ALT Linux Master 2.2

Руководство разработчика

P. Cederqvist, С. Индлин, Д. Левин, А. Махоткин,
В. Филиппов

ALT Linux Master 2.2: Руководство разработчика

P. Cederqvist, С. Индлин, Д. Левин, А. Махоткин, В. Филиппов

Часть I. ALT Packaging

Глава 1. *ALT* specfile conventions

Преобразование оригинального текста в *DocBook* : Ю. Зотов

Обоснование

При разработке этих правил решались следующие задачи:

Обеспечить желаемую функциональность: наши пакеты должны отвечать определённым правилам, о которых пойдёт речь несколько позже. Для этого надо, чтобы spec-файлы обеспечивали выполнение этих правил.

Помочь разработчику: так как spec-файлы все ещё пишут люди, то их работу нужно свести к тому минимуму, который, собственно, и требует участия человека. Разработчик не должен копировать блоки кода из файла в файл, ибо эта неинтеллектуальная работа отнимает массу сил и чревата ошибками. Для этого есть макросы. Если какой-то код появляется в разных spec-файлах более одного раза, то надо написать макрос(ы).

Spec-файлы

Устаревшие конструкции

Не следует использовать устаревшие конструкции — они лишь загромождают spec-файл, снижая тем самым его читабельность. К устаревшим конструкциям, в частности, относятся:

- тэг BuildRoot;
- строки вида `rm -rf $RPM_BUILD_ROOT`;
- %_defattr со стандартными аргументами в начале файлов и секций `%files`;
- секция `%clean`, пустая либо без разумного содержания.

Фигурные скобки

Нет смысла засорять текст spec-файла ненужными фигурными скобками. Избавится от них можно с помощью команды **cleanup_spec** *spec-файл*

Выравнивание

Используйте табуляции для выравнивания. Избегайте пробельных символов в конце строк.

Порядок тэгов

Рекомендуемый порядок заголовочных тэгов: Name, Version, Release, Serial, далее Summary, License, Group, Url, Packager, BuildArch, потом Source*, Patch*, далее PreReqs, Requires, Provides, Conflicts, и, наконец, Prefix, BuildPreReqs, BuildRequires. Разумеется, не все из вышеперечисленных тэгов используются, равно как встречаются и другие редко используемые тэги. В связи с тем, что BuildRequires зарезервирован для автоматически вычисляемых зависимостей, для указания особых зависимостей следует использовать BuildPreReq.

Значения тэгов

Значение тэга от его имени следует разделять одним пробелом. Элементы списка значений следует разделять запятой с последующим пробелом. Значение тэга Summary следует начинать с прописной буквы. Значение тэга Summary не следует завершать точкой. Значения тэгов Summary и %description могут содержать названия команд только в не изменённом виде.

Группы

Значение тэга Group должно соответствовать действительности и при этом принадлежать фиксированному множеству, перечисленному в файле `/usr/lib/rpm/GROUPS`.

ChangeLog

При формировании первой строки changelog-записи используйте утилиту **add_changelog** *spec-файл*. Описание изменений должно быть информативным; недостаточно объявить о наличии изменений, необходимо их все явно перечислить.

Файлы локализации

Если в состав пакета входят файлы локализации либо другие файлы на разных языках, следует использовать макрос %find_lang. Подробную информацию можно получить, выполнив команду **/usr/lib/rpm/find-lang -h**.

Внутрипакетные зависимости

При работе с мультипакетными spec-файлами соблюдайте правило внутрипакетных зависимостей: Если один пакет в какой-либо мере зависит от другого подпакета, то эта зависимость должна быть указана полностью, включая не только имя, но также версию, релиз и serial (если есть). Например, «`Requires: %name = %version-%release`» или «`Requires: %name = %serial:%version-%release`». Обратите внимание на синтаксис: знак равенства, в отличие от дефиса, окружён пробелами.

Разделяемые библиотеки

Пакеты, содержащие как разделяемые библиотеки, так и использующие их программы, должны быть разделены на подпакеты таким образом, чтобы в подпакет, содержащий разделяемые библиотеки, не входили использующие их программы. Это, в частности, позволяет уменьшить количество циклических зависимостей. По традиции, имена пакетов, состоящих только из разделяемых библиотек, должны начинаться с префикса `lib` либо содержать его внутри слова. При разделении подпакетов следует помнить о внутрипакетных зависимостях.

Каждый пакет, содержащий разделяемые библиотеки в каталоге `/lib`, `/usr/lib` или `/usr/X11R6/lib`, должен их регистрировать при установке/обновлениях и удалении с помощью макросов %post_ldconfig и %postun_ldconfig соответственно.

Статические библиотеки

Статические библиотеки должны паковаться в отдельные подпакеты, что связано со спецификой их использования. Если имя devel-подпакета заканчивается суффиксом `-devel`, то имя нового devel-static-подпакета будет заканчиваться суффиксом `-devel-static`. При разделении подпакетов следует помнить о внутрипакетных зависимостях: в списке зависимостей devel-static-подпакета должна присутствовать зависимость от `-devel` = %version-%release.

Переименование пакетов

Иногда пакеты переименовывают. Например, это случается при упаковке разделяемых библиотек. В таких случаях следует указывать правильную информацию о зависимостях, необходимую для корректного обновления. В частности, должен присутствовать:

- тэг `Provides: старое_имя = %version`;
- тэг `Obsoletes: старое_имя`.

Патчи

Наименование патчей

При создании новых патчей, а также при импортировании патчей из других источников необходимо придерживаться единых правил наименования имён патч-файлов: ***NAME*-*VERSION*-*ORIGIN*-*WHAT*.patch**, где

- ***NAME*** и ***VERSION*** — имя и версия пакета, для которого сделан патч;
- ***ORIGIN*** — аббревиатуры источников патча (обычно дистрибутивов);
- ***WHAT*** — краткое описание патча.

В случае, когда патч образован из нескольких частей, полученных из разных источников, компонента имени ***ORIGIN*** должна содержать аббревиатуры всех источников. Если патч был создан или адаптирован для *ALT Linux*, то в ***ORIGIN***, соответственно, должно присутствовать `-alt-`. Для патчей, созданных на базе CVS, компонента имени ***ORIGIN*** должна начинаться с `cvs-`***YYYYMMDD***.

При составлении описания патча следует иметь в виду следующие общепринятые сокращения:

`makefile`

патчи, затрагивающие исключительно `makefile*`;

`bound`

проверки на границы (буфера, целых чисел, и т.п.);

`config`

патчи, затрагивающие исключительно конфигурационные файлы;

`configuse`

патчи, затрагивающие исключительно `configure*`;

`doc`

патчи, затрагивающие исключительно документацию;

`fixes`

кумулятивные патчи и/или исправления по надёжности и/или безопасности;

`format`

патчи на использование форматирования строк (`printf`);

`install`

патчи, направленные на возможность выполнения **make install** непривилегированным пользователем;

`linux`

патчи, предназначенные для портирования ПО на Linux;

`man`

патчи, затрагивающие исключительно man-страницы;

`texinfo`

патчи, затрагивающие исключительно документацию в формате texinfo;

`tmp`

патчи, предназначенные для решения различных вопросов, связанных с временными файлами;

`vitmp`

патчи, направленные на поддержку **vitmp(1)**;

`warnings`

патчи, исправляющие ошибки, найденные компилятором.

Исходный код

Формат хранения

Исходный код большого объёма (как архивы, так и патчи, по статусу приравненные к ним) следует хранить в упакованном виде. Метод сжатия, `gzip` или `bzip2`, следует выбирать таким образом, чтобы размер архива был минимальным, например, с помощью утилиты **zme**. Исключение составляют nosurce-пакеты: в них отсутствующие файлы следует указывать в виде корректных URL.

Глава 2. *ALT Linux* RPM: особенности версии `rpm-4.0.4-alt5`

Преобразование оригинального текста в *DocBook* : Ю. Зотов

Обоснование

При разработке изменений и дополнений к RPM решались следующие задачи:

Обеспечить желаемую функциональность: наши пакеты должны отвечать определенным правилам, о которых пойдёт речь несколько позже. Для этого надо, чтобы spec-файлы обеспечивали выполнение этих правил.

Помочь разработчику: так как spec-файлы все ещё пишут люди, то их работу нужно свести к тому минимуму, который, собственно, и требует участия человека. Разработчик не должен копировать блоки кода из файла в файл, ибо эта неинтеллектуальная работа отнимает массу сил и чревата ошибками. Для этого есть макросы. Если какой-то код появляется в разных spec-файлах более одного раза, то надо написать макрос(ы).

Новые тэги

BuildHost

С помощью этого тэга можно переопределить значение hostname, которое RPM помещает в заголовок каждого пакета. По умолчанию, как и ранее, используется значение, возвращаемое **uname(2)**.

Устаревшие тэги

BuildRoot

Времена, когда тэг BuildRoot в spec-файле определял, какой каталогRPM будет использовать в качестве BuildRoot, прошли безвозвратно. Теперь этот тэг не несёт никакой информации и может (и должен) быть опущен. Вместо этого используется значение макроса %buildroot, который определён как «`%{_tmppath}/%{name}-buildroot`» в файле `/usr/lib/rpm/macros` и может быть переопределён в любом месте, где допускается определять макросы. В случае, если макрос

%buildroot не определён либо его значение представляет собой недопустимое значение «/», сборка пакета не будет выполнена.

Новые макросы

Встроенные макросы

%homedir

домашний каталог пользователя, вызывающего этот макрос;

%homedir{user}

домашний каталог пользователя `user`;

Макросы для часто используемых каталогов

manpages: %_man1dir, %_man2dir, %_man3dir, %_man4dir, %_man5dir, %_man6dir, %_man7dir, %_man8dir, %_man9dir;
X11R6: %_x11dir, %_x11bindir, %_x11libdir, %_x11includedir, %_x11mandir, %_x11datadir, %_x11fontsdir;
лицензии: %_licensedir;
меню: %_menudir, %_iconsdir, %_miconsdir, %_liconsdir;
emacs: %_emacslispdir;
tcl: %_tcllibdir, %_tcldatadir;
другие системные: %_initdir, %_lockdir, %_logdir, %_cachedir, %_spooldir.

Управление опциями компилятора gcc

%add_optflags *<options>*

добавить указанные параметры в стандартный набор %opflags;

%remove_optflags *<options>*

убрать указанные параметры из стандартного набора %opflags;

%optflags_core

базовые параметры;

%_optlevel

уровень оптимизации;

%optflags_optimization

параметры, отвечающие за оптимизацию, кроме архитектурно-зависимых;

%optflags_warnings

warning options;

%optflags_debug

debugging options;

%optflags_shared

параметры, применяемые для создания relocatable файлов;

%optflags_nocpp

параметры, отключающие поддержку C++ exceptions и C++ RTTI;

%optflags_notraceback

`-fomit-frame-pointer`;

%optflags_fastmath

`-ffast-math`;

%optflags_strict

`-fstrict-aliasing`;

%optflags_kernel

параметры, используемые при компиляции ядра и его модулей.

По умолчанию, стандартный набор %opflags состоит из «`%optflags_core %optflags_warnings %optflags_optimization`».

Макросы-надстройки над утилитой make

%__nprocs

число процессоров, доступных для сборки с помощью %make_build;

%make_build

вызов **make** с параметром, обеспечивающим оптимальную параллельную сборку в данной среде;

%make_install

вызов **make** с инициализацией переменной `INSTALL`, что в случае корректной реализации `Makefile`-ов пакета позволяет сохранить дату последней модификации файлов, что особенно важно для документации;

%makeinstall

«%make_install *<инициализация других переменных, используемых многими Makefile-ами>* install»

Регистрация разделяемых библиотек

%post_ldconfig, %post_ldconfig_lib

регистрация новых/обновлённых библиотек;

%post_ldconfig_sys

регистрация новых/обновлённых системных библиотек (которые могут быть использованы в chroot'ах);

%postun_ldconfig

отмена регистрации удалённых библиотек.

Регистрация документации в формате info

%install_info

регистрация новых/обновлённых info-страниц;

%uninstall_info

отмена регистрации удалённых info-страниц.

Регистрация меню

%update_menus

регистрация новых/обновлённых меню;

%clean_menus

отмена регистрации удалённых меню.

Регистрация каталогов scrollkeeper

%update_scrollkeeper

регистрация новых/обновлённых каталогов;

%clean_scrollkeeper

отмена регистрации удалённых каталогов.

Вспомогательные макросы %configure

%__libtoolize

путь к скрипту **libtoolize**;

%_configure_script

путь к скрипту **configure**;

%_configure_target

целевая платформа для **configure**;

%_configure_gettext

`--without-included-gettext`.

Серверные макросы

%post_service

регистрация нового сервиса при установке, перезапуск при обновлении;

%preun_service

отмена регистрации сервиса и его выключение при удалении.

Макросы, определяющие некоторые аспекты packaging policy

%buildroot

значение BuildRoot;

%_defattr

атрибуты файлов и каталогов по умолчанию для каждой секции `%files` и для каждого файла, включаемого в таких секциях;

%_cleanup_method

метод, используемый при удалении ненужных файлов в секции `%install`;

%_compress_method

метод, используемый при сжатии документации в секции `%install`;

%_findprov_default_method

метод, используемый по умолчанию при поиске предоставляемых зависимостей;

%_findreq_default_method

метод, используемый по умолчанию при поиске требуемых зависимостей;

%_fixup_method

метод, используемый при исправлении файлов в секции `%install`;

%_verify_elf_method

метод, используемый при проверке ELF-файлов в секции `%install`;

%_strip_method

метод, используемый при обработке ELF-файлов в секции `%install`;

%_{cleanup,compress,fixup,strip,verify_elf,findreq,findprov}_topdir

точка начала поиска файлов, обрабатываемых соответствующим методом;

%_{cleanup,compress,fixup,strip,verify_elf,findreq,findprov}_skiplist

список шаблонов файлов, которые будут пропущены при обработке соответствующим методом;

%set_{cleanup,compress,fixup,strip,verify_elf}_method

изменить значение соответствующего макроса;

%set_{cleanup,compress,fixup,strip,verify_elf,findreq,findprov}_{topdir,skipl

изменить значение соответствующего макроса;

%add_{cleanup,compress,fixup,strip,verify_elf,findreq,findprov}_skiplist

добавить значение в соответствующий список.

Вызов вспомогательных программ

%find_lang

вызов /**usr/lib/rpm/find-lang**

%strip_executable

вызов /**usr/lib/rpm/brp-strip** для обработки *ELF executables*;

%strip_relocatable

вызов /**usr/lib/rpm/brp-strip** для обработки *ELF relocatables*;

%strip_shared

вызов /**usr/lib/rpm/brp-strip** для обработки *ELF shared objects*;

%strip_static

вызов /**usr/lib/rpm/brp-strip** для обработки *ELF ar archives*;

%cleanup_build

вызов /**usr/lib/rpm/brp-cleanup**;

%compress_docs

вызов /**usr/lib/rpm/brp-compress**;

%strip_binaries

вызов /**usr/lib/rpm/brp-strip**;

%clean_buildroot

выполнение `rm -rf %buildroot`, если %buildroot не указывает на настоящий `/`.

Управление процессом сборки

%buildmulti

альтернативная директива %build для случая, когда в секции `%build` происходит заполнение %buildroot. Вообще говоря, такой техники стоит избегать во всех случаях, когда это возможно;

%_build_lang

значение переменных `LANG`, `LANGUAGE` и `LC_ALL`;

%_build_display

значение переменной `DISPLAY`;

%_build_xauthority

значение переменной `XAUTHORITY`;

%__ccache_cc

значение переменной `CCACHE_CC`;

%__ccache_dir

значение переменной `CCACHE_DIR`;

Версии некоторых установленных в системе пакетов

glibc: %__glibc_version, %__glibc_version_major, %__glibc_version_minor;
gcc: %__gcc_version, %__gcc_version_major, %__gcc_version_minor, %__gcc_version_base
python: %__python_version

%get_version

версия указанного пакета;

%get_release

релиз указанного пакета;

%get_serial

serial указанного пакета;

%add_serial

serial указанного пакета в виде, пригодном для включения в spec-файл;

%get_SVR

тройка значений `serial:version-release` указанного пакета;

%get_NSVR

четвёрка значений `name-serial:version-release` указанного пакета;

%get_dep

строка вида `name >= serial:version-release`, построенная по указанному пакету;

Эти макросы, как правило, используются в пакетах, сборка которых возможна с различными версиями этих программ, если эти версии правильно учитывать.

Управление процессом обработки spec-файлов

%def_with, %def_without, %def_enable, %def_disable

установка значения макросов условия с указанием значения по умолчанию;

%check_def

проверка макросов условия на непротиворечивость;

%subst_with, %subst_enable

подстановка значения макросов условия;

%defined, %undefined

проверка на существование макроса;

%with, %without, %enabled, %disabled

проверка значения макросов условия;

%ifdef, %ifndef

ветвление по факту существования макроса;

%if_with, %if_without, %if_enabled, %if_disabled

ветвление по значению макросов условия;

Прочие макросы

%intel

список архитектур Intel™, совместимых с `i386`;

%amd

список архитектур AMD™, совместимых с `i386`;

%ix86

список всех архитектур, совместимых с `i386`;

компоненты макроса %packager

%packagerName, %packagerAddress

%_internal_gpg_path

путь к связке ключей *ALT Linux Team*[1].

Новые параметры rpm

`-bE`

новый режим работы RPM, при котором происходит только подстановка макросов;

`--nowait-lock`

не блокировать процесс, если база данных RPM занята;

[1] `http://www.altlinux.ru`

`--fancypercent`

отображать дополнительную информацию о процентах проделанной работы при установке/обновлении пакетов;

`--nopatch`

не включать указанные патчи в исходный пакет;

`--nosource`

не включать указанные исходники в исходный пакет;

`--lastchange`

вывести информацию о последнем изменении пакета;

`--changes-since`

вывести информацию обо всех изменениях пакета, начиная с указанной версии.

Новые возможности rpm по сборке пакетов

По окончании выполнения секции `%install` RPM выполняет ряд действий:

- удаление ненужных файлов и каталогов;
- исправление прав доступа к файлам и каталогам;
- упаковка документации;
- удаление отладочной информации;
- коррекция символических ссылок на разделяемые библиотеки;
- перекомпиляция python-модулей.

Автоматическое удаление ненужных файлов

Все файлы и каталоги, подпадающие под правило определения ненужных файлов и каталогов, удаляются. В частности, по умолчанию подлежат удалению

- файлы с именами `DEADJOE`, `.SUMS`, `TAGS`, `core`;
- файлы, заканчивающиеся на `~`, `.orig`, `.rej`, `.bak`;

• каталоги с именем `CVS`.

Поддерживаются следующие методы определения файлов и каталогов, подлежащих удалению:

`none, skip`

поиска и удаления не производится;

`auto`

метод по умолчанию, определенный в файле **/usr/lib/rpm/brp-cleanup**;

`*`

специальный метод; переданное значение используется в качестве имени программы, которая будет вызвана для поиска и удаления ненужных файлов.

Какой метод будет использован в каждом конкретном случае, зависит от значения макроса %_cleanup_method; значение по умолчанию для этого макроса — `auto`.

Автоматический поиск и исправление конфигурационных файлов, используемых прежде всего при разработке ПО

Поддерживаются следующие типы файлов, подлежащих проверке и исправлению:

`none, skip`

поиска и проверки не производится;

`binconfig`

поиск и обработка shell-скриптов по шаблону `/usr/bin/*-config`;

`pkgconfig`

поиск и обработка файлов по шаблону `/usr/lib/pkgconfig/*.pc`;

`libtool`

поиск и обработка `.la`-файлов;

Какой метод будет использован в каждом конкретном случае, зависит от значения макроса %_fixup_method; значение по умолчанию для этого макроса — `binconfig pkgconfig libtool`.

Автоматическое исправление прав доступа к файлам и каталогам

Права доступа ко всем файловым объектам, находящимся в `$RPM_BUILD_ROOT`, проверяются и корректируются согласно следующим правилам:

- каталоги `/usr/share`, `/usr/include`, `/usr/X11R6/share`, `/usr/X11R6/include`, `/usr/X11R6/man` со всем содержимым должны быть доступны по чтению всем пользователям;
- ничто из содержимого каталога `/usr`, за исключением `/usr/src`, не должно быть доступно по записи не-владельцу, за исключением владельца файлов.
- никакие suid и/или sgid-файлы не должны быть доступны по чтению (и тем более по записи), за исключением владельца файлов.

Автоматическое сжатие man- и info-документации с поддержкой различных методов сжатия

Вся документация пакета, распознаваемая как man- или info-документация, по окончании работы секции `%install`, сжимается согласно выбранному методу. Поддерживаются следующие методы сжатия:

`bzip2`

сжатие с помощью **bzip2 -9**;

`gzip`

сжатие с помощью **gzip -9n**;

`auto`

сжатие с помощью **gzip -9n** либо **bzip2 -9** в зависимости от того, какой вариант окажется эффективнее;

`none`

производится декомпрессия файлов вместо сжатия;

`skip`

процедура сжатия пропускается полностью.

Какой метод будет использован в каждом конкретном случае, зависит от значения макроса %_compress_method; значение по умолчанию для этого макроса — `auto`. По окончании процедуры сжатия производится выравнивание ссылок, которые, возможно, требуют коррекции в связи с изменениями имён файлов в процессе их сжатия.

Автоматическая проверка ELF-файлов с поддержкой различных стратегий

Иногда в результате сборки пакета получаются ELF-файлы, содержащие неверную и/или недопустимую информацию в некоторых секциях, таких как `RPATH`. Поэтому по окончании работы секции `%install` проверяются все собранные ELF-файлы. Выбор типов файлов определяется значением макроса %_verify_elf_method, которое есть набор из следующих возможных значений:

`none, skip`

поиска и проверки не производится;

`relaxed`

проверка только на наличие недопустимых элементов в `RPATH`;

`normal`

`relaxed` + проверка на наличие более чем одного элемента в `RPATH`;

`strict`

проверка на наличие непустого `RPATH`.

Значение по умолчанию для макроса %_verify_elf_method в данный момент равно `normal`.

Автоматическое удаление отладочной информации из ELF-файлов с поддержкой различных стратегий выбора файлов, подлежащих обработке

Зачастую возможно уменьшить размер получаемых в результате сборки пакета ELF-файлов без потери качества за счёт удаления из них отладочной информации. Поэтому по окончании работы секции `%install` все ELF-файлы выбранных типов обрабатываются программой `strip`. Выбор типов файлов определяется значением макроса %_strip_method, которое есть набор из следующих возможных значений:

`executable`

ELF executable;

`relocatable`

ELF relocatable;

`shared`

ELF shared object;

`static`

ar archive.

Кроме того, есть возможность вызывать `strip` вручную, для этой цели предназначены макросы %strip_executable, %strip_relocatable, %strip_shared, %strip_static. Синтаксис этих макросов подробно изложен в **/usr/lib/rpm/brp-strip --help**.

Автоматическая перекомпиляция python-модулей

Как известно, python-модули обычно компилируют в байтовую форму для увеличения быстродействия при последующей работе с ними. Каждый такой модуль, помимо всего прочего, хранит время своего создания и полное имя файла, в котором должен находиться. В связи с последним обстоятельством скомпилированные модули, созданные в результате работы секции `%install`, непригодны, ибо не могут быть использованы после установки пакета. По этой причине теперь по окончании работы секции `%install` производится перекомпиляция всех

python-модулей таким образом, чтобы их можно было использовать после установки пакета. В качестве байт-компилятора будет использоваться программа, имя которой хранится в макросе %__python. Обычно это **/usr/bin/python**, однако в некоторых случаях может потребоваться изменить это значение на другое (например, в случае сборки пакета `python` или если по какой-то причине перекомпиляция не нужна).

Автоматический поиск требуемых и предоставляемых зависимостей

В дополнение к стандартному поиску зависимостей от/для разделяемых библиотек, реализована поддержка поиска требуемых зависимостей для shell- и perl-скриптов, поиска зависимостей, определяемых наличием специальных файлов в пакете, а также поддержка поиска предоставляемых зависимостей для perl-скриптов.

Изменение семантики тэгов, управляющих поиском зависимостей

Новые возможности RPM по автоматическому поиску зависимостей при сборке пакетов управляются, как и прежде, значениями тэгов AutoReq, AutoProv и AutoReqProv. К стандартным значениям `yes`/`no` (`true`/`false`), таким образом, добавлены новые возможные значения, являющиеся именами методов поиска зависимостей:

`lib/nolib`

включение/выключение поиска зависимостей от/для разделяемых библиотек;

`shell/noshell`

включение/выключение поиска зависимостей в shell-скриптах;

`perl/noperl`

включение/выключение поиска зависимостей в perl-скриптах;

`files/nofiles`

включение/выключение поиска зависимостей, определяемых наличием специальных файлов в пакете;

`default`

то же, что и `yes`;

`none, off`

то же, что и `no`;

`all`

включение всех возможных методов поиска зависимостей.

Значением тэга может являться как один метод, так и перечисление методов. По умолчанию, для каждого подпакета собираемого пакета AutoReq = AutoProv = `yes`, что на практике означает использование макросов %_findreq_default_method и %_findprov_default_method для определения методов поиска зависимостей.

Автоматическая очистка BuildRoot

Перед выполнением секции `%install` и по окончании выполнения секции `%clean` RPM автоматически очищает BuildRoot с помощью макроса %clean_buildroot. Это значит, что больше не нужно использовать эти ужасные **rm -rf $RPM_BUILD_ROOT**. Секция `%clean` вообще может (и должна) быть опущена, если в ней не содержится ничего, кроме этого «**rm**». В тех редких случаях, когда в spec-файле производится заполнение BuildRoot не в секции `%install`, как это должно быть, а в секции `%build`, что в принципе неправильно, можно перенести точку очистки BuildRoot из начала секции `%install` в начало секции `%build`, если заменить директиву `%build` на макрос `%buildmulti`.

Упрощение секции `%files`

Ранее в начале каждой секции `%files` было необходимо указывать атрибуты файлов и каталогов создаваемых пакетов с помощью довольно однообразно используемой директивы %defattr. Теперь это происходит автоматически в начале каждой секции `%files`, а также в начале каждого файла, включаемого в секцию `%files` с помощью опции `-f`. Точнее говоря, в качестве этой директивы используется значение макроса %_defattr. Таким образом, прежнее использование директивы %defattr в начале секций и файлов следует считать упразднённым.

Сборка пакетов привилегированным пользователем

То, что когда-то было необходимостью, со временем стало излишним, а порой и просто опасным. Теперь, когда все без исключения пакеты можно (и нужно) собирать непривилегированным пользователем во избежание риска разрушения системы и некорректной сборки, сборка пакетов привилегированным пользователем по умолчанию запрещена. Этот запрет можно снять путём изменения значения макроса %_allow_root_build.

Литература

[wwwrpm] Официальный web-сайт rpm: http://www.rpm.org/ .

[mailrpm] Список рассылки для разработчиков rpm: rpm-list{@}redhat.com .

[maxrpm] E. C. Bailey. February 17, 1997. Maximum RPM, (доступна также online-версия по адресу http://www.rpm.org/max-rpm/ и в формате PostScript по адресу http://www.rpm.org/local/maximum-rpm.ps.gz).

Глава 3. *ALT* Secure Packaging Policy

Преобразование оригинального текста в *DocBook* : Ю. Зотов

Массовые операции над файлами и каталогами (секции: `%setup`, `%build`, `%install`, `%pre*`, `%post*`, `%trigger*`)

При массовой обработке файлов и каталогов (glob expansion, find и др.) *НЕОБХОДИМО* отделять команду с параметрами от списка аргументов разделителем «--» везде, где это поддерживается.

Обоснование: Массовые операции над файлами, имена которых начинаются на «-», могут давать неверный результат в случае неиспользования «--».

При использовании утилиты **find** для изменения файлов и каталогов *НЕОБХОДИМО* использовать параметр `-print0`; соответствующие ему параметры других утилит:

xargs: -r0
grep: -Z
sort: -z

Обоснование: Использование **find** при работе с каталогами, содержащими объекты с нестандартными именами (пробелами и др.), без использования `-print0` приводит к неправильному результату.

Пример 3.1. Правильное использование **find**

```
find -type f -print0 |
   xargs -r0 %__grep -FZl 'mawk gawk' -- |
   xargs -r0 %__perl -pi -e 's/mawk gawk/gawk mawk/g' --
```

Операции с временными файлами

При необходимости создания временных файлов и/или каталогов следует использовать утилиту **mktemp(1)** совместно с командой **trap**, например:

```
TMPFILE="`mktemp -t somename.XXXXXXXXXX`" || exit 1
```

```
exit_handler()
{
    local rc=$?
    trap '' EXIT
    rm -f -- "$TMPFILE"
    exit $rc
}
trap exit_handler EXIT HUP INT PIPE TERM QUIT
```

Не следует пользоваться фиксированными либо предсказуемыми именами для создания временных файлов в общедоступных каталогах, таких как `/tmp`. Не следует оставлять временные файлы в случае успешного окончания текущей стадии сборки пакета.

Чужие и системные каталоги и файлы (секции: `%install`, `%files`)

Пакеты *НЕ ДОЛЖНЫ* включать в свой состав чужие каталоги и файлы, в частности, системные объекты файловой системы, а также файлы устройств (последнее — прерогатива пакета `dev`).

Обоснование: У каждого объекта файловой системы, имеющего отношение к дистрибутиву, должен и может быть только один владелец (или группа родственных владельцев в случае, когда несколько подпакетов одного пакета совместно используют общий каталог). Это лучше обеспечивает управление атрибутами объектов файловой системы, а также решает многие проблемы определения сборочных зависимостей между пакетами.

Атрибуты файлов и каталогов (секции: `%install`, `%files`)

Права доступа на привилегированные исполняемые файлы

Привилегированные исполняемые файлы, т.е. исполняемые файлы с установленными битами suid и/или sgid, *НЕ ДОЛЖНЫ* быть доступны по чтению (и тем более по записи) кому-либо, кроме процессов с `uid==0`.

Разделы, предназначенные для использования readonly

Пакеты *НЕ ДОЛЖНЫ* содержать файлы и каталоги в поддереве `/usr`, разрешающие доступ по записи кому-либо, кроме процессов с `uid==0`. Более того, программам, входящим в пакет, во время своей работы *НЕ СЛЕДУЕТ* полагаться на то, что файловые объекты, находящиеся в `/usr`, доступны по записи.

Файлы и каталоги, доступные для записи

Пакеты *НЕ ДОЛЖНЫ* содержать файлы и каталоги, доступные для записи всем пользователям. Должны быть предусмотрены меры по разграничению доступа, например, путём предоставления доступа по записи определённой группе(ам).

Владельцы файлов

Пакеты *НЕ ДОЛЖНЫ* содержать файлы, принадлежащие псевдо-пользователям, если в процессе работы к этим файлам осуществляется доступ процессов с другим uid либо с более широкими правами доступа. К таким файлам, в частности, относятся исполняемые, конфигурационные и неизменяемые файлы.

Обоснование: Псевдо-пользователь не должен иметь право изменять эти файлы; нарушение этого правила, как правило приводит к очевидной возможности осуществления pseudouser/root compromise.

Владельцы каталогов

Пакеты *НЕ ДОЛЖНЫ* содержать каталоги, принадлежащие псевдо-пользователям. Вместо этого следует использовать каталоги, принадлежащие root, с установленным sticky bit и доступом группы по записи.

Обоснование: Псевдо-пользователь не должен иметь право изменять атрибуты каталогов, а также файлы и каталоги, созданные другими пользователями; нарушение этого правила, как правило приводит к возможности осуществления pseudouser/root compromise.

Блокировки (секции: `%build`, `%install`, `%files`)

Разные пакеты, использующие блокировки для работы с общими файловыми объектами, такими как mbox'ы, во избежание потери данных *ДОЛЖНЫ* придерживаться единого механизма блокировки.

Например, для блокировки mbox'ов *НЕОБХОДИМО* использовать метод, за которым закрепилось имя *fcntl*. Не допускается использование привилегированных программ для dotlocking'а.

Обоснование: Каждая привилегированная программа — это дополнительная степень риска для системы, в которой такая программа установлена. Поэтому следует минимизировать потребность в подобных средствах. Метод блокировки fcntl опирается на системный вызов `fcntl(2)`, удовлетворяющий стандарту POSIX, и, следовательно, более широко распространённый, чем его аналог `flock(2)`.

Часть II. Работа с CVS

Глава 4. Обзор

Эта глава предназначена для людей, никогда ранее не использовавших CVS и, возможно, никогда не использовавших управление версиями.

Если вы уже знакомы с CVS и просто хотите изучить конкретную возможность или вспомнить определенную команду, вы, вероятно, можете пропустить всю главу.

Что такое CVS?

> Не помнящие прошлого обречены повторять его.
>
> —Джордж Сантаяна

CVS — это система контроля версий. Используя ее, вы можете вести историю ваших файлов с исходными текстами.

Например, иногда при определенном изменении в коде могут появиться ошибки, которые вы не сможете обнаружить в течение длительного времени. С помощью CVS вы легко можете обратиться к старым версиям, чтобы точно выяснить, что именно привело к ошибке. Иногда это сильно помогает.

Конечно, вы можете хранить каждую версию каждого файла, которые вы создаете. Это будет стоить вам невероятного объема дискового пространства. CVS хранит все версии файла в одном файле таким образом, что запоминаются лишь изменения между версиями.

CVS также поможет, если вы являетесь членом группы разработчиков одного проекта. Очень легко попортить чужие изменения, если только вы не крайне аккуратны. Некоторые редакторы, такие как GNU Emacs, стараются проследить, чтобы два человека не изменяли одновременно один и тот же файл. К сожалению, если кто-то использует другой редактор, эта предосторожность не сработает. CVS решает эту проблему, изолируя разработчиков друг от друга. Каждый работает в своем собственном каталоге, а затем CVS объединяет законченные работы.

CVS появился из набора sh-скриптов, автором которых был Dick Grune, опубликованных в группе новостей `comp.sources.unix` в томе 6 в декабре 1986 года. Несмотря на то, что ни строчки кода из тех скриптов не присутствует в CVS, основы алгоритма устранения конфликтов взяты именно оттуда.

В апреле 1989 года Brian Berliner спроектировал и реализовал CVS. Jeff Polk позднее помог ему с поддержкой модулей и ветвей поставщика.

Получить CVS можно разными способами, включая свободное получение в Интернете. За информацией о получении и по другим вопросам обращайтесь на:

```
http://www.cyclic.com/
http://www.loria.fr/~molli/cvs-index.html
```

Имеется список рассылки `info-cvs`, посвященный обсуждению CVS. Чтобы подписаться на него или отписаться, пишите на `info-cvs-request@gnu.org`.

Если вы предпочитаете группы новостей usenet, найдите `comp.software.config-mgmt`, посвященную обсуждению разнообразных систем управления конфигурацией, не только CVS. В будущем возможно создание `comp.software.config-mgmt.cvs`, если в `comp.software.config-mgmt` будет наличествовать достаточное количество обсуждений CVS.

Можно также подписаться на список рассылки `bug-cvs`. Чтобы подписаться, напишите на bug-cvs-request@gnu.org.

Чем не является CVS?

CVS сделает для вас множество вещей, но не пытается быть всем сразу.

CVS не является системой управления сборкой

Несмотря на то, что структуры вашего репозитория и файла модулей взаимодействуют с системой управления сборкой (то есть файлами `Makefile`), они принципиально независимы. CVS не указывает, как собирать тот или иной проект. Она просто хранит файлы, предоставляя возможность обращаться к ним, используя задуманную вами структуру дерева. CVS не указывает, как использовать дисковое пространство в извлеченных каталогах. Если вы создадите `Makefile` или скрипты в каждом каталоге так, что они должны знать относительную позицию всего остального, то дело кончится тем, что придется извлекать весь репозиторий. Если вы модуляризуете вашу работу и создадите систему сборки, которая будет совместно использовать файлы, (посредством ссылок, монтирования, `VPATH` в `Makefile`'ах и т. д.), то сможете использовать дисковое пространство любым угодным вам способом. Помните только, что любая подобная система требует серьезной работы по созданию и поддержанию. CVS не пытается справиться с

возникающими при этом вопросами. Конечно же, вам следует поместить средства, созданные для поддержки системы сборки (скрипты, `Makefile`'ы, и т. д.), под CVS. Выяснение того, какие файлы следует перекомпилировать при каком-либо изменении, опять же, не является задачей CVS. Традиционным подходом является использование make для сборки, и использование специальной утилиты для генерации зависимостей, используемых программой make.

CVS не является заменой руководителю

Предполагается, что вы общаетесь с вашим начальником и лидером проекта достаточно часто, чтобы знать о графике работ, точках слияния, именах веток и датах выпуска. Если это не так, что CVS никак не сможет помочь. CVS — это инструмент, заставляющий ваш код плясать под вашу дудку. Но вы и композитор, и исполнитель. Ни один инструмент не играет сам и не сочиняет собственной музыки.

CVS не является заменой общения разработчиков

Встретившись с конфликтом, состоящим из единственной строки, большинство разработчиков справляются с ними без особого труда. Однако, более общее определение конфликта включает в себя проблемы, которые слишком трудно решить без взаимодействия разработчиков. CVS не может обнаружить, что синхронные изменения в одном или нескольких файлах привели к логическому конфликту. Понятие конфликт, которое использует CVS, строго текстуально. Такие конфликты появляются, когда изменения в основном файле достаточно близки, чтобы напугать программу слияния (то есть diff3). CVS совершенно неспособна помочь в устранении нетекстуальных или распределенных конфликтов в логике программы. Пример: предположим, вы изменили аргументы функции `X`, описанной в файле `A`. В то же самое время кто-то еще редактирует файл `B`, добавив новый вызов функции `X`, используя старые аргументы. CVS ничем не сможет помочь. Возьмите привычку читать спецификации и беседовать с коллегами.

CVS не ведет контроля изменений

Под контролем изменений имеется в виду несколько вещей. Во-первых, это может означать отслеживание ошибок, то есть хранение базы данных обнаруженных ошибок и состояние каждой (исправлена ли она? в какой версии? согласился ли обнаруживший ее, что она исправлена?). О взаимодействии с внешней системой отслеживания ошибок можно прочитать в файлах `rcsinfo` и `verifymsg` . Другим аспектом контроля изменений является отслеживание того факта,

что изменения в нескольких файлах в действительности являются одним и тем же согласованным изменением. Если вы фиксируете несколько файлов одной командой **cvs commit**, то CVS забывает, что эти файлы были зафиксированы одновременно, и единственная вещь, их объединяющая — это одинаковые журнальные записи. В данном случае может помочь ведение файла `ChangeLog` в стиле GNU. Еще одним аспектом контроля изменений, в некоторых системах является возможность отслеживать статус каждого изменения. Некоторые изменения были написаны разработчиком, некоторые были изучены другим разработчиком, и так далее. Обычно при работе с CVS в этом случае создается diff файл, (используя команды **cvs diff** или **diff**), который посылается по электронной почте кому-нибудь, кто потом применит этот diff-файл, используя программу **patch**. Это очень гибко, но зависит от внешних по отношению к CVS механизмов, чтобы убедиться, что ничего не упущено.

CVS не является системой автоматического тестирования

Впрочем, имеется возможность принудительного выполнения серии тестов, используя файл `commitinfo`. Я, однако же, не очень много знаю о проектах, использовавших эту возможность, и есть ли в ней какие-нибудь ловушки и подводные камни.

CVS не имеет встроенной модели процесса

Некоторые системы обеспечивают способы убедиться, что изменения и релизы проходят через определенные ступени, получая одобрение на каждой. Вообще говоря, этого можно добиться с помощью CVS, но это может потребовать немного больше работы. В некоторых случаях вы будете использовать файлы `commitinfo`, `loginfo`, `rcsinfo` или `verifymsg`, чтобы убедиться, что предприняты определенные шаги, прежде чем CVS позволит зафиксировать изменение. Подумайте также, должны ли использоваться такие возможности, как ветви разработки и метки, чтобы, скажем, поработать над новой веткой разработки, а затем объединять определенные изменения со стабильной веткой, когда эти изменения одобрены.

Пример работы с CVS

В качестве введения в CVS мы приведем здесь типичную сессию работы с CVS. Первое, что необходимо понимать, это то, что CVS хранит все файлы в централизованном репозитории (см. Глава 5. *Репозиторий*); в этой главе предполагается, что репозиторий настроен.

Предположим, что вы работаете над простым компилятором. Исходный текст состоит из нескольких C-файлов и `Makefile`'а. Компилятор называется 'tc' (Тривиальный Компилятор), а репозиторий настроен так, что имеется модуль `tc`.

Получение исходного кода

Сначала вам надо получить рабочую копию исходного кода для `tc`. Используйте команду

```
$ cvs checkout tc
```

при этом будет создан каталог `tc`, в который будут помещены все файлы с исходными текстами.

```
$ cd tc
$ ls
CVS Makefile backend.c driver.c frontend.c parser.c
```

Каталог `CVS` используется для внутренних нужд CVS. Обычно вам не следует редактировать или удалять файлы, находящиеся в этом каталоге.

Вы запускаете свой любимый редактор, работаете над `backend.c` и через пару часов вы добавили фазу оптимизации в компилятор. Замечание для пользователей RCS и SCCS: не требуется блокировать файлы, которые вы желаете отредактировать.

Фиксирование изменений

После того, как вы проверили, что компилятор все еще компилируется, вы решили создать новую версию файла `backend.c`. При этом в репозитории появится ваш новый файл `backend.c`, который станет доступным всем, использующим этот репозиторий.

```
$ cvs commit backend.c
```

CVS запускает редактор, чтобы позволить вам ввести журнальную запись. Вы набираете «Добавлена фаза оптимизации», сохраняете временный файл и выходите из редактора.

Переменная окружения `$CVSEDITOR` определяет, какой именно редактор будет вызван. Если `$CVSEDITOR` не установлена, то используется

`$EDITOR`, если она, в свою очередь, установлена. Если обе переменные не установлены, используется редактор по умолчанию для вашей операционной системы, например, **vi** под UNIX или **notepad** для Windows 95/NT.

Вдобавок, CVS проверяет переменную окружения `VISUAL`. Существуют различные мнения о том, требуется ли такое поведение и должны ли дальнейшие версии CVS проверять переменную `VISUAL` или игнорировать её. В любом случае, лучше всего будет убедиться, что `VISUAL` или вообще не установлена, или установлена в то же значение, что и `EDITOR`.

Когда CVS запускает редактор, в шаблоне для ввода журнальной записи перечислены измененные файлы. Для клиента CVS этот список создается путём сравнения времени изменения файла с его временем изменения, когда он был получен или обновлен. Таким образом, если время изменения файла изменилось, а его содержимое осталось прежним, он будет считаться измененным. Проще всего в данном случае не обращать на это внимания — в процессе фиксирования изменений CVS определит, что содержимое файла не изменилось и поведет себя должным образом. Следующая команда **update** сообщит CVS, что файл не был изменен, и его время изменения будет возвращено в прежнее значение, так что этот файл не будет мешаться при дальнейших фиксированиях.

Если вы хотите избежать запуска редактора, укажите журнальную запись в командной строке, используя флаг `-m`, например:

```
$ cvs commit -m «Добавлена фаза оптимизации» backend.c
```

Уборка за собой

Перед тем, как перейти к другим занятиям, вы решаете удалить рабочую копию `tc`. Конечно же, это можно сделать так:

```
$ cd ..
$ rm -r tc
```

но лучшим способом будет использование команды **release**:

```
$ cd ..
$ cvs release -d tc
```

```
M driver.c
? tc
You have [1] altered files in this repository.
Are you sure you want to release (and delete) directory 'tc': n
** 'release' aborted by user choice.
```

Команда **release** проверяет, что все ваши изменения были зафиксированы. Если включено журналирование истории, то в файле истории появляется соответствующая пометка.

Если вы используете команду **release** с флагом `-d`, то она удаляет вашу рабочую копию.

В вышеприведенном примере команда release выдала несколько строк. `? tc` означает, что файл `tc` неизвестен CVS. Беспокоиться не о чем, `tc` — это исполняемый файл компилятора, и его не следует хранить в репозитории.

`M driver.c` — более серьезное сообщение. Оно означает, что файл `driver.c` был изменен с момента последнего получения из репозитория.

Команда **release** всегда сообщает, сколько измененных файлов находится в вашей рабочей копии исходных кодов, а затем спрашивает подтверждения перед удалением файлов или внесения пометки в файл истории.

Вы решаете перестраховаться и отвечаете n RET, когда **release** просит подтверждения.

Просмотр изменений

Вы не помните, что изменяли файл `driver.c`, поэтому хотите посмотреть, что именно случилось с ним.

```
$ cd tc
$ cvs diff driver.c
```

Эта команда сравнивает версию файла `driver.c`, находящейся в репозитории, с вашей рабочей копией. Когда вы рассматриваете изменения, вы вспоминаете, что добавили аргумент командной строки, разрешающий фазу оптимизации. Вы фиксируете это изменение и высвобождаете модуль.

```
$ cvs commit -m «Добавлена фаза оптимизации» driver.c
Checking in driver.c;
/usr/local/cvsroot/tc/driver.c,v <-- driver.c
```

```
new revision: 1.2; previous revision: 1.1
done
$ cd ..
$ cvs release -d tc
? tc
You have [0] altered files in this repository.
Are you sure you want to release (and delete) directory 'tc': y
```

Глава 5. Репозиторий

В репозитории CVS хранит полные копии всех файлов и каталогов, находящихся под контролем версий.

Обычно вам никогда не придется напрямую обращаться к файлам в репозитории. Вместо этого вы будете использовать команды CVS для получения вашей собственной копии файлов в вашем рабочем каталоге, а затем будете работать с этой копией. Когда вы внесли определенные изменения, вы помещаете (или фиксируете) их в репозиторий. Теперь в репозитории хранится информация о сделанных вами изменениях: что именно и когда было изменено и прочая подобная информация. Заметьте, что репозиторий не является подкаталогом рабочего каталога, и обратное также неверно; они находятся в совершенно разных местах.

CVS может обращаться к репозиторию множеством способов. Репозиторий может находиться на локальной машине, на соседней машине или же на машине, находящейся на другом континенте. Чтобы различать способы доступа к репозиторию, его имя начинается с метода доступа. Например, метод доступа *:local:* означает, что репозиторий находится в локальном каталоге. Например, `:local:/usr/local/cvsroot'` означает, что репозиторий находится в каталоге `/usr/local/cvsroot` на компьютере, на котором используется CVS. Другие методы доступа описаны в разделе “Сетевые репозитории”.

Если метод доступа не указан, и имя репозитория не содержит `:`, то предполагается метод *:local:*. Если в имени содержится `:`, то предполагается метод доступа *:ext:* или *:server:*. Например, если ваш локальный репозиторий находится в `/usr/local/cvsroot`, то вы можете использовать `/usr/local/cvsroot` вместо `:local:/usr/local/cvsroot`. Но если, например, под Windows NT ваш локальный репозиторий находится в `c:\src\cvsroot`, то вы должны указать метод доступа, то есть `:local:c:\src\cvsroot`.

Репозиторий делится на две части. `$CVSROOT/CVSROOT` содержит административные файлы CVS. Все прочие каталоги содержат модули, определенные пользователем.

Как сообщить CVS, где находится репозиторий

Существует несколько способов сообщить CVS, где искать репозиторий. Вы можете явно задать репозиторий в командной строке с помощью ключа `-d` («directory», каталог):

```
cvs -d /usr/local/cvsroot checkout yoyodyne/tc
```

Другим вариантом является установка переменной окружения `$CVSROOT` в полный путь до корня репозитория, например, `/usr/local/cvsroot`. Чтобы установить `$CVSROOT`, пользователи csh и tcsh должны поместить в свой файл `~/.cshrc` или `~/.tcshrc` такую строку:

```
setenv CVSROOT /usr/local/cvsroot
```

Пользователи sh и bash должны поместить в свой файл `.profile` или `.bashrc` такие строки

```
CVSROOT=/usr/local/cvsroot
export CVSROOT
```

Имя репозитория, указанное с помощью `-d`, будет использоваться вместо указанного в переменной окружения `$CVSROOT`. Когда вы извлечете рабочую копию из репозитория, эта копия будет помнить, из какого именно репозитория ее извлекли (эта информация хранится в файле `CVS/Root` в рабочем каталоге).

Ключ `-d` и файл `CVS/Root` переопределяют репозиторий, заданный в переменной окружения `$CVSROOT`. Если репозиторий, заданный ключом `-d`, отличается от репозитория, указанного в файле `CVS/Root`, используется первый из них. Конечно же, для правильного функционирования в обоих местах должен быть упомянут один и тот же репозиторий.

Как данные хранятся в репозитории

В большинстве случаев неважно, как именно CVS хранит информацию в репозитории. В действительности, формат уже менялся однажды и, скорее всего, изменится в будущем. Так как в большинстве случаев весь доступ к репозиторию происходит посредством команд CVS, такие изменения не приводят к каким-либо разрушениям.

Однако, в некоторых случаях необходимо знать, как именно CVS хранит данные в репозитории, например, если вы хотите следить за блокировками файлов, которые делает CVS или если вам потребуется изменить права доступа к файлам в репозитории.

Где хранятся файлы в репозитории

Общая структура репозитория — это дерево каталогов, соответствующее каталогам в рабочей копии. Предположим, например, что репозиторий находится в

```
/usr/local/cvsroot
```

Вот возможное дерево каталогов (показаны только каталоги):

```
/usr
|
+--local
| |
| +--cvsroot
| | |
| | +--CVSROOT
   |      (административные файлы)
   |
   +--gnu
   |   |
   |   +--diff
   |   |   (исходный текст GNU diff)
   |   |
   |   +--rcs
   |   |   (исходный текст RCS)
   |   |
   |   +--cvs
   |       (исходный текст CVS)
   |
   +--yoyodyne
       |
       +--tc
       | |
       | +--man
       | |
       | +--testing
       |
       +---(другое программное обеспечение фирмы Yoyodyne)
```

Внутри каталогов находятся файлы истории для каждого файла, находящегося под контролем версий. Имя файла истории состоит из

имени соответствующего файла и суффикса `,v`. Вот как выглядит дерево каталогов для `yoyodyne/tc`:

```
$CVSROOT
|
+--yoyodyne
| |
| +--tc
| | |
    +--Makefile,v
    +--backend.c,v
    +--driver.c,v
    +--frontend.c,v
    +--parser.c,v
    +--man
    |   |
    |   +--tc.1,v
    |
    +--testing
        |
        +--testpgm.t,v
        +--test2.t,v
```

Файл истории содержит, помимо всего прочего, достаточно информации, чтобы воссоздать любую ревизию файла, журнал всех зафиксированных изменений и имена всех пользователей, сделавших эти изменения. Файлы истории известны как RCS-файлы, потому что первой программой, которая создавала файлы этого формата, была система контроля версий RCS. Полное описание формата файлов находится на странице руководства rcsfile(5), распространяемого вместе с RCS, или в файле `doc/RCSFILES` из комплекта исходных текстов CVS. Этот формат файла используется повсеместно — множество других программ могут по меньшей мере импортировать файлы этого формата.

Файлы RCS, используемые в CVS, несколько отличаются от стандартного формата. Волшебные ветки — самое большое отличие; см. раздел “Волшебные номера веток”. Имена меток, которые позволяет использовать CVS, являются подмножеством тех, что позволены в RCS; см. раздел “Метки ревизий”.

Права доступа к файлам

Все файлы `,v` создаются с правами «только для чтения», и вам не следует изменять эти права доступа. Каталоги в репозитории должны быть доступны для записи тем, кому разрешено изменять файлы в каждом каталоге. Это обычно означает, что вам нужно создать группу пользователей UNIX (см. страницу руководства group(5)), состоящую из лиц, участвующих в создании проекта, и настроить репозиторий так, чтобы эта группа была владельцем каталога с проектом.

Это означает, что ограничивать доступ к файлам можно только на уровне каталога.

Заметьте, что пользователи должны иметь права на запись в каталог и для извлечения файлов, потому что CVS должна создать файлы блокировки.

Заметьте также, что пользователи должны иметь права на запись в файл `CVSROOT/val-tags`. CVS использует этот файл, чтобы отслеживать, какие метки разрешены (этот файл иногда обновляется, когда используются и когда создаются метки).

Каждый RCS-файл принадлежит пользователю, который последним зафиксировал изменения в этот файл. Этот факт не столь важен, главное — кто владелец каталога.

CVS пытается установить адекватные права доступа к файлам для новых каталогов, которые создаются в дереве, но если вам требуется, чтобы новый каталог имел права доступа, отличающиеся от его родительского каталога, вы должны задать это вручную. Если вы установите переменную окружения `CVSUMASK`, то она будет задавать, какие права доступа к файлам CVS использует при создании каталогов и/или файлов в репозитории. `CVSUMASK` не влияет на права доступа к файлам в рабочем каталоге; такие файлы имеют права, обычные для новых файлов, разве что только иногда CVS создает их с правами только для чтения.

Заметьте, что при использовании клиент-серверного CVS (см. раздел “Сетевые репозитории”) не существует нормального способа установить `CVSUMASK`; установка его на клиентской машине не играет роли. Если вы соединяетесь с помощью rsh, то можете устанавливать `CVSUMASK` в файле `.bashrc` или `.cshrc`, как описано в документации на вашу операционную систему. Это поведение может измениться в будущей версии CVS; не полагайтесь на то, что установка `CVSUMASK` на клиентской машине не играет роли.

При использовании сервера парольной аутентификации (*pserver*) обычно требуются гораздо более жесткие права доступа к каталогу

`$CVSROOT` и каталогам, находящимся в нём; см. раздел “Настройка сервера для парольной аутентификации”.

Некоторые операционные системы позволяют определенным программам выполнять операции, которые не может выполнять тот, кто вызывает эти программы. Таковы, например, возможности setuid или setgid в UNIX или установленные образы в VMS. CVS не разрабатывался, чтобы использовать такие возможности, и поэтому попытки установить CVS таким образом обеспечат защиту только лишь от случайных ошибок; те, кто желает обойти защиту, смогут это сделать и, в зависимости от конкретных условий, смогут получить доступ еще куда-либо помимо CVS. Вы можете попробовать использовать *pserver*. Эта возможность также способна создать ложное чувство безопасности или открыть дыру, большую чем та, которую вы пытаетесь закрыть, поэтому внимательно прочтите главу о безопасности сервера парольной аутентификации, если вы собираетесь его использовать. Дополнительная информация в разделе “Настройка сервера для парольной аутентификации”.

Специфические для Windows права доступа

Некоторые вопросы, связанные с правами доступа, специфичны для операционных систем класса Window (Windows 95/98, Windows NT и, скорее всего, будущие подобные операционные системы. Часть нижесказанного может быть применима к OS/2, хотя я не уверен).

Чердак

Вы заметите, что иногда CVS помещает RCS-файлы в каталоге `Attic` («чердак»). Например, если `CVSROOT` — это `/usr/local/cvsroot`, и мы говорим о файле `backend.c` в каталоге `yoyodyne/tc`, то обычно этот файл находится в

```
/usr/local/cvsroot/yoyodyne/tc/backend.c,v
```

Если же он попадает на чердак, то он будет находиться в

```
/usr/local/cvsroot/yoyodyne/tc/Attic/backend.c,v
```

С точки зрения пользователя неважно, находится файл на чердаке или нет, так как CVS сам следит за этим и при необходимости заглядывает на чердак в поисках файла. В случае же, если вы хотите

знать точно, то RCS-файл хранится на чердаке тогда и только тогда, когда головная ревизия ствола находится в состоянии dead (мертвое). «Мертвое» состояние означает, что файл был удален или же никогда не добавлялся в эту ветку. Например, если вы добавите файл в ветку, то его стволовая ревизия будет в «мертвом» состоянии, а ревизия на ветке — нет.

Каталог CVS в репозитории

Каталог `CVS` в каждом репозитории содержит информацию об атрибутах файлов (в файле `CVS/fileattr`); смотри `fileattr.h` среди исходных текстов CVS за дополнительной информацией. В будущем в этом каталоге могут оказать другие дополнительные файлы, поэтому сегодняшние реализации должны игнорировать неизвестные файлы.

Это поведение реализовано только в версиях 1.7 и выше.

Блокировки в репозитории

Эта глава ориентирована на людей, пишущих утилиты, обращающиеся к репозиторию CVS, не конфликтуя при этом с другими программами, обращающимися к тому же репозиторию. Если вы запутаетесь в описываемых здесь концепциях, как то блокировка чтения, блокировка записи и мертвая блокировка, то обратитесь к литературе по операционным системам или базам данных.

Файлы в репозитории, чьи имена начинаются с `#cvs.rfl` — это блокировки чтения. Файлы, чьи имена начинаются с `#cvs.wfl` — это блокировки записи. Старые версии CVS (до версии 1.5) создавали также файлы с именами, начинающимися с `#cvs.tfl`, но такие файлы здесь не обсуждаются. Каталог `#cvs.lock` служит основной блокировкой, то есть перед тем, как создавать какую-либо еще блокировку, сначала необходимо создать основную блокировку.

Чтобы создать блокировку чтения, сначала создайте каталог `#cvs.lock`. В большинстве операционных систем операция создания каталога является атомарной. Если попытка создания завершилась неудачно, значит, основная блокировка уже существует, поэтому подождите немного и попробуйте еще. После получения блокировки `#cvs.lock` создайте файл, чье имя состоит из `#cvs.rfl`, и информацией по вашему выбору, например, имя машины и номер процесса. Потом удалите каталог `#cvs.lock`, чтобы снять основную блокировку. Теперь можно читать репозиторий. Когда чтение окончено, удалите файл `#cvs.rfl`, чтобы снять блокировку чтения.

Чтобы получить блокировку записи, сначала создайте каталог `#cvs.lock`, как и в случае с блокировкой чтения. Затем убедитесь, что в репозитории нет файлов, чьи имена начинаются с `#cvs.rfl`. Если они имеются, удалите `#cvs.lock`, подождите немного и попробуйте снова. Если блокировок чтения нет, создайте файл с именем, состоящим из `#cvs.wfl` и какой-нибудь информации по вашему выбору, например, имени машины и номера процесса. Не удаляйте блокировку `#cvs.lock`. Теперь вы можете писать в репозиторий. Когда запись окончена, сначала удалите файл `#cvs.wfl`, а затем каталог `#cvs.lock`. Заметьте, что в отличие от файла `#cvs.rfl`, файл `#cvs.wfl` имеет чисто информационное значение; он не оказывает блокирующего эффекта, который в данном случае достигается использованием главной блокировки (`#cvs.lock`).

Заметьте, что каждая блокировка (чтения или записи) блокирует единственный каталог в репозитории, включая `Attic` и `CVS`, но не включая подкаталоги, которые представляют собой другие каталоги, находящиеся под контролем версий. Чтобы заблокировать целое дерево, вам следует заблокировать каждый каталог (заметьте, что если вы не сможете получить хотя бы одну блокировку в этом процессе, то следует отменить все уже полученные блокировки, затем подождать и попробовать снова, во избежание мертвых блокировок.)

Заметьте также, что CVS ожидает, что доступ к отдельным файлам `foo,v` контролируется блокировками записи. RCS использует в качестве блокировок файлы `,foo,`, но CVS не поддерживает такую схему, поэтому рекомендуется использование блокировки записи. Смотри комментарии к функции `rcs_internal_lockfile` в исходном коде CVS, где находится дополнительное обсуждение и мотивация.

Как в каталоге CVSROOT хранятся файлы

Каталог `$CVSROOT/CVSROOT` содержит различные административные файлы. В каком-то смысле этот каталог подобен любому другому каталогу в репозитории; он содержит RCS-файлы, чьи имена заканчиваются на `,v`, и многие команды CVS оперируют с ними обычным образом. Однако, имеется несколько различий.

Для каждого административного файла, в дополнение к RCS-файлу, хранится его последняя ревизия. Например, есть RCS-файл `loginfo,v` и файл `loginfo`, содержащий последнюю ревизию, находящуюся в `loginfo,v`. Когда вы фиксируете административный файл, CVS должен написать:

```
cvs commit: Rebuilding administrative file database
```

и обновить его извлеченную копию в `$CVSROOT/CVSROOT`. Если это не так, значит, что-то случилось с CVS . Чтобы ваши CVS обращался с вашими собственными файлами точно так же, вы можете добавить их имена в административный файл `checkoutlist`.

По умолчанию, файл `modules` ведет себя как описано выше. Если же он становится очень большим, то хранение в виде плоского файла может привести к медленному поиску модулей (я не уверен, что это все еще столь же важно, как и тогда, когда эта возможность впервые появилась; я не видел расчетов быстродействия). Таким образом, внеся определенные изменения в исходный код CVS, можно хранить файл модулей в базе данных, которая имеет интерфейс с ndbm, например, Berkeley db или GDBM. Если эта опция используется, то база данных модулей будет храниться в файлах `modules.db`, `modules.pag` и/или `modules.dir`.

Как данные хранятся в рабочем каталоге

Пока мы описываем внутреннюю работу CVS, которая иногда становится видна, мы можем также поговорить о том, что CVS хранит в каталогах `CVS` в рабочих каталогах. Как и в случае с репозиторием, CVS обрабатывает эту информацию, и обычно вы обращаетесь к ней посредством команд CVS. В некоторых случаях, однако, бывает полезно напрямую работать с содержимым этих каталогов, например, в графической оболочке jCVS или пакете VC для emacs. Такие программы должны следовать рекомендациям в этой главе, если они желают нормально работать совместно с другими программами, использующими те же самые файлы, включая будущие их версии, а также с CVS, работающим из командной строки.

Каталог `CVS` содержит несколько файлов. Программы, читающие этот каталог, должны игнорировать файлы, находящиеся в этом каталоге, но не документированные здесь, чтобы дать возможность развития в будущем.

Файлы хранятся в текстовом формате, соответствующем соглашениям операционной системы. Это означает, что рабочие каталоги не переносимы между системами с разными форматами хранения текстовых файлов. Это сделано специально, исходя из того, что сами файлы,

находящиеся под управлением CVS, вероятно, также не переносимы между такими платформами.

`Root`

Этот файл содержит текущий корневой каталог CVS, как описано в разделе “Как сообщить CVS, где находится репозиторий”.

`Repository`

Этот файл содержит каталог в репозитории, которому соответствует текущий каталог. Здесь может быть имя с полным или относительным путем; CVS способна обрабатывать оба варианта, начиная с версии 1.3. Относительный путь отсчитывается от корня, хотя использование абсолютного пути довольно распространено и программы должны уметь обрабатывать оба варианта. Например, после команды

```
cvs -d :local:/usr/local/cvsroot checkout yoyodyne/tc
Root будет содержать
:local:/usr/local/cvsroot
```

а `Repository` будет содержать или

```
/usr/local/cvsroot/yoyodyne/tc
```

или

```
yoyodyne/tc
```

Если рабочий каталог не имеет соответствующего каталога в репозитории, то `Repository` должен содержать `CVSROOT/Emptydir`.

`Entries`

В этом файле перечислены файлы и каталоги в рабочем каталоге. Первый символ каждой строки указывает тип каждой строки. Если символ нераспознан, программа, читающая файл, должна спокойно пропустить эту строку, чтобы дать возможность развития в будущем. Если первый символ — это `/`, то формат строки таков:

```
/имя/ревизия/метка времени[+конфликт]/опции/тэг или дата
```

где '[' и ']' не являются частью строки, но указывают, что '+' и отметка о конфликте не обязательны. name — это имя файла в каталоге. ревизия — это номер ревизии, на которой основан файл в рабочем каталоге, или '0' для добавленного файла, или '-', за которым следует номер ревизии, для удаленного файла. метка времени — это время, когда CVS создала этот файл; если это время отличается от текущего времени модификации файла, значит, он был изменен. Метка времени записывается в UTC (по Гринвичу), в формате, используемом функцией стандарта ISO C `asctime()` (например, 'Sun Apr 7 01:29:26 1996'). Можно написать также строку в другом формате, например, 'Result of merge', чтобы указать, что файл всегда должен считаться измененным. Эта строка — вовсе не специальный случай: чтобы узнать, изменился ли файл, CVS берет дату модификации файла и просто сравнивает строку со строкой метка времени. конфликт указывает, что произошел конфликт. Если эта строка совпадает с действительным временем модификации, значит, пользователь еще не справился с конфликтом. опции содержат прилипшие ключи командной строки (например, `-kb` для двоичных файлов). тэг или дата содержит либо 'T', за которой следует имя тэга, либо 'D', за которой следует прилипший тэг или дата. Заметьте, что если метка времени содержит пару меток времени, разделенных пробелом, а не единственную метку времени, значит, вы имеете дело с версией CVS ранее 1.5 (этот случай здесь не документирован). Если первый символ в строке в файле `Entries` — это 'D', это означает подкаталог. 'D' на отдельной строке указывает, что программа, которая создала файл `Entries`, умеет обращаться с подкаталогами (то есть, если такая строка присутствует, и нет других строк, начинающихся с 'D', значит, подкаталогов нет). В противном случае строка выглядит так:

```
D/имя/заполнитель1/заполнитель2/заполнитель3/заполнитель4
```

где имя — это имя подкаталога, а все поля заполнитель должны игнорироваться, в целях будущих расширений. Программы, изменяющие файлы `Entries`, должны сохранять значения этих полей. Строки в файле `Entries` могут быть в любом порядке.

`Entries.Log`

В этом файле хранится та же самая информация, что и в файле `Entries`, и с его помощью можно обновлять эту информацию без

необходимости полностью переписывать файл `Entries`, включая возможность сохранять информацию, даже если программа, писавшая в `Entries` и `Entries.Log` аварийно завершилась. Программы, читающие файл `Entries` должны также проверять существование файла `Entries.Log`. Если последний существует, то они должны прочесть файл `Entries` и внести в него изменения из файла `Entries.Log`, после чего рекомендуется записать заново файл `Entries` и удалить файл `Entries.Log`. Формат строки файла `Entries.Log` — односимвольная команда, за которой следует строка, в формате `Entries`. Команда — это либо 'A' для указания, что строка добавляется, либо 'R' — если строка удаляется, или любой другой символ — если эту строку следует проигнорировать (для будущих расширений). Если второй символ строки в файле `Entries.Log` — не пробел, значит, файл был создан старой версией CVS (здесь не документируется). Программы, которые пишут, но не читают, могут спокойно игнорировать `Entries.Log`.

`Entries.Backup`

Это временный файл. Рекомендованное использование — записать новый файл `Entries` в `Entries.Backup`, затем переименовать его (атомарно, если возможно) в `Entries`.

`Entries.Static`

Единственная вещь, интересующая нас об этом файле — существует он или нет. Если существует, это значит, что была получена только часть каталога и CVS не будет создавать в нем дополнительных файлов. Чтобы очистить этот файл, используйте команду update с опцией `-d`, чтобы получить дополнительные файлы и удалить `Entries.Static`.

`Tag`

В этом файле находятся прилипшие тэги и даты для этого каталога. Первый символ — 'T' для тэга ветки, 'N' для обычного тэга или 'D' для даты. Другие символы должны игнорироваться, для будущих расширений. За этим символом следует тэг или дата. Заметьте, что прилипшие тэги и даты применяются к добавляемым файлам; они могут отличаться от тэгов и дат, прилипших к отдельным файлам. Общая информация о прилипших тэгах и датах находится в разделе “Липкие метки”.

`Checkin.prog`

В этих файлах хранятся имена программ, заданных опциями `-i` и `-u` в файле `modules`, соответственно.

`Update.prog`

См. `Checkin.prog`.

`Notify`

В этом файле хранятся уведомления (например, для **edit** или **unedit**), которые еще не было отосланы на сервер. Их формат еще не документирован здесь.

`Notify.tmp`

Этот файл по отношению к файлу `Notify` является тем же, что `Entries.Backup` по отношению к `Entries`. Чтобы создать файл `Notify`, сначала запишите его новое содержимое в `Notify.tmp`, затем (атомарно, если возможно), переименуйте его в `Notify`.

`Base`

Если используются слежения, то команда **edit** сохраняет исходную копию файла в каталоге `Base`. Это позволяет команде **unedit** работать, даже если нет доступа к серверу.

`Baserev`

В этом файле перечислены ревизии каждого файла в каталоге `Base`. Формат таков:

`Вимя/ревизия/расширение`

поле расширение должно быть проигнорировано, для будущих расширений.

`Baserev.tmp`

Этот файл по отношению к `Baserev` является тем же, чем `Entries.Backup` по отношению к `Entries`. Чтобы создать записать файл `Baserev`, сначала запишите его новое содержимое в `Baserev.tmp`, затем (атомарно, если возможно), переименуйте его в `Baserev`.

`Template`

Этот файл содержит шаблон, заданный файлом `rcsinfo`. Он используется только клиентом; не-клиент-серверные варианты CVS напрямую обращаются к `rcsinfo`.

Административные файлы

Каталог `$CVSROOT/CVSROOT` содержит несколько административных файлов. Можно использовать CVS и без этих файлов, но некоторые команды лучше работают, если хотя бы файл `modules` должным образом настроен. В сущности, этот файл является наиболее важным, в нем описываются все модули в репозитории. Вот пример этого файла:

```
CVSROOT CVSROOT
modules CVSROOT modules
cvs gnu/cvs
rcs gnu/rcs
diff gnu/diff
tc yoyodyne/tc
```

Файл `modules` представляет собой текстовый файл. В простейшем случае каждая строка содержит имя модуля, пробел и имя каталога, где находится этот модуль, относительно `$CVSROOT`.

Строка, которая определяет модуль `modules`, использует возможности, здесь не описанные.

Редактирование административных файлов

Административные файлы можно редактировать точно так же, как и любой другой модуль. Используйте команду **cvs checkout CVSROOT**, чтобы получить рабочий каталог, редактируйте его и зафиксируйте изменения обычным образом.

Случается, что фиксируется административный файл с ошибкой. Обычно можно исправить ошибку и зафиксировать новую версию, но иногда особенно серьезная ошибка может привести к невозможности фиксирования изменений.

Несколько репозиториев

Иногда необходимо иметь много репозиториев, например, если у вас есть две группы разработчиков, работающих над разными проектами, у которых нет общего кода. Все, что вам требуется, чтобы работать с несколькими репозиториями — указать необходимый, используя переменную среды `CVSROOT`, опцию CVS `-d` или (если у вас уже есть рабочий каталог) просто работая по умолчанию с тем репозиторием,

из которого был извлечен рабочий каталог (См. раздел “Как сообщить CVS, где находится репозиторий”).

Серьезным преимуществом нескольких репозиториев является то, что они могут находиться на различных серверах. При использовании CVS версии 1.10 единственная команда может работать с каталогами из разных репозиториев. С помощью разрабатываемых версий CVS можно извлекать исходные тексты с нескольких серверов. CVS сам разберется с обходом дерева каталогов и соединениями с разными серверами при необходимости. Вот пример создания рабочего каталога:

```
cvs -d server1:/cvs co dir1
cd dir1
cvs -d server2:/root co sdir
cvs update
```

Команды **cvs co** создают рабочий каталог, а команда cvs update соединится с server2, чтобы обновить каталог `dir1/sdir`, и с server1, чтобы обновить все остальное.

Создание репозитория

Чтобы настроить CVS-репозиторий, сначала выберите машину и диск, на котором будет храниться история ревизий исходных текстов. Требования к процессору и памяти умеренны, поэтому подойдет практически любая машина. Детали описаны в разделе “Требования к серверу”.

Если вы импортируете RCS-файлы из другой системы, начальное дисковое пространство можно оценить как суммарный размер этих файлов. В дальнейшем можно рассчитывать на троекратный размер исходных текстов, которые вы будете хранить под контролем версий (когда-нибудь вы перерастете этот предел, но не слишком скоро). На машинах разработчики требуется дисковое пространство для рабочего каталога каждого разработчика (все дерево или его кусок, в зависимости от того, над чем работает программист).

К репозиторию должен быть доступ (прямой или с помощью сетевой файловой системы) со всех машин, которые будут использовать CVS в серверном или локальном режиме; клиентские машины не требуют никакого доступа к репозиторию кроме протокола CVS. Использование CVS для доступа только для чтения все равно требует прав на запись в репозиторий для создания файлов блокировок.

Чтобы создать репозиторий, выполните команду **cvs init**. Она создаст пустой репозиторий в корневом каталоге CVS, заданном обычным образом (см. раздел Глава 5. *Репозиторий*). Например,

```
cvs -d /usr/local/cvsroot init
```

cvs init следит, чтобы не перезаписать уже существующие файлы, поэтому никакого вреда от запуска **cvs init** по уже настроенному репозиторию не произойдет.

cvs init включит журналирование истории; если вы не хотите этого, удалите файл истории после выполнения **cvs init**.

Резервное копирование репозитория

Файлы в репозитории, в сущности, не обладают никакими особыми свойствами, в большинстве случаев можно делать их резервные копии как обычно. Есть, однако, несколько аспектов, которые необходимо учитывать.

Во-первых, с параноидальной точки зрения, следует либо не использовать CVS во время резервного копирования, либо сделать так, чтобы программа резервного копирования блокировала репозиторий в процессе. Чтобы не использовать CVS, вы можете запретить логины на машины, которые могут иметь доступ к репозиторию, отключить CVS-сервер или сделать что-либо подобное. Детали зависят от вашей операционной системы и от настройки CVS. Чтобы заблокировать CVS, создайте файлы блокировок (`#cvs.rfl`) в каждом каталоге репозитория. Даже учитывая вышесказанное, если вы просто скопируете файлы, ничего особенно страшного не произойдет. Однако, при восстановлении из резервной копии репозиторий может находиться в неустойчивом состоянии, что, впрочем, нетрудно исправить вручную.

Когда вы восстанавливаете репозиторий из резервной копии, предполагая, что репозиторий изменился с момента последнего резервного копирования, рабочие каталоги, которые не пострадали, могут ссылаться на ревизии, не существующие более в репозитории. Попытка выполнения CVS в таких каталогах приведет к сообщению об ошибке. Один из способов вернуть все изменения в репозиторий таков:

- Получите новый рабочий каталог.
- Скопируйте файлы из рабочего каталога, сделанного перед аварией, поверх файлов в новом рабочем каталоге (не копируйте содержимое каталогов `CVS`).

- Работая в новом рабочем каталоге, используйте команды типа **cvs update** и **cvs diff**, чтобы выяснить, что изменилось, а затем зафиксируйте изменения в репозиторий.

Перемещение репозитория

Точно так же, как и в случае с резервным копированием файлов, перемещение репозитория с места на место сводится к перемещению набора файлов.

Основная вещь, которую нужно учитывать — это то, что рабочие каталоги ссылаются на репозиторий. Самый простой способ справиться с этим — получить свежий рабочий каталог после перемещения. Конечно, вам следует сначала убедиться, что старый рабочий каталог был зафиксирован перед перемещением, или вы уверены, что не потеряете своих изменений. Если вы действительно хотите использовать уже существующий рабочий каталог, то это возможно с помощью хирургического вмешательства в файлы `CVS/Repository`. См. раздел "Как данные хранятся в рабочем каталоге", в котором описываются файлы `CVS/Repository` и `CVS/Root`, но если вы не уверены, то, наверное, лучше не пытаться.

Сетевые репозитории

Рабочая копия исходных текстов и репозиторий могут быть на разных машинах. Использование CVS таким образом известно как режим клиент/сервер. Вы выполняете CVS-клиент на машине, на которой смонтирован ваш рабочий каталог, и говорите ему общаться с машиной, на которой смонтирован репозиторий, с CVS-сервером. Вообще использование сетевого репозитория похоже на использование локального, только формат имени репозитория таков:

```
:метод:пользователь@машина:/путь/к/репозиторию
```

Детали зависят от того, как вы соединяетесь с сервером.

Если метод не указан, а имя репозитория содержит *:*, то метод по умолчанию — *ext* или *server*, в зависимости от платформы; оба метода описаны в разделе "Соединение с помощью rsh".

Требования к серверу

Простой ответ: требования к серверу умеренны — если дерево каталогов не очень большое, и активность не слишком высока, то подойдет машина с 32Mb памяти или даже меньше.

В реальной жизни, конечно, все сложнее. Оценка пикового использования памяти достаточна, чтобы оценить общие требования. Здесь документированы две такие области максимального потребления памяти; все остальные по сравнению с ними незначительны, чтобы мы обновили документацию.

Первая область большого потребления памяти — извлечения больших рабочих каталогов. Сервер состоит из двух процессов на каждого обслуживаемого клиента. Потребление памяти дочерним процессом должно быть невелико. Родительский процесс же, особенно когда сетевые соединения медленны, может вырасти до размеров, чуть больших размера исходных тестов, или до двух мегабайт, смотря что больше.

Умножая размер каждого CVS-сервера на количество клиентов, которые вы ожидаете одновременно, вы оцените требуемый размер памяти у сервера. По большей части память, потребляемая родительским процессом, будет находиться в файле подкачки, а не в физической памяти.

Вторая область большого потребления памяти — **diff** при фиксировании изменений в больших файлах. Это требуется даже для бинарных файлов. Можно предусмотреть использование примерно десятикратного размера самого большого файла, который только будет фиксироваться, хотя пятикратный размер будет вполне адекватен. Например, если вы хотите фиксировать файл размером в десять мегабайт, то в машине, на которой выполняется фиксирование (сервер или локальная машина, на которой находится репозиторий), должно быть сто мегабайт. Скорее всего, это будет файл подкачки, а не физическая память. Так как эта память требуется на непродолжительное время, то особенной нужды выделять память под несколько одновременных фиксирований нет.

Потребление ресурсов для клиентской машины еще более умеренны — любая машина, способная выполнять соответствующую операционную систему, будет пригодна.

Информация о требованиях к дисковому пространству находится в разделе “Создание репозитория”.

Соединение с помощью rsh

CVS использует протокол rsh для работы с сетевым репозиторием, поэтому на сетевой машине должен быть создан файл `.rhosts`, позволяющий доступ данному пользователю.

Например, предположим, что вы пользователь 'mozart' на локальной машине 'toe.example.com', а сервер находится на 'faun.example.com'. На машине 'faun' поместите в файл `.rhosts` в домашнем каталоге пользователя 'bach' следующее:

```
toe.example.com mozart
```

Потом протестируйте, что rsh работает, запустив

```
rsh -l bach faun.example.org 'echo $PATH'
```

Затем вам следует убедиться, что **rsh** найдет сервер. Убедитесь, что путь, напечатанный в результате выполнения этого примера содержит каталог, содержащий исполняемый файл **cvs**, который является серверной версией CVS. Вы можете установить путь в `.bashrc`, `.cshrc`, и т. п., но не в файлах `.login` или `.profile`. Можно также установить переменную среды `CVS_SERVER` на клиентской машине, чтобы указать, какой исполняемый файл вы хотите использовать, например, `/usr/local/bin/cvs-1.6`.

Не требуется редактировать `inetd.conf`, чтобы запустить CVS как демона.

Вы можете использовать в `CVSROOT` два метода доступа для **rsh**. *:server:* задает использование внутреннего клиента **rsh**, который поддерживается только в некоторых портах CVS. *:ext:* указывает внешнюю программу rsh. По умолчанию это rsh, но вы можете установить переменную среды `CVS_RSH`, чтобы выполнять другую программу, которая может соединиться с сервером (например, **remsh** на HP-UX 9, потому что **rsh** немного отличается. Эта программа должна уметь пересылать данные с сервера и на сервер, не изменяя их; например, rsh из Windows NT не подходит, потому что он транслирует CR-LF в LF. Порт CVS для OS/2 содержит хэк, который передает **rsh** параметр `-b`, чтобы обойти это, но поскольку это может привести к проблемам с программами, не являющимися стандартным rsh, это может быть изменено в будущем. Если вы устанавливаете `CVS_RSH` в ssh или какую-нибудь другую замену **rsh**, то инструкции по настройке `.rhosts`, скорее

всего, неприменимы, поэтому обратитесь к документации по соответствующей программе.

Продолжая наш пример, предположив, что вы хотите обратиться к модулю `foo` в репозитории `/usr/local/cvsroot` на машине 'faun.example.org', вы набираете:

```
cvs -d :ext:bach@faun.example.org:/usr/local/cvsroot checkout foo
```

(Можно не писать 'bach@', если имена пользователей совпадают на локальной и сетевой машинах.)

Прямое соединение с парольной аутентификацией

Клиент CVS также может соединяться с сервером, используя протокол с паролем. Это особенно полезно, когда использование rsh неосуществимо, (например, если сервер находится за файерволлом), и kerberos также недоступен.

Чтобы использовать этот метод, необходима некоторая настройка как сервера, так и клиентов.

Настройка сервера для парольной аутентификации

Во-первых, вы, вероятно, хотите усилить права доступа к каталогам `$CVSROOT` и `$CVSROOT/CVSROOT`. См. раздел "Прямое соединение с парольной аутентификацией".

На стороне сервера следует редактировать файл `/etc/inetd.conf`, чтобы inetd знал, что следует выполнять команду **cvs pserver**, когда кто-либо пытается соединиться с соответствующим портом. По умолчанию номер порта равен 2401; это значение можно изменить, если перед компиляцией установить параметр `CVS_AUTH_PORT` в другое значение.

Если ваш inetd позволяет использование номеров портов в `/etc/inetd.conf`, то можно использовать такую строку (строка отформатирована, чтобы влезла на страницу):

```
2401 stream tcp nowait root /usr/local/bin/cvs cvs -f
--allow-root=/usr/cvsroot pserver
```

Вы можете также использовать ключ командной строки `-T`, чтобы указать временный каталог.

Ключ командной строки `--allow-root` задает разрешенный каталог `CVSROOT`. Клиенты, пытающиеся использовать другой каталог, не смогут соединиться. Если вы хотите разрешить доступ к нескольким каталогам `CVSROOT`, повторите эту опцию.

Если ваш inetd требует текстовых имен сервисов вместо номеров портов, поместите эту строчку в `/etc/services`:

```
cvspserver 2401/tcp
```

и напишите cvspserver вместо 2401 в файле `/etc/inetd.conf`.

После всего этого перезапустите inetd или заставьте его перечитать файлы конфигурации.

Так как клиент хранит и пересылает пароли практически открытым тестом (см. раздел “Прямое соединение с парольной аутентификацией”), то может использоваться отдельный файл паролей для CVS, чтобы пользователи не раскрывали своих обычных паролей при доступе к репозиторию. Этот файл — `$CVSROOT/CVSROOT/passwd`. В этом файле используется обычный формат строк, разделенных двоеточиями, типа того, что используется в файле `/etc/passwd` в Unix-системах. В этом файле несколько полей: имя пользователя CVS, необязательный пароль и необязательное имя системного пользователя, на правах которого будет работать CVS после успешной аутентификации. Вот пример файла `passwd`, в котором находится пять строк:

```
anonymous:
bach:ULtgRLXo7NRxs
spwang:1sOp854gDF3DY
melissa:tGX1fS8sun6rY:pubcvs
qproj:XR4EZcEs0szik:pubcvs
```

(Пароли шифруются стандартной функцией UNIX `crypt()`, поэтому можно просто перенести пароль из обычного файла `/etc/passwd`.

Первая строка в этом примере предоставляет доступ любому CVS-клиенту, пытающемуся аутентифицироваться с именем anonymous и любым паролем, включая пустой пароль. (Это обычное решение для машин, предоставляющих анонимный доступ только для чтения; информация о предоставлении доступа только для чтения находится в разделе “Доступ к репозиторию только для чтения”.

Вторая и третья строки предоставляют доступ пользователям bach и spwang, если они знают соответствующий пароль.

Четвертая строка предоставляет доступ пользователю melissa, если она знает правильный пароль. При этом сама серверная программа CVS на самом деле выполняется на правах системного пользователя pubcvs. Таким образом, в системе не требуется заводить пользователя melissa, но обязательно должен быть пользователь pubcvs.

Пятая строка демонстрирует, что системные пользователи могут использоваться совместно: любой клиент, который успешно аутентифицируется как qproj, будет работать на правах системного пользователя pubcvs, так же, как и melissa. Таким образом, вы можете создать единственного общего системного пользователя для каждого проекта в вашем репозитории, и предоставить каждому разработчику свою собственную строку в файле `$CVSROOT/CVSROOT/passwd`. Имя CVS-пользователя в каждой строке будет разным, но имя системного пользователя будет одним и тем же. Причина, по которой нужно иметь разные имена пользователей CVS в том, что все действия CVS будут журналироваться под этими именами: когда melissa фиксирует изменения в проекте, эта фиксация записывается в историю проекта под именем melissa, а не pubcvs. Причина, по которой следует иметь одиночного системного пользователя в том, что вы сможете задать права доступа к соответствующим каталогам репозитория так, что только этот системный пользователь будет иметь права на запись.

Если в строке присутствует поле с системным пользователем, то все команды CVS выполняются на правах этого пользователя; если системное имя не задано, то CVS просто берет имя пользователя CVS в качестве имени системного пользователя, и работает на его правах. В любом случае, если в системе нет такого пользователя, то CVS-сервер откажется работать, даже если клиент сказал правильный пароль.

Пароль и имя системного пользователя могут отсутствовать (при отсутствии последнего не следует писать двоеточие, которое служит разделителем полей). Например, файл `$CVSROOT/CVSROOT/passwd` может выглядеть так:

```
anonymous::pubcvs
fish:rKa5jzULzmhOo:kfogel
sussman:1sOp854gDF3DY
```

Когда пароль пропущен или пустой, то аутентификация произойдет успешно с любым паролем, включая пустую строку. Однако, двоеточие после имени пользователя CVS всегда обязательно, даже если пароль пуст.

CVS также может использовать стандартную системную аутентификацию. При парольной аутентификации сервер сначала проверяет наличие пользователя в файле `$CVSROOT/CVSROOT/passwd`. Если пользователь обнаружен в этом файле, то соответствующая строка будет использована для аутентификации, как описано выше. Если же пользователь не найден, или файле `passwd` не существует, то сервер пытается аутентифицировать пользователя с помощью системных процедур (это "резервное" поведение может быть запрещено, установив SystemAuth=no в файле `config`. Помните, однако, что использование системной аутентификации может увеличить риск нарушения безопасности: операции CVS будут аутентифицироваться его обычным паролем, который будет передаваться по сети в текстовом виде. См. раздел “Вопросы безопасности при парольной аутентификации”, где описаны детали.

В настоящее время единственный способ поместить пароль в `CVSROOT/passwd` — это вырезать его откуда-нибудь еще. Когда-нибудь появится команда cvs passwd.

В отличие от большинства файлов в `$CVSROOT/CVSROOT`, обычно практикуется редактирование файла `passwd` прямо в репозитории, без использования CVS. Это из-за риска безопасности, связанного с извлечением этого файла в чью-нибудь рабочую копию. Если вы хотите, чтобы файл `passwd` извлекался вместе с остальными файлами в `$CVSROOT/CVSROOT`, см. раздел “Как в каталоге CVSROOT хранятся файлы”.

Использование клиента с парольной аутентификацией

Для того, чтобы выполнить команду CVS в сетевом репозитории с помощью сервера парольной аутентификации, нужно задать протокол pserver, имя пользователя, машину, на которой находится репозиторий, и путь к репозиторию. Например:

```
cvs -d :pserver:bach@faun.example.org:/usr/local/cvsroot checkout
someproj
```

или

```
CVSROOT=:pserver:bach@faun.example.org:/usr/local/cvsroot
cvs checkout someproj
```

Однако, если только вы не работаете с публичным репозиторием (то есть таким, где имя определенного пользователя не требует использования пароля), вам сначала потребуется войти в систему. При входе в систему проверяется ваш пароль. Это происходит при выполнении команды **login**, которая спрашивает у вас пароль:

```
cvs -d :pserver:bach@faun.example.org:/usr/local/cvsroot login
CVS password: _
```

После того, как вы ввели пароль, CVS проверяет этот пароль на сервере. Если результат положителен, то комбинация имени пользователя, машины, пути к репозиторию и пароля сохраняются в специальном файле, чтобы при дальнейшей работе с этим репозиторием от вас не требовалось запускать **cvs login**. (Если результат проверки отрицателен, CVS пожалуется, что пароль неверен, и, естественно, он не будет сохранен.)

Пароли обычно хранятся в файле `$HOME/.cvspass`. Этот файл можно прочитать глазами, и, до какой-то степени, можно отредактировать руками. Заметьте, впрочем, что пароли не хранятся в совсем открытом виде: они тривиально закодированы, чтобы защититься от нечаянного подсматривания (например, системным администратором или кем-либо другим, не настроенным враждебно).

Изменить место расположения этого файла можно, установив переменную окружения `CVS_PASSFILE`. При использовании этой переменной не забудьте установить её перед использованием **cvs login**. Если вы этого не сделаете, то последующие команды CVS не смогут найти паролей для отправки на сервер.

После того, как вы вошли в систему, все команды CVS, использующие этот сетевой репозиторий и имя пользователя, смогут аутентифицироваться, используя этот сохраненный пароль. Поэтому, например:

```
cvs -d :pserver:bach@faun.example.org:/usr/local/cvsroot checkout↩
foo
```

будет работать без дополнительных вопросов (если только пароль не изменится на сервере, в этому случае вам нужно ещё раз выполнить **cvs login**).

Заметьте, что если забыть про *:pserver:* в имени репозитория, то CVS будет считать, что вы собираетесь использовать rsh (см. раздел “Соединение с помощью rsh”).

Конечно же, после того, как вы извлекли рабочую копию, то можно не задавать имя репозитория при работе с ней, потому что CVS может и сама взять это имя из каталога `CVS/`.

Пароль к определенному сетевому репозиторию можно удалить из файла паролей с помощью команды **cvs logout**.

Вопросы безопасности при парольной аутентификации

Пароли хранятся на стороне клиента тривиально зашифрованным открытым текстом и передаются точно так же. Такое шифрование используется только для предотвращения нечаянного подсматривания пароля (например, системный администратор, случайно заглянувший в файл) и не предотвращает даже самые тривиальные атаки.

Отдельный файл паролей CVS (см. раздел “Настройка сервера для парольной аутентификации”) позволяет использовать для доступа к репозиторию пароль, отличающийся от пароля для доступа к машине. С другой стороны, если пользователь получил доступ к репозиторию для чтения и записи, он может различными способами выполнять программы на сервере. Таким образом, доступ к репозиторию означает также довольно широкий диапазон другого доступа к системе. Можно было бы модифицировать CVS, чтобы предотвратить это, но до сих пор никто этого не сделал. Более того, могут быть другие способы, которыми люди, имеющие доступ к репозиторию, получат доступ к системе; никто не производил тщательного аудита.

Заметьте, что из-за того, что каталог `$CVSROOT/CVSROOT` содержит `passwd` и прочие файлы, использующиеся в целях безопасности, нужно следить за правами доступа к этому каталогу так же хорошо, как из правами доступа к `/etc`. То же самое применимо к самому каталогу `$CVSROOT` и любому каталогу, находящему в нем. Кто угодно, получив доступ для записи в этот каталог, сможет стать любым пользователем в системе. Заметьте, что эти права доступа обычно строже при использовании *pserver*.

Вообще, любой, кто получает пароль, получает доступ к репозиторию, и, до некоторой степени, доступ к самой системе. Пароль доступен всем, кто может перехватить сетевые пакеты или прочитать защищенный (принадлежащий пользователю) файл. Если вы хотите настоящей безопасности, используйте Kerberos.

Прямое соединение с использованием GSSAPI

GSSAPI — это общий интерфейс к системам сетевой безопасности, таким как Kerberos 5.

Если у вас есть рабочая библиотека GSSAPI, то ваш CVS может совершать TCP-соединения с сервером, аутентифицируясь с помощью GSSAPI. Для этого CVS нужно скомпилировать с поддержкой GSSAPI; при конфигурировании CVS пытается определить, наличествуют ли в системе библиотеки GSSAPI, использующие Kerberos версии 5. Вы также можете дать `configure` флаг `--with-gssapi`.

Соединение аутентифицируется, используя GSSAPI, но сам поток данных не аутентифицируется по умолчанию. Вы должны использовать глобальный ключ командной строки `-a`, чтобы запросить аутентификацию потока.

Передаваемые данные по умолчанию не шифруются. Как сервер, так и клиент могут быть скомпилированы с поддержкой шифрования; используйте ключ командной строки **configure --enable-encrypt**. Для включения шифрования используйте ключ командной строки `-x`.

Соединения GSSAPI обрабатываются на стороне сервера тем же сервером, что производит парольную аутентификацию; смотри раздел “Настройка сервера для парольной аутентификации”. Если вы используете, например, Kerberos, обеспечивающий хорошую аутентификацию, вы, вероятно, захотите также устранить возможность аутентифицироваться с использованием паролей открытым текстом. Для этого создайте пустой файл `CVSROOT/passwd` и поместите `SystemAuth=no` в файл конфигурации `config`.

Сервер GSSAPI использует principal name cvs/имя-машины, где имя-машины — это каноническое имя сервера. Вам потребуется настроить ваш механизм GSSAPI.

Для соединения с использованием GSSAPI, используйте *:gserver:*. Например,

```
cvs -d :gserver:faun.example.org:/usr/local/cvsroot checkout foo
```

Прямое соединение с помощью Kerberos

Самый простой способ использования Kerberos — это kerberos rsh, что описано в разделе “Соединение с помощью rsh”. Основной недостаток использования rsh — тот, что все данные должны проходить

сквозь дополнительные программы, что замедляет работу. Поэтому если у вас установлен Kerberos, вам следует использовать прямые TCP-соединения, аутентифицируясь с помощью Kerberos.

Эта глава относится к системе Kerberos версии 4. Kerberos версии 5 поддерживается посредством общего интерфейса сетевой безопасности GSSAPI, как описано в предыдущей главе.

CVS должен быть скомпилирован с поддержкой kerberos; при конфигурировании CVS пытается определить, какая версия Kerberos присутствует на машине. Вы можете также использовать ключ командной строки **configure --with-krb4**.

Пересылаемые данные по умолчанию не шифруются. Как клиент, так и сервер должны быть скомпилированы с использованием шифрования; используйте ключ командной строки **configure --enable-encryption**. Для включения шифрования используйте глобальный ключ командной строки `-x`.

На сервере требуется отредактировать `/etc/inetd.conf`, чтобы запустить **cvs kserver**. Клиент по умолчанию использует порт 1999; если вы хотите использовать другой порт, задайте его на клиентской машине в переменной окружения `CVS_CLIENT_PORT`.

Когда вы захотите использовать CVS, сначала, как обычно, получите билет (kinit); этот билет должен позволять вам зарегистрироваться на сервере. Затем выполните

```
cvs -d :kserver:faun.example.org:/usr/local/cvsroot checkout foo
```

Предыдущие версии CVS могли в случае неудачи использовать соединение с помощью rsh; текущие версии так не делают.

Использование параллельного cvs server для соединения

Этот метод доступа позволяет вам соединяться с репозиторием, находящимся на локальном диске, используя сетевой протокол. Другими словами, он делает то же самое, что и *:local:*, но при этом с особенностями и ошибками, существующими у сетевого, а нее локального CVS.

Для каждодневных операций вы, скорее всего, предпочтете *:local:* или *:fork:*, в зависимости от ваших предпочтений. Конечно, *:fork:* особенно полезен при тестировании и отладке cvs и сетевого протокола. Точнее, мы избавляемся от необходимости настройки сети, таймаутов, проблем с аутентификацией, свойственных сетевому доступа, но при этом пользуемся собственно сетевым протоколом.

Чтобы соединиться, используя метод доступа *:fork:*, добавьте его к имени локального репозитория, например:

```
cvs -d :fork:/usr/local/cvsroot checkout foo
```

Как и при использовании *:ext:*, сервер по умолчанию называется **cvs**. Если установлена переменная окружения `CVS_SERVER`, используется ее значение.

Доступ к репозиторию только для чтения

Существует возможность предоставить публичный доступ к репозиторию только для чтения, используя сервер парольной аутентификации (см. раздел “Прямое соединение с парольной аутентификацией”). (Прочие методы доступа не имеют явной поддержки для доступа только для чтения, потому что все эти методы подразумевают регистрацию на машине с репозиторием, и поэтому пользователь может делать все, что позволяют ему права доступа к файлам).

Пользователь, имеющий доступ к репозиторию только для чтения, может выполнять все команды CVS, не изменяющие репозиторий, за исключением определенных "административных" файлов (таких, как файлы блокировок и файл истории). Может потребоваться использовать эту возможность совместно с возможностью использования псевдонимов пользователей (см. раздел “Настройка сервера для парольной аутентификации”).

В отличие от предыдущих версий CVS, пользователи с доступом только для чтения должны быть способны только читать репозиторий, но не выполнять программы на сервере или другим способом получать ненужные уровни доступа. Говоря точнее, закрыты все ранее известные дыры в безопасности. Так как эта возможность появилась недавно и не подвергалась исчерпывающему анализу безопасности, вы должны действовать с максимально необходимой осторожностью.

Есть два способа указать доступ пользователя только для чтения: включающий и исключающий.

Включающий способ означает, что пользователь явно указывается в файле `$CVSROOT/CVSROOT/readers`, в котором просто перечисляются "в столбик" пользователи. Вот пример:

```
melissa
```

```
splotnik
random
```

(Не забудьте символ новой строки в конце файла).

Исключающий способ означает, что все, кто имеет доступ к репозиторию для записи, перечисляются в файле `$CVSROOT/CVSROOT/writers`. Если этот файл существует, то все пользователи, не упомянутые в нем, получают доступ только для чтения (конечно, даже пользователи только для чтения должны быть упомянуты в файле `CVSROOT/passwd`). Файл `writers` имеет тот же формат, что и файл `readers`.

Замечание: если ваш файл `CVSROOT/passwd` отображает пользователей CVS в системных пользователей (см. раздел “Настройка сервера для парольной аутентификации”), убедитесь, что вы предоставляете или не предоставляете доступ только для чтения пользователям CVS, а не системным пользователям. Это означает, что в файлах `readers` и `writers` должны находиться пользователи CVS, которые могут не совпадать с системными пользователями.

Вот полное описание поведения сервера, принимающему решение, какой тип доступа предоставить:

- Если файл `readers` существует, и данный пользователь не упомянут в нем, он получает доступ только для чтения.
- Если существует файл `writers`, и этот пользователь НЕ упомянут в нем, то он также получает доступ только для чтения (это так даже если файл `readers` существует, но пользователь не упомянут в нем).
- В противном случае пользователь получает полный доступ для чтения и записи.

Конечно, возможен конфликт, если пользователь упомянут в обоих файлах. Такой конфликт разрешается консервативно и такой пользователь получает доступ только для чтения.

Временные каталоги на сервере

В процессе работы CVS-сервер создает временные каталоги. Они называются `cvs-servpid`, где `pid` — это номер процесса сервера. Они находятся в каталоге, указанном в переменной окружения `TMPDIR`, ключом командной строки `-T` или в `/tmp` по умолчанию.

В большинстве случаев сервер сам удалит временный каталог в конце работы. В некоторых случаях сервер может завершиться, не удалив свой временный каталог, например:

- если сервер аварийно завершается из-за внутренней ошибки, он может оставить временный каталог, чтобы облегчить отладку;
- если сервер был убит так, что не смог убрать за собой (например, **kill -KILL** под UNIX);
- система прекращает свою работу, не сообщив предварительно серверу об этом факте.

В таких случаях вы должны вручную удалить каталоги `cvs-servpid`. Если нет сервера с номером процесса pid, то сделать это можно совершенно безопасно.

Глава 6. Начинаем проект под CVS

Так как переименование файлов и перемещение их между каталогами слегка неудобно, первое, что вам следует сделать, когда вы начинаете новый проект — продумать организацию файлов. Собственно, перемещать и переименовывать файлы можно, но это, во-первых, увеличивает возможность недопонимания, а во-вторых, у CVS есть некоторые неполадки, например, при переименовании каталогов. См. раздел “Перемещение и переименование файлов”.

Дальнейшие действия зависят от конкретной ситуации.

Помещение файлов в репозиторий

Первым шагом будет создание файлов в репозитории. Это может быть сделано несколькими различными способами.

Создание дерева каталогов из нескольких файлов

Когда вы начнете использовать CVS, вы, скорее всего, уже имеете несколько проектов, которые можно поместить под контроль CVS. В этих случаях самым простым методом будет использование команды import. Самым простым объяснением, вероятно, будет привести пример. Если файлы, которые вы хотите поместить под CVS, находятся в `wdir`', а вы хотите, чтобы они появились в репозитории в каталоге `$CVSROOT/yoyodyne/rdir`, вы можете сказать:

```
$ cd wdir
$ cvs import -m "Imported sources" yoyodyne/rdir yoyo start
```

Если вы не укажете журнальное сообщение с помощью ключа командной строки `-m`, то CVS запустит редактор, в котором можно будет набрать это сообщение. Строка 'yoyo' — это тэг производителя, а 'start' — это тэг релиза. В данном контексте они могут не иметь назначения, но CVS требует их присутствия.

Теперь вы можете проверить, что все работает и удалить ваш исходный каталог.

```
$ cd ..
$ mv dir dir.orig
$ cvs checkout yoyodyne/dir # объяснение следует
$ diff -r dir.orig yoyodyne/dir
$ rm -r dir.orig
```

Было бы неплохо удалить изначальные файлы, чтобы случайно не начать редактировать их в `dir` без использования CVS. Конечно же, перед удалением хорошо было бы убедиться, что у вас есть резервная копия исходных текстов.

Команда **checkout** получает в качестве аргумента имя модуля (как в предыдущих примерах) или имя каталога относительно `$CVSROOT`, как в вышеприведенном примере.

Хорошо было бы проверить, что права доступа на созданные CVS каталоги правильны, и что эти каталоги принадлежат должным группам. См. раздел “Права доступа к файлам”.

Если какие-то из файлов, которые нужно импортировать, являются бинарными, вам потребуется использовать обертки, чтобы указать, какие именно.

Создание файлов из других систем контроля версий

Если у вас есть проект, который обслуживается другой системой контроля версий, например, RCS, вы можете захотеть поместить эти файлы под управление CVS и сохранить историю изменения этих файлов.

Из RCS

Если вы использовали RCS, то найдите все файлы RCS — обычно файлу `foo.c` будет соответствовать файл `RCS/foo.c,v` (этот файл может также находиться в другом месте, обратитесь к документации на RCS. Затем создайте соответствующие каталоги в CVS, если они еще не существуют. Затем скопируйте файл в соответствующие каталоги в репозитории (имя в репозитории должно совпадать с именем исходного файла с добавлением суффикса ’,v’; файлы находятся прямо в соответствующем каталоге репозитория, а не в подкаталоге `RCS/`. Это — один из редких случаев, когда желателен прямой доступ к репозиторию, без использования команд CVS. Теперь вы можете извлечь новый рабочий каталог. Файл RCS не должен быть заблокирован, когда вы

перемещаете его под управление CVS, иначе у CVS будут проблемы при работе с этим файлом.

Из другой системы контроля версий

Многие системы контроля версий способны экспортировать файлы RCS в стандартном формате. Если ваша система умеет так делать, экспортируйте файлы RCS и следуйте вышеприведенным инструкциям. Если это не так, вероятно, лучшим выходом будет написать скрипт, который извлекает каждую ревизию файла, используя интерфейс командной строки старой системы, а затем фиксирующий эти ревизии в CVS. Скрипт `sccs2rcs`, упомянутый ниже, является хорошим примером.

Из SCCS

В каталоге 'contrib/' среди исходных текстов CVS есть скрипт `sccs2rcs`, конвертирующий файлы SCCS в файлы RCS. Замечание: вы должны выполнить этот скрипт на машине, на которой установлен как SCCS, так и RCS, и этот скрипт не поддерживается.

Из PVCS

В каталоге `contrib/` среди исходных текстов CVS есть скрипт `pvcs_to_rcs`, преобразующий архивы PVCS в файлы RCS. Вы должны выполнить этот скрипт на машине, на которой установлены как PVCS, так и RCS, и как и все прочее в каталоге `contrib/`, этот скрипт не поддерживается. Детали описаны в комментариях к скрипту.

Создание дерева каталогов с нуля

Для нового проекта самым простым способом, вероятно, будет создать пустую структуру каталогов, например:

```
$ mkdir tc
$ mkdir tc/man
$ mkdir tc/testing
```

Затем используйте команду **import**, чтобы создать соответствующую (пустую) структуру каталогов внутри репозитория:

```
$ cd tc
$ cvs import -m "Created directory structure" yoyodyne/dir yoyo start
```

Затем используйте команду **add**, чтобы добавлять файлы и новые каталог по мере их появления.

Убедитесь, что права доступа, которые CVS дает новым каталогам в `$CVSROOT`, правильны.

Определение модуля

Следующим шагом будет определение модуля в файле `modules`. Это необязательно, но модули удобны для группирования связанных файлов и каталогов.

В простых случаях нижеследующих шагов достаточно для определения модуля.

1. извлеките рабочую копию файла 'modules':

```
$ cvs checkout CVSROOT/modules
$ cd CVSROOT
```

2. отредактируйте этот файл, вставив в него строку, определяющую модуль. См. раздел ???. Полное описание файла `modules`' можно найти в разделе ???. Например, для описания модуля `tc` можно использовать такую строку:

```
tc yoyodyne/tc
```

3. зафиксируйте ваши изменения в файле `modules`

```
$ cvs commit -m "Added the tc module." modules
```

4. освободите модуль `CVSROOT`

```
$ cd ..
$ cvs release -d CVSROOT
```

Глава 7. Ревизии

В большинстве случаев использования CVS не требуется сильно беспокоиться о номерах ревизий; CVS присваивает номера типа 1.1, 1.2 и т. д., и этого достаточно. Некоторые, однако, хотели бы иметь больше информации и лучше контролировать то, как CVS присваивает номера ревизий.

Если необходимо отслеживать набор ревизий, содержащих более одного файла, например, ревизии, попавшие в конкретную версию программы, используются метки, т. е. буквенные имена ревизий, которые можно присвоить каждому номеру ревизии файла.

Номера ревизий

Каждая ревизия файла имеет уникальный номер ревизии. Номера ревизий выглядят как '1.1', '1.2', '1.3.2.2' или даже '1.3.2.2.4.5'. Номер ревизии всегда содержит четное количество десятичных чисел, разделенных точкой. По умолчанию ревизия 1.1 — первая ревизия файла. В номере каждой следующей ревизии самая правая цифра увеличивается на единицу. Вот пример нескольких ревизий, новые версии находятся правее старых:

```
+-----+    +-----+    +-----+    +-----+    +-----+
! 1.1 !----! 1.2 !----! 1.3 !----! 1.4 !----! 1.5 !
+-----+    +-----+    +-----+    +-----+    +-----+
```

Может также оказаться, что в номерах ревизий будет больше одной точки, например, '1.3.2.2'. Такие номера означают ревизии, находящиеся наветках (см. раздел Глава 8. *Создание и слияние ветвей*); эти номера подробно описаныв в разделе “Ветки и ревизии”.

Версии и ревизии

Как описано выше, у файла может быть несколько ревизий. У программного продукта может быть несколько версий. Программным продуктам обычно дают номера версий типа '4.1.1'.

Назначение номеров ревизий

По умолчанию, CVS назначает номер ревизии, оставляя первую цифру и увеличивая вторую. Например, 1.1, 1.2, 1.3.

При добавлении нового файла вторая цифра всегда будет единицей, а первая цифра будет равняться самой большой первой цифре номера ревизии каждого файла в каталоге. Например, если в каталоге находятся файлы с ревизиями 1.7, 3.1, 4.12, то добавленный файл получит номер ревизии 4.1.

Обычно совершенно не требуется заботиться о номерах ревизий — проще думать о них, как о служебных номерах, за которыми следит CVS, а также о метках, обеспечивающих хороший способ различать, например, версию 1 вашего продукта от версии 2 (см. раздел "Метки ревизий"). Однако, если вы хотите установить номер ревизии, вам поможет ключ командной строки `-r` команды **cvs commit**. Ключ `-r` подразумевает использование ключа `-f`, в том смысле, что он приводит к фиксированию файлов, даже если он не были изменены.

Например, для того, что задать всем вашим файлам, включая те, что не изменились, номер ревизии 3.0, выполните команду

```
$ cvs commit -r 3.0
```

Заметьте, что номер, который вы указываете вместе с ключом `-r`, должен быть больше любого существующего номера ревизии. Скажем, если существует ревизия 3.0, вы не можете сказать **cvs commit -r 1.3**. Если вы хотите параллельно отслеживать несколько версий программного продукта, вам нужно создать ветку (см. раздел Глава 8. *Создание и слияние ветвей*).

Метки ревизий

Номера ревизий живут своей собственной жизнью. Они могут совершенно никак не соотноситься с номером версии вашего программного продукта. В зависимости от того, как вы используете CVS, номера ревизий могут измениться несколько раз между двумя выпусками продукта. Например, файлы с исходными текстами RCS 5.6 имеют такие номера ревизий:

```
ci.c 5.21
co.c 5.9
ident.c 5.3
rcs.c 5.12
rcsbase.h 5.11
rcsdiff.c 5.10
rcsedit.c 5.11
```

```
rcsfcmp.c 5.9
rcsgen.c 5.10
rcslex.c 5.11
rcsmap.c 5.2
rcsutil.c 5.10
```

Вы можете использовать команду **tag**, чтобы задать буквенное имя определенной ревизии файла. Вы можете использовать ключ командной строки `-v` команды **status**, чтобы увидеть все метки, которые имеет файл, а также какие номера ревизий они представляют. Имена меток должны начинаться с буквы и могут содержать буквы, цифры и знаки '-' и '_'. Два имени меток *BASE* и *HEAD* зарезервированы для использования в CVS. Предполагается, что будущие зарезервированные имена будут иметь специальный вид, например, начинаться с символа '.', чтобы избежать конфликтов с действительными именами меток.

Вы захотите выбрать какое-либо соглашение об именах меток, основываясь, например, на имени программы и номере ее версии. Например, можно взять имя программы, за которым следует номер версии, в котором символ '.' заменен на '-', так что CVS 1.9 будет помечен как cvs1-9. Если вы выберете стабильные правила именования, вам не придется постоянно угадывать, называется ли метка cvs-1-9, cvs1_9 или как-то еще. Вы можете даже принудительно задать эти правила именования в файле `taginfo`.

В нижеследующем примере показано, как добавить метку к файлу. Команды должны выполняться внутри вашего рабочего каталога, то есть там, где находится файл `backend.c`.

```
$ cvs tag rel-0-4 backend.c
T backend.c
$ cvs status -v backend.c
===================================================================
File: backend.c          Status: Up-to-date

   Version:              1.4 Tue Dec 1 14:39:01 1992
   RCS Version:          1.4 /u/cvsroot/yoyodyne/tc/backend.c,v
   Sticky Tag:           (none)
   Sticky Date:          (none)
   Sticky Options:       (none)

   Existing Tags:
```

```
      rel-0-4                 (revision: 1.4)
```

Редко требуется помечать одиночные файлы. Гораздо чаще нужно пометить все файлы, составляющие модуль, в стратегической точке цикла разработки, например, когда выпущена новая версия.

```
  $ cvs tag rel-1-0 .
cvs tag: Tagging .
T Makefile
T backend.c
T driver.c
T frontend.c
T parser.c
```

(Если вы дадите CVS каталог в качестве параметра командной строки, она обычно оперирует над всеми файлами в этом каталоге и, рекурсивно, ко всем подкаталогам, которые тот содержит. См. раздел Глава 9. *Рекурсивное поведение*).

Команда **checkout** имеет ключ командной строки `-r`, позволяющий извлечь определенную ревизию модуля. Этот флаг упрощает извлечение исходного текста, из которого сделана версия 1.0 модуля `tc` в когда-нибудь в будущем.

```
$ cvs checkout -r rel-1-0 tc
```

Это полезно, например, если кто-то заявляет, что в той версии была ошибка, но вы не можете найти ее в текущей рабочей копии.

Вы можете также извлечь модуль по состоянию на любую дату. Задав команде **checkout** ключ командной строки `-r`, следует остерегаться липких меток; см. раздел “Липкие метки”.

Когда вы помечаете более одного файла, вы можете думать о метке как о кривой, проведенной по таблице имен файлов и их номеров ревизий. Скажем, у нас есть пять файлов со следующими ревизиями:

```
file1   file2   file3   file4   file5

1.1     1.1     1.1     1.1  /--1.1* <-*- TAG
1.2*-   1.2     1.2    -1.2*-
1.3  \- 1.3*-   1.3   / 1.3
1.4           \ 1.4  /  1.4
               \-1.5*-   1.5
```

```
            1.6
```

Когда-то в прошлом, ревизии, отмеченные звездочками, были помечены. Вы можете думать о метке, как о ручке, приделанной к кривой, нарисованной на помеченных ревизиях. Когда вы тянете за ручку, вы получаете все помеченные ревизии. Еще одним способом представления является прямая линия, вдоль которой вы смотрите на набор файлов, и вдоль которой выровнены помеченные ревизии, например:

```
file1   file2   file3   file4   file5

             1.1
             1.2
       1.1    1.3                _
1.1    1.2    1.4    1.1         /
1.2*----1.3*----1.5*----1.2*----1.1   (--- <--- Look here
1.3           1.6    1.3         \_
1.4                  1.4
                     1.5
```

Что пометить в рабочем каталоге

Пример в предыдущей секции демонстрирует один из самых распространенных способов выбрать, какие ревизии пометить, а именно: выполнение команды **cvs tag** без параметров заставляет CVS выбрать ревизии, которые извлечены в текущем рабочем каталоге. Например, если копия файла `backend.c` в рабочем каталоге была извлечена из ревизии 1.4, то CVS пометит ревизию 1.4. Заметьте, что метка прилагается непосредственно к ревизии 1.4 в репозитории. Пометка — это не изменение файла, и не какая-либо операция, при которой сначала модифицируется рабочий каталог, а затем команда **cvs commit** переносит изменения в репозиторий.

Возможно, неожиданным обстоятельством того факта, что **cvs tag** оперирует с репозиторием, является то, что вы помечаете извлеченные ревизии, которые могут отличаться от файлов, измененных в вашем рабочем каталоге. Если вы хотите избежать ошибочного выполнения этой операции, укажите команде **cvs tag** ключ командной строки `-c`. Если в рабочем каталоге имеются измененные файлы, CVS завершится с сообщением об ошибке, не пометив ни одного файла:

```
$ cvs tag -c rel-0-4
cvs tag: backend.c is locally modified
cvs [tag aborted]: correct the above errors first!
```

Как помечать по дате или ревизии

Команда **cvs rtag** помечает репозиторий по состоянию на определенную дату и время (может использоваться для пометки последней ревизии). **rtag** работает прямо с содержимым репозитория (не требуется сначала извлекать рабочий каталог).

Нижеследующие ключи командной строки указывают, по какой дате или номеру ревизии помечать.

`-D` *дата*	Помечает самую новую ревизию не позднее даты.
`-f`	Полезно только вместе с `-D дата` или `-r метка`. Если не обнаружено соответствующей ревизии, вместо игнорирования файла используется самая новая ревизия.
`-r` *метка*	Помечать только файлы, содержащие существующую метку метка.

Команда **cvs tag** также позволяет выбрать файлы по ревизии или по дате, используя те же ключи командной строки `-D` и `-f`. Однако, это, скорее всего, вовсе не то, что вам надо, потому что **cvs tag** выбирает, какие файлы помечать, основываясь на файлах, существующих в рабочем каталоге, а не на файлах, существовавших на заданную дату или в заданной ревизии. Таким образом, обычно лучше использовать **cvs rtag**. Исключением могут быть случаи типа:

```
cvs tag -r 1.4 backend.c
```

Удаление, перемещение и удаление меток

Обычно метки не изменяются. Они существуют, чтобы хранить историю репозитория, поэтому изменять и удалять их обычно не нужно.

Однако же, могут быть случаи, в которых метки используются лишь временно или случайно помечаются неверные ревизии. Таким образом, нужно удалить, переместить или переименовать метку. Предупреждение: команды в этой секции опасны, они навсегда уничтожают информацию об истории и восстановление после ошибок может быть трудным или невозможным. Если вы — администратор CVS, вы можете захотеть ограничить использование этих команд с помощью файла `taginfo`.

Чтобы удалить метку, задайте ключ командной строки `-d` команде **cvs tag** или с**vs rtag**. Например:

```
cvs rtag -d rel-0-4 tc
```

удаляет метку rel-0-4 из модуля tc.

Когда мы говорим перемещение метки, мы хотим, чтобы существующее имя указывало на другие ревизии. Например, метка stable может указывать на ревизию 1.4 файла `backend.c`, а мы хотим, чтобы она указывала на ревизию 1.6. Чтобы переместить метку, задайте ключ командной строки `-F` командам **cvs tag** или **cvs rtag**. Например, вышеупомянутая задача может быть решена так:

```
cvs tag -r 1.6 -F stable backend.c
```

Когда мы говорим переименовать метку, мы хотим, чтобы другое имя указывало на те же ревизии, что и существующее. Например, мы могли ошибиться в написании имени метки и хотим исправить его, пока остальные не начали его использовать. Чтобы переименовать метку, сначала создайте новую метку, используя ключ командной строки `-r` команды **cvs rtag**, затем удалите старое имя. После этого новая метка указывает на точно те же самые файлы, что и старая. Например:

```
cvs rtag -r old-name-0-4 rel-0-4 tc
cvs rtag -d old-name-0-4 tc
```

Пометки при добавлении и удалении файлов

Пометки довольно запутанно взаимодействуют с операциями добавления и удаления файлов; в основном CVS отслеживает, существует файл или нет, не особенно беспокоясь о пустяках. По умолчанию, помечаются только файлы, которые имеют ревизии, соответствующие тому, что помечается. Файлы, которые еще не существуют или которые уже были удалены, просто пропускаются при пометке, при этом CVS знает, что отсутствие метки означает, что файл не существует в помеченном месте.

Однако, при этом можно потерять небольшое количество информации. Например, предположим, что файл был добавлен, а затем удален. Затем, если для этого файла отсутствует метка, нет способа сказать, потому ли это, что метка соответствует времени перед тем, как файл был добавлен, или после того, как он был удален. Если вы выполните **cvs rtag** с ключом командной строки `-r`, то CVS помечает файлы, которые были удалены, избегая таким образом проблемы. Например, можно указать `-r HEAD`, чтобы пометить головную ревизию.

Команда **cvs rtag** имеет ключ командной строки `-a`, очищающий метку с удаленных файлов, которые в противном случае не были бы помечены. Например, можно указать этот ключ вместе с `-F` при перемещении метки. Если переместить метку без `-a`, то метка на удаленных файлах все еще ссылалась бы на старую ревизию и не отражала бы того факта, что файл был удален. Я не считаю, что это обязательно, если указано `-r`, как отмечено выше.

Липкие метки

Иногда ревизия, находящаяся в рабочем каталоге, содержит также дополнительную информацию о себе: например, она может находиться на ветке (см. раздел Глава 8. *Создание и слияние ветвей*), или же может быть ограничена с помощью **checkout -D** или **update -D** версиями, созданными ранее указанной даты. Так как эта информация долговременно сохраняется, то есть действует на последующие команды над рабочей копией, то мы называем ее липкой.

В большинстве случаев липкость — это запутанный аспект CVS, о котором вам не следует думать. Однако, даже если вы не желаете использовать эту возможность, вы все же захотите что-нибудь узнать о липких метках (например, как их избежать!).

Можно использовать команду **status**, чтобы посмотреть, какие установлены липкие метки или даты:

```
$ cvs status driver.c
===================================================================
File: driver.c          Status: Up-to-date

   Version:            1.7.2.1 Sat Dec 5 19:35:03 1992
   RCS Version:        1.7.2.1 /u/cvsroot/yoyodyne/tc/driver.c,v
   Sticky Tag:         rel-1-0-patches (branch: 1.7.2)
   Sticky Date:        (none)
   Sticky Options:     (none)
```

Липкие метки остаются на ваших рабочих файлах до тех пор, пока вы не удалите их с помощью **cvs update -A**. Опция `-A` извлекает версию файла из головной ревизии ствола и забывает обо всех липких метках, датах и ключах командной строки.

Самое распространенное использование липких меток — указать, над какой ветвью идет работа, что описано в разделе “Доступ к веткам”. Однако, липкие метки также используются и без веток. Предположим, например, что вы хотите избежать обновления вашего рабочего каталога, чтобы защититься от изменений, которые делают ваши коллеги. Вы, конечно, можете просто не выполнять команду **cvs update**. Если же вы хотите избежать обновления только части большого дерева, то липкие метки могут помочь. Если вы извлечете определенную ревизию, скажем, 1.4, то она станет липкой. Последующие команды **cvs update** не станут извлекать последнюю ревизию до тех пор, пока вы не очистите метку с помощью **cvs update -A**. Точно так же, использование ключа командной строки `-D` команд **update** и **checkout** задает липкую дату, которая используется для будущих извлечений.

Люди часто хотят извлечь старую версию файла без установки липкой метки. Это можно сделать с помощью ключа командной строки `-p` команд **checkout** или **update**, которая посылает содержимое файла на стандартный вывод. Например:

```
$ cvs update -p -r 1.1 file1 >file1
===================================================================
Checking out file1
RCS: /tmp/cvs-sanity/cvsroot/first-dir/Attic/file1,v
```

```
VERS: 1.1
***************
$
```

Однако, это не самый простой способ, если вы спрашиваете, как отменить последнее фиксирование (в этом примере — поместить `file1` обратно в то состояние, в котором он был в ревизии 1.1). В этом случае лучше будет использовать ключ командной строки `-j` команды **update**; дальнейшее обсуждение находится в разделт “Слияние изменений между двумя ревизиями”.

Глава 8. Создание и слияние ветвей

CVS позволяет изолировать изменения в отдельной линии разработки, называемой веткой. Когда вы изменяете файлы на ветке, эти изменения не появляются в основном стволе или на других ветках.

Позже вы можете переместить изменения с одной ветки на другую или же с ветки в ствол, это называется слиянием. Сначала выполняется **cvs update -j**, чтобы слить изменения в рабочий каталог, а затем эти изменения можно зафиксировать, что фактически приведет к копированию изменений на другую ветку.

Для чего хороши ветви?

Предположим, был выпущен tc версии 1.0. Вы продолжаете его разработку, планируя выпустить версию 1.1 через пару месяцев. Через некоторое время ваши пользователи начинают жаловаться на серьезную ошибку. Вы извлекаете версию 1.0 (см. раздел "Метки ревизий") и находите ошибку, для исправления которой требуется всего лишь тривиальное изменение). Однако же, текущая версия исходников находится в крайне нестабильном состоянии и не стабилизируется по крайней мере еще месяц. Вы не можете выпустить исправленную версию, основываясь на свежих исходниках.

В подобной ситуации имеет смысл создать ветку в дереве ревизий, содержащую файлы, из которых состояла версия 1.0. Затем вы вносите изменения в ветвь без вторжения в основной ствол. Потом вы сможете либо внести те же самые изменения в основной ствол, либо оставить их только на ветви.

Создание ветви

Вы можете создать ветвь, используя **cvs tag -b**. Например, если вы находитесь в каталоге с рабочей копией:

```
$ cvs tag -b rel-1-0-patches
```

Это отщепляет ветку, основанную на текущей ревизии рабочей копии, и присваивает этой ветке имя 'rel-1-0-patches'.

Важно понимать, что ветки создаются в репозитории, а не в рабочей копии. Создание ветки, основанной на текущей ревизии, как в

вышеприведенном примере, НЕ переключает рабочую копию на использование ветки (см. раздел “Доступ к веткам”, где описано, как сделать это).

Можно также создать ветку вообще без использования рабочей копии, используя **rtag**.

```
$ cvs rtag -b -r rel-1-0 rel-1-0-patches tc
```

`-r rel-1-0` означает, что эта ветка имеет корневую ревизию, соответствующую метке ’rel-1-0’. Это не обязательно должна быть самая последняя ревизия: довольно часто бывает полезно отщепить ветку от старой ревизии (например, для исправления ошибки в старой версии, которая в основном стабильна).

Как и в случае с **tag**, ключ командной строки `-b` заставляет **rtag** создать ветку (а не символьное имя ревизии). Заметьте, что номера ревизий, соответствующих ’rel-1-0’, скорее всего, будут разными в разных файлах.

Таким образом, полный эффект этой команды — создать новую ветку, которая называется ’rel-1-0-patches’, в модуле `tc`, которая растет в дереве ревизий из точки, помеченной как ’rel-1-0’.

Доступ к веткам

Вы можете извлечь ветку двумя способами: извлекая ее из репозитория в чистом каталоге или переключая существующую рабочую копию на ветку.

Для того, чтобы извлечь ветку из репозитория, выполните команду `checkout` с ключом командной строки `-r`, с именем метки в качестве параметра (См. раздел “Создание ветви”).

```
$ cvs checkout -r rel-1-0-patches tc
```

Если у вас уже есть рабочая копия, вы можете переключить ее на нужную ветку с помощью **update -r**:

```
$ cvs update -r rel-1-0-patches tc
```

или, что то же самое:

```
$ cd tc
$ cvs update -r rel-1-0-patches
```

Неважно, что рабочая копия была извлечена из основного ствола или какой-нибудь другой ветки: вышеприведенная команда переключит ее на указанную ветку. Подобно обычной команде **update**, **update -r** сливает сделанные изменения, уведомляя вас о произошедших конфликтах.

Когда вы связываете рабочую копию с какой-либо веткой, она будет оставаться связанной, пока вы не укажете обратного. Это означает, что изменения, которые фиксируются из рабочей копии, будут добавлять новые ревизии на ветку, оставляя без изменений основной ствол и другие ветки.

Чтобы узнать, на какой ветви находится рабочая копия, можно использовать команду **status**. В том, что она вывела на экран, обратите внимание на поле, которое называется 'Sticky tag' (см. раздел "Липкие метки") — здесь CVS сообщает, на какой ветви находятся рабочие файлы:

```
$ cvs status -v driver.c backend.c
===================================================================
File: driver.c Status: Up-to-date

Version: 1.7 Sat Dec 5 18:25:54 1992
RCS Version: 1.7 /u/cvsroot/yoyodyne/tc/driver.c,v
Sticky Tag: rel-1-0-patches (branch: 1.7.2)
Sticky Date: (none)
Sticky Options: (none)

Existing Tags:
rel-1-0-patches (branch: 1.7.2)
rel-1-0 (revision: 1.7)

===================================================================
File: backend.c Status: Up-to-date

Version: 1.4 Tue Dec 1 14:39:01 1992
RCS Version: 1.4 /u/cvsroot/yoyodyne/tc/backend.c,v
Sticky Tag: rel-1-0-patches (branch: 1.4.2)
Sticky Date: (none)
Sticky Options: (none)
```

```
Existing Tags:
rel-1-0-patches (branch: 1.4.2)
rel-1-0 (revision: 1.4)
rel-0-4 (revision: 1.4)
```

Не смущайтесь тем, что номера ветвей для каждого файла различны ('1.7.2' и '1.4.2', соответственно). Метка ветви одна и та же, 'rel-1-0-patches', и все файлы действительно находятся на одной и той же ветке. Номера лишь отражают ту точку в истории файла, в которой появилась ветвь. Из вышеприведенного примера можно узнать, что перед тем, как была создана ветка, `driver.c` претерпел больше изменений, чем `backend.c`.

См. раздел "Ветки и ревизии", где подробно описано, как устроены номера ветвей.

Ветки и ревизии

Обычно история ревизий файла — это линейная возрастающая последовательность номеров (см. раздел "Номера ревизий"):

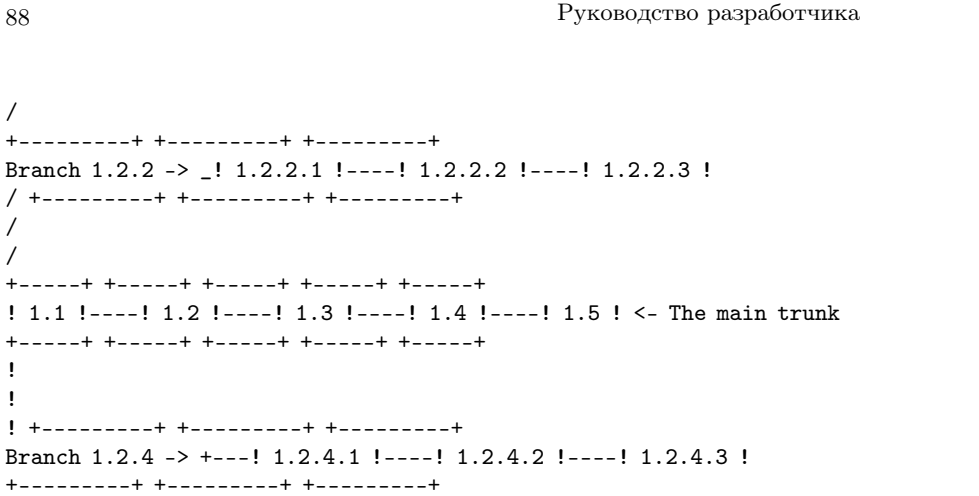
```
+-----+ +-----+ +-----+ +-----+ +-----+
! 1.1 !----! 1.2 !----! 1.3 !----! 1.4 !----! 1.5 !
+-----+ +-----+ +-----+ +-----+ +-----+
```

Однако же, CVS не ограничен линейной разработкой. Дерево ревизий может быть расщеплено на ветви, где каждая ветвь — самостоятельная линия разработки. Изменения, сделанные на одной ветке, легко могут быть внесены также и в основной ствол.

Каждая ветка имеет номер ветки, состоящий из нечетного числа десятичных чисел, разделенных точками. Номер ветки создается путем добавления целого числа к номеру ревизии, от которой была отщеплена ветка. Номера веток позволяют отщеплять от одной и той же ревизии несколько веток.

Все ревизии на ветке имеют номера ревизий, образованные путем добавления порядкового номера к номеру ветки. Вот иллюстрация создания веток.

```
+-------------+
Branch 1.2.2.3.2 -> ! 1.2.2.3.2.1 !
/ +-------------+
/
```



Обычно не требуется задумываться о точных деталях того, как строятся номера веток, но вот еще подробности: когда CVS создает номер ветки, он берет первое неиспользованное четное число, начиная с двойки. Поэтому, если вы хотите создать ветку от ревизии 6.4, она будет называться 6.4.2. Номера веток, заканчивающиеся на ноль (например, 6.4.0), используются для внутренних нужд CVS (см. раздел “Волшебные номера веток”). Ветка 1.1.1 имеет специальное значение.

Волшебные номера веток

В этой секции описана возможность CVS, называющаяся волшебные ветки. В большинстве случаев вам не потребуется беспокоиться о волшебных ветках, так как CVS сам следит за ними. Однако, при определенных условиях их можно увидеть, и поэтому полезно было бы узнать, как они работают.

Номера веток состоят из нечетного количества десятичных целых чисел, разделенных точками. См. раздел “Номера ревизий”. Однако же, это не полная правда. Из соображений эффективности CVS иногда вставляет лишний ноль во вторую справа позицию (1.2.4 становится 1.2.0.4, а 8.9.10.11.12 становится 8.9.10.11.0.12 и так далее).

CVS довольно хорошо прячет такие «волшебные» ветки, но в нескольких местах ему это не удается:

- Номера волшебных веток появляются в выдаче cvs log.
- q
  Вы не можете указать символическое имя ветки в команде cvs admin.

Можно использовать команду **admin**, чтобы переназначить символическое имя ветки на то, которое ожидает увидеть CVS. Например, если R4patches присвоено ветке 1.4.2 (волшебный номер 1.4.0.2) в файле `numbers.c`', можно сделать так:

```
$ cvs admin -NR4patches:1.4.2 numbers.c
```

Это сработает, только если хотя бы одна ревизия уже была зафиксирована на ветке. Будьте очень осторожны, чтобы не присвоить метку неправильному числу, так как нет способа узнать, чему была присвоена эта метка вчера (за исключением ежедневного резервного копирования).

Слияние веток

Вы можете объединить изменения, сделанные на ветке, с вашей рабочей копией, добавив флаг `-j ветка` к команде **update**. В результате CVS внедряет в рабочую копию изменения, сделанные между корневой ревизией ветки и свежайшей ревизией на этой ветке.

Ключ командной строки `-j` означает «объединить» (join).

Представьте себе такое дерево ревизий:

```
+-----+ +-----+ +-----+ +-----+
! 1.1 !----! 1.2 !----! 1.3 !----! 1.4 ! <- основной ствол
+-----+ +-----+ +-----+ +-----+
!
!
! +---------+ +---------+
Ветвь R1fix -> +---! 1.2.2.1 !----! 1.2.2.2 !
+---------+ +---------+
```

Ветке 1.2.2 была назначена метка (символьное имя) 'R1fix'. В нижеследующем примере предполагается, что модуль `mod` содержит единственный файл, `m.c`.

```
$ cvs checkout mod # Извлечь последнюю ревизию, 1.4

$ cvs update -j R1fix m.c # Слить все изменения, сделанные на ветке,
# т. е. изменения между ревизиями 1.2
# и 1.2.2.2, в рабочую копию файла
```

```
$ cvs commit -m "Included R1fix" # создать ревизию 1.5.
```

В результате операции слияния может произойти конфликт. В это случае вам сначала надо справиться с ним перед фиксированием изменений.

Команда **checkout** также поддерживает флаг `-j ветка`. Можно добиться эффекта, обсуждавшегося выше, с помощью

```
$ cvs checkout -j R1fix mod
$ cvs commit -m "Добавлен R1fix"
```

многократное слияние из ветки

Мы продолжаем обсуждение примера. Теперь дерево ревизий выглядит так:

```
+-----+ +-----+ +-----+ +-----+ +-----+
! 1.1 !----! 1.2 !----! 1.3 !----! 1.4 !----! 1.5 ! <- ствол
+-----+ +-----+ +-----+ +-----+ +-----+
! *
! *
! +---------+ +---------+
Ветвь R1fix -> +---! 1.2.2.1 !----! 1.2.2.2 !
+---------+ +---------+
```

Здесь линия из звездочек представляет собой слияние ветки 'R1fix' с основным стволом, обсуждавшееся только что.

Предположим теперь, что разработка ветки 'R1fix' продолжается:

```
+-----+ +-----+ +-----+ +-----+ +-----+
! 1.1 !----! 1.2 !----! 1.3 !----! 1.4 !----! 1.5 ! <- ствол
+-----+ +-----+ +-----+ +-----+ +-----+
! *
! *
! +---------+ +---------+ +---------+
Ветвь R1fix -> +---! 1.2.2.1 !----! 1.2.2.2 !----! 1.2.2.3 !
+---------+ +---------+ +---------+
```

и теперь вы опять хотите слить свежайшие изменения с основным стволом. Если бы вы просто использовали команду **cvs update -j**

R1fix m.c, то CVS попыталась бы опять слить уже слитые изменения, что привело бы к нежелательным результатам.

Вместо этого вам нужно указать, что вы хотите слить только те изменения на ветке, что еще не были объединены со стволом. Для этого вы указываете два ключа командной строки `-j`, и CVS сливает изменения между первой и второй ревизиями. Например, в этом случае самым простым способом будет

```
cvs update -j 1.2.2.2 -j R1fix m.c # Слить изменения между 1.2.2.2 и
# головой ветки R1fix
```

Проблемой здесь является то, что вы должны вручную указать ревизию 1.2.2.2. Чуть лучшим подходом будет использование даты совершения последнего слияния.

```
cvs update -j R1fix:yesterday -j R1fix m.c
```

Еще лучше было бы помечать ветку 'R1fix' после каждого слияния со стволом, и использовать эту метку для дальнейших слияний:

```
cvs update -j merged_from_R1fix_to_trunk -j R1fix m.c
```

Слияние изменений между двумя ревизиями

С помощью двух флагов `-j ревизия`, команды **update** и **checkout** могут сливать изменения между любыми двумя ревизиями в ваш рабочий файл.

Команда

```
$ cvs update -j 1.5 -j 1.3 backend.c
```

отменит изменения, сделанные между ревизиями 1.3 и 1.5. Обратите внимание на порядок указания ревизий!

Если вы попробуете использовать эту опцию при работе с несколькими файлами, помните, что номера ревизий, вероятно, будут сильно отличаться для разных файлов. В таких случаях почти всегда следует использовать символьные метки, а не номера ревизий.

Указав два ключа командной строки `-j`, можно также отменить удаление или добавление файла. Например, предположим, у вас есть файл `file1`, существовавший в ревизии 1.1. Затем вы удалили его, создав «мертвую» ревизию 1.2. Теперь предположим, что вы хотите добавить его опять, с тем же самым содержимым, что он имел ранее. Вот как сделать это:

```
$ cvs update -j 1.2 -j 1.1 file1
U file1
$ cvs commit -m test
Checking in file1;
/tmp/cvs-sanity/cvsroot/first-dir/file1,v <-- file1
new revision: 1.3; previous revision: 1.2
done
$
```

При слиянии можно добавлять и удалять файлы

Если изменения, которые вы сливаете, включают в себя удаление или добавление каких-либо файлов, то команда **update -j** учтет такие добавления и удаления.

Например:

```
cvs update -A
touch a b c
cvs add a b c ; cvs ci -m «added» a b c
cvs tag -b branchtag
cvs update -r branchtag
touch d ; cvs add d
rm a ; cvs rm a
cvs ci -m «added d, removed a»
cvs update -A
cvs update -jbranchtag
```

После того, как эти команды выполнены, а также выполнена команда **cvs commit**, файл `a` будет удален, а файл `d` будет добавлен в основной ствол.

Глава 9. Рекурсивное поведение

Почти все подкоманды CVS работают рекурсивно, если вы укажете в качестве аргумента каталог. Например, представим себе такую структуру каталогов:

```
$HOME
|
+--tc
| |
+--CVS
| (служебные файлы CVS)
+--Makefile
+--backend.c
+--driver.c
+--frontend.c
+--parser.c
+--man
| |
| +--CVS
| | (служебные файлы CVS)
| +--tc.1
|
+--testing
|
+--CVS
| (служебные файлы CVS)
+--testpgm.t
+--test2.t
```

Если `tc` — это текущий рабочий каталог, то верны следующие утверждения:

- **cvs update testing** эквивалентно

  ```
  cvs update testing/testpgm.t testing/test2.t
  ```

- **cvs update testing man** обновляет все файлы в подкаталогах
- **cvs update .** или просто **cvs update** обновляет все файлы в каталоге `tc`.

Если команде **update** не было дано ни одного аргумента, то она обновит все файлы в текущем рабочем каталоге и во всех его подкаталогах. Другими словами, `.` является аргументом по умолчанию для **update**. Это также истинно для большинства подкоманд CVS, а не только для команды **update**.

Рекурсивное поведение подкоманд CVS может быть отключено с помощью ключа командной строки `-l`, и наоборот, ключ командной строки `-R` может использоваться для принудительной рекурсии, если `-l` был указан в `~/.cvsrc`.

```
$ cvs update -l # Не обновлять файлы в подкаталогах
```

Глава 10. Добавление, удаление и переименование файлов и каталогов

В процессе разработки проекта часто требуется добавлять, удалять или переименовывать файлы и каталоги. Исходя из общих принципов, требуется, чтобы CVS запоминала факт совершения такого действия, вместо того, чтобы совершать необратимое изменение, точно так же, как она обращается с изменениями файлов. Точные механизмы, действующие в этих случаях, зависят от конкретной ситуации.

Добавление файлов в каталог

Для того, чтобы добавить новый файл в каталог, совершите следующие шаги:

- Сначала у вас должна быть рабочая копия каталога. См. раздел Получение исходного кода.
- Создайте новый файл в рабочей копии каталога.
- Используйте комманду **cvs add имя_файла**, чтобы сообщить CVS, что вы хотите хранить историю изменений этого файла. Если в файле хранятся двоичные данные, добавьте ключ командной строки `-kb`.
- Используйте команду **cvs commit имя_файла**, чтобы поместить файл в репозиторий. Другие разработчики не увидят этот файл, пока вы не выполните эту команду.

Можно также использовать команду **add** для добавления нового каталога.

В отличие от большинства других команд, команда **add** не является рекурсивной. Вы даже не можете сказать **cvs add foo/bar**. Вместо этого, вам потребуется выполнить

```
$ cd foo
$ cvs add bar
```

Команда: **cvs add [-k kflag] [-m сообщение] файлы ...**

Добавить файлы в список на помещение в репозиторий. Файлы или каталоги, указанные в команде **add**, должны существовать в текущем

каталоге. Для того, чтобы добавить в репозиторий целое дерево каталогов, например, файлы, полученные от стороннего поставщика, используйте команду **import**.

Добавленные файлы не помещаются в репозиторий, пока вы не выполните команду **commit**, зафиксировав тем самым изменения. Выполнение команды **add** для файла, который был удален командой **remove**, отменит действие **remove**, если после нее еще не была выполнена команда **commit**. См. раздел "Удаление файлов", там находится пример использования команды.

Ключ командной строки `-k` задает способ по умолчанию, которым будут извлекаться файлы.

Ключ командной строки `-m` задает описание файла. Описание появляется в журнале истории, если разрешено его использование. Также это описание будет сохранено в репозитории, когда файл будет зафиксирован. Команда **log** показывает это описание. Описание может быть изменено с помощью команды **admin -t**. Если вы опустите флаг `-m описание`, то у вас не спросят описания, а будет использована пустая строка.

Например, нижеследующие команды добавляют файл 'backend.c' в репозиторий:

```
$ cvs add backend.c
$ cvs commit -m «Early version. Not yet compilable.» backend.c
```

Когда вы добавляете файл, он добавляется только на ту ветку, над которой вы работаете (см. раздел Глава 8. *Создание и слияние ветвей*). Вы можете позднее поместить добавления на другую ветку, если захотите (См. раздел "При слиянии можно добавлять и удалять файлы").

Удаление файлов

Содержимое каталогов меняется. Добавляются новые файлы, исчезают старые. Однако же, вам хотелось бы извлекать точные копии старых версий вашего проекта.

Вот как можно удалить файл, сохранив доступ к его старым ревизиям:

- Убедитесь, что у вас неизмененная версия этого файла. См. раздел Просмотр изменений, там описан один из способов убедиться в этом. Можно также использовать команды **status** или **update**.

Если вы удалите файл без предварительной фиксации изменений, вы, конечно же, не сможете извлечь этот файл в том виде, в котором он находился перед удалением.

- Удалите файл из рабочей копии каталога. Например, можно использовать программу **rm**.
- Выполните **cvs remove имя-файла**, чтобы сообщить CVS, что вы действительно хотите удалить этот файл.
- Выполните **cvs commit имя-файла**, чтобы зафиксировать удаление файла в репозитории.

Когда вы фиксируете удаление файла, CVS запоминает, что этого файла более не существует. Впрочем, он может существовать на одних ветках и не существовать на других, или же можно впоследствии добавить другой файл с тем же самым именем. CVS корректно создаст или не станет создавать файл, основываясь на ключах командной строки `-r` или `-D`, заданных в командах **checkout** или **update**.

Команда: **cvs remove [ключи] файлы ...**

Помещает файлы в список на удаление из репозитория (для того, чтобы эта команда сработала, нужно, чтобы файлы были удалены из рабочего каталога). Эта команда не удаляет файлы из репозитория, пока вы не зафиксируете удаление. Полный список ключей находится в разделе ???.

Вот пример удаления нескольких файлов:

```
$ cd test
$ rm *.c
$ cvs remove
cvs remove: Removing .
cvs remove: scheduling a.c for removal
cvs remove: scheduling b.c for removal
cvs remove: use 'cvs commit' to remove these files permanently
$ cvs ci -m «Removed unneeded files»
cvs commit: Examining .
cvs commit: Committing .
```

Для удобства можно удалять файлы и одновременно делать **cvs remove**, используя ключ командной строки `-f`. Например, вышеприведенный пример можно переписать так:

```
$ cd test
$ cvs remove -f *.c
```

```
cvs remove: scheduling a.c for removal
cvs remove: scheduling b.c for removal
cvs remove: use 'cvs commit' to remove these files permanently
$ cvs ci -m «Removed unneeded files»
cvs commit: Examining .
cvs commit: Committing .
```

Если вы выполните команду **remove**, а затем перемените свое решение, еще не зафиксировав удаление, то команду **remove** можно отменить с помощью команды **add**.

```
$ ls
CVS ja.h oj.c
$ rm oj.c
$ cvs remove oj.c
cvs remove: scheduling oj.c for removal
cvs remove: use 'cvs commit' to remove this file permanently
$ cvs add oj.c
U oj.c
cvs add: oj.c, version 1.1.1.1, resurrected
```

Если вы осознаете свою ошибку перед выполнением команды **remove**, можно использовать **update**, чтобы воскресить файлы:

```
$ rm oj.c
$ cvs update oj.c
cvs update: warning: oj.c was lost
U oj.c
```

Когда вы удаляете файл, он удаляется только с той ветки, на которой вы работаете (см. раздел Глава 8. *Создание и слияние ветвей*). Позже можно слить удаления на другую ветку, если захотите (см. раздел "При слиянии можно добавлять и удалять файлы").

Удаление каталогов

В принципе удаление каталогов в чем-то подобно удалению файлов — вы не хотите, чтобы каталог существовал в текущем рабочем каталоге, но вы хотите также, чтобы можно было извлекать старые версии проекта, в которых еще существовал каталог.

Можно удалить каталог, удалив все файлы в нем. Нет способа удалить сам каталог. Вместо этого вы задаете командам **cvs update**,

cvs checkout или **cvs export** ключ командной строки `-P`, который заставит CVS удалять пустые каталоги в рабочем каталоге. Вероятно, лучше всего будет всегда указывать `-P`, если вы хотите, чтобы существовал пустой каталог, поместите в него пустой файл, например, `.keepme`, чтобы не дать CVS с ключом `-P` удалить этот каталог.

Заметьте, что при использовании ключей `-r` или `-D` с командами **checkout** и **export** подразумевается также использование `-P`. При этом CVS сможет создать или не создавать каталог, в зависимости от того, находились ли в этом каталоге какие-либо файлы в конкретной версии проекта.

Перемещение и переименование файлов

Перемещение файлов в другой каталог или переименование их несложно, но некоторые аспекты могут быть неочевидными. Перемещение и переименование каталогов еще сложнее. См. раздел “Перемещение и переименование каталогов”.

В нижеприведенных примерах предполагается, что файл `old` переименовывается в `new`.

Обычный способ переименования

Обычным способом перемещения файла является копирование `old` в `new`, а затем выполнение команд CVS для удаления файла `old` из репозитория и добавления туда файла `new`.

```
$ mv old new
$ cvs remove old
$ cvs add new
$ cvs commit -m "old переименован в new" old new
```

Это самый простой способ переместить файл, он не подвержен ошибкам, и сохраняет историю совершенных действий. Заметьте, что для доступа к истории файла нужно указать старое или новое имя, в зависимости от периода истории, к которому вы обращаетесь. Например, **cvs log old** выдаст журнал вплоть до момента переименования.

Когда `new` фиксируется, нумерация его ревизий начнется с нуля, обычно с 1.1, поэтому если это вам не нравится, используйте ключ командной строки `-r номер` команды **commit**. Дальнейшую информацию смотрите в разделе “Номера ревизий”

Перемещение файла с ревизиями

Этот метод более опасен, потому что требует перемещения файлов в репозитории. Прочтите всю главу перед попытками применить этот метод!

```
$ cd $CVSROOT/dir
$ mv old,v new,v
```

Преимущества:

- Журнал изменений сохраняется.
- Номера ревизий не изменяются.

Недостатки:

- Нет простого способа извлечь старые версии проекта из репозитория. Файл будет извлекаться под именем `new` даже для версий проекта, в которых он еще не был переименован.
- Не сохраняется информация о том, когда был переименован файл.
- Могут произойти неприятности, если кто-нибудь захочет поработать с файлом ревизий, пока вы его перемещаете. Убедитесь, что никто более не обращается к репозиторию, пока вы выполняете операцию.

Копирование файла с ревизиями

Этот способ также требует прямых изменений репозитория. Он безопасен, но не без подводных камней.

- Копировать RCS-файл в репозитории

```
$ cd $CVSROOT/dir
$ cp old,v new,v
```

- Удалить старый файл

```
$ cd ~/dir
$ rm old
$ cvs remove old
$ cvs commit old
```

- Удалить все метки из `new`

```
$ cvs update new
$ cvs log new
```

- Запомнить все метки, не являющиеся именами веток

```
$ cvs tag -d tag1 new
$ cvs tag -d tag2 new
...
```

Удалив метки, вы сможете извлекать старые ревизии.
Преимущества:

- Извлечение старых ревизий работает корректно, если вы используете для извлечения ревизий ключ командной строки `-rметка`, а не `-Dдата`.
- Журнал изменений остается в целости и сохранности.
- Номера ревизий не искажаются.

Недостатки:

- Нет способа легко увидеть историю файла до переименования.

Перемещение и переименование каталогов

Обычный способ переименовать или переместить каталог — переименовать или переместить каждый файл в нем, как описано в разделе “Обычный способ переименования”. Затем следует извлечь их заново, используя ключ командной строки `-P`, как описано в разделе “Удаление каталогов”.

Если вам действительно нужно возиться с репозиторием, чтобы переименовать или удалить каталог в репозитории, вы можете сделать это так:

1. Уведомить всех, у кого есть извлеченная копия каталога, что каталог будет переименован. Они должны зафиксировать свои изменения и удалить рабочие копии, перед тем, как вы предпримете дальнейшие шаги.

2. Переименуйте каталог внутри репозитория.

```
$ cd $CVSROOT/родительский-каталог
$ mv старый-каталог новый-каталог
```

3. Исправьте административные файлы CVS, если это требуется (например, если вы переименовали целый модуль).
4. Сообщите всем, что они могут извлечь свои рабочие копии опять и продолжить работу.

Если кто-то не удалил свою рабочую копию, команды CVS будут отказываться работать, пока он не удалит каталог, которого больше не существует в репозитории.

Почти всегда гораздо лучшим способом будет переместить файлы в каталоге, вместо того, чтобы перемещать каталог, потому что иначе вы, скорее всего, не сможете корректно извлