достаются» через контекстное меню, открывающееся по щелчку правой кнопкой мыши в окне изображения.

Рис. 5-2

Рис. 5-3

Зато можно воспользоваться (непривычным для пользователей упрощенных графических сред, таких, как «Майкрософт Уиндоуз») свойством графической библиотеки GTK+, называемым «линия отрыва». Каждое меню наверху содержит пунктирную линию, щелкнув мышью на которой можно превратить это меню в самостоятельное инструментальное окно, сохраняющееся, пока пользователь явным образом его не закроет (Рис. 5-2). Таким образом (учитывая настраиваемость самой системы меню) можно в любой момент создать на экране дополнительные «панели инструментов», содержащие функции, которыми в ближайшее время предполагается воспользоваться. Привыкшему к другому стилю работы пользователю это непривычно, однако, привыкнув, эту интерфейсную особенность можно использовать весьма эффективно.

Множественность окон, к сожалению, может создавать некоторые неудобства в средах без развитого инструментария управления окнами. В более развитых средах можно «склеить» несколько окон, чтобы они перемещались как единое целое или «поднять» окно, чтобы оно продолжало оставаться видимым, даже если будет активизировано окно, расположенное «под ним» (собственно, многооконное визуальное решение и создано в расчете на наличие таких средств), а если такие средства отсутствуют (как, например, в «Майкрософт Уиндоуз»), пользование редактором на мониторе с малым разрешением может быть отягощено необходимостью совершать какие-то дополнительные действия.

Основная интерактивная функциональность, доступная посредством «инструментов» в главном окне, достаточно традиционна для программ этого класса. Она включает, в том числе:

- выделение области изображения (прямоугольной, эллиптической или произвольной формы, а также ограниченной кривыми Безье). Последовательно выделяемые области могут образовывать пересечения, объединения или вычитания;
- выделение связной области («волшебная палочка») с заданием параметров связности;
- перемещение, копирование, заливку выделенных областей;
- кадрирование (обрезку) изображения;
- изменение масштаба отображения на экране;
- вращение, масштабирование, искривление и зеркальное отображение изображения;
- ввод текста;
- выбор текущего цвета («пипетка»);
- заливку области сплошным цветом или градиентом;
- рисование «карандашом» или «кистью» произвольной формы и очистку «ластиком».

В базовую функциональность «ГИМП» входит также возможность захвата изображения со сканера и с экрана.

5.5 Фильтрация и синтез изображений

Основной прием автоматизированной обработки изображений — фильтрация их целиком, либо выделенных в них областей. Большая часть упомянутых внешних модулей реализует именно функцию фильтрации. Среди наиболее важных в практической обработке изображений фильтров отметим:

- изменение цвета, насыщенности, яркости и контраста изображения;
- удаление «шума»;
- повышение резкости и размывание, выделение краев.

Значительное количество фильтров имитирует различные «эффекты»: от просмотра изображения через волнистое стекло до натяжения его на сферу, цилиндр или тор.

Синтез изображений в базовой поставке «ГИМП» и известных нам фильтрах не слишком развит, однако есть ряд фильтров, создающих фрактальные изображения (в том числе, натуроморфные).

Литература

1. Министерство образования Российской Федерации. «Обязательный минимум содержания образования по информатике».

2. Максим Отставнов (ред.). «Свободное программное обеспечение: бизнес-модели и корпоративные инициативы» — М.: ГУ-ВШЭ, 2001. Расширенную версию этого сборника можно найти на: http://www.otstavnov.com/fsr

3. Эрик Рэймонд. «Собор и базар» // «Открытые системы», ## 09-10, 1999 г.

4. С.В. Сергеев, Н.А. Роганова. Практическая информатика: учебное пособие. Ч.1 — М.: МГИУ, 2001. См. тж. http://www.ctc.msiu.ru/materials/

5. «Информатика» # 29, 2002 г.

6. Документация на StarOffice 5.2 («Руководство пользователя» и «Инструкция по инсталляции», в совокупности около 600 стр.) входит в русское «коробочное» издание этого пакета. «Руководство пользователя» также (в значительно улучшенном переводе) издавала и включала в «коробочные» издания своих дистрибутивов, содержащих StarOffice, московская компания ASPLinux (www.asplinux.ru). Следует заметить, что «OO.o» 1.0 несколько отличается от StarOffice 5.2 и более ранних версий; однако единственным серьезным с точки зрения пользователя отличием является лишь отсутствие единого «рабочего стола» в первом.

7. Сергей Глушаков, Алексей Сурядный. Linux для дома и офиса: Учебный курс. — Харьков: «Фолио», 2002. — 389 с. Несмотря на название, бОльшая часть этой книги посвящена прикладным свободным пакетам, и в том, что касается «OO.o» и «Мозилла», ее содержание неспецифично для Linux (или каких-либо конкретных ОС вообще).

Базовой функциональности OW посвящено 110 страниц, ОС — 80, а «OpenDraw» уделили всего 7 страниц.

8. Антон Ионов и др. OpenOffice.ru. Руководство пользователя. — М.: ALT Linux, 2002. — 115 с. Двадцать пять страниц этой брошюры, не продающейся отдельно, но входящей в «коробочную» комплектацию ALT Linux 2.0 Master и OpenOffice.ru и доступной на сайте http://docs.openoffice.ru, посвящено OW, и это действительно кратчайшая инструкция по использованию. Двадцать пять страниц посвящено «OpenDraw», пятнадцать страниц -- «OpenCalc».

Для методических целей ценно то, что это свободная книга, изданная под лицензией GNU для свободной документации (GNU FDL) и, соответственно, она может использоваться (включая модификацию и распространение) без дополнительных разрешений или отчислений авторам (не забывайте о соблюдении неимущественных прав авторов и о том, что распространение модифицированной версии допускается лишь на усло-

виях той же лицензии).

9. С.В.Андреев, Н.А.Роганова. Практическая информатика: Учебное пособие. Часть 1. — М.: МГИУ, 2001. — 348 с. Это тоже свободная книга под GNU FDL, ее можно купить на бумаге или скачать отсюда: www.ctc.msiu.ru/materials/books.php. Пособие консервативно и, несмотря на обновление, пока описывает StarWriter 5.2, а не OW; впрочем, как уже отмечалось, разница между ними с точки зрения пользователя невелика.

10. Т.Робертс, «Текстовые редакторы» // «Человеческий фактор». Т. 6. — М.: «Мир», 1992

11. Дэвид Тейнсли «Linux и Unix: программирование в shell» — Киев: BHV, 2001

12. Ричард Столлмен. Руководство по GNU Emacs (тринадцатая редакция) — М.: Институт логики, 1999

13. Karin Kylander & Olof S Kylander. GIMP: The Official Handbook. The Coriolis Group: 1999, ISBN 1-57610-520-2

Карина и Олаф Киландер взяли на себя нелегкий труд официального документирования GIMP, созданного полутора сотнями программистов. Написанный ими «Официальный справочник по “ГИМП”» доступен (в разных форматах, включая готовые к печати postscript- и pdf-файлы) со страниц http://manual.gimp.org, он также издан на бумаге The Coriolis Group.

Остается надеяться, что рано или поздно эта книга появится и по-русски, а пока скачать это монументальное тысячестраничное издание рекомендуется всем, кто читает по-английски и использует или готовится использовать «ГИМП». Тем, кто пока не собирается — тоже, поскольку, помимо всего прочего, эта книга — одно из лучших введений в растровую компьютерную графику вообще.

14. Carey Bunks. Grokking the GIMP. — New Riders Publishing, 2000; ISBN: 0735709246; 352 pp.

Из другой литературы, посвященной «ГИМП», можно порекомендовать написанный Кэрием Банксом учебник Grokking the GIMP, также доступный в Сети на http://gimp-savvy.com/BOOK/.

15. Nikolai Bezroukov. The Orthodox File Manager (OFM) Paradigm. Work in progress // http://www.softpanorama.org/OFM/Ofm_00.shtml

16. D. Englebart, W. English. A Research Center for Augmenting Human Intellect // AFIPS Conf. Proc., Fall Joint Computer Conf. — San Francisco, 1968.

17. Фредерик Брукс. «Мифический человеко-месяц: двадцать лет спустя» // Его же. «Мифический человеко-месяц». — СПб.: «Символ-плюс», 1999

18. Виктор Вагнер. *True UNIX GUI* // http://www.ice.ru/~vitus/thoughts/true_unix_gui.txt/true_unix_gui.txt

19. «Домашний компьютер» #12, 2002 г. Материалы номера также доступны онлайн (http://www.homepc.ru/offline/2002/78/).

20. Матт Уэлш и др. «Запускаем Linux» — СПб.: «Символ», 2000.

21. Максим Отставнов. «Прикладные свободные программы в школе». — М.: «Медиа Технолоджи сервис», 2003.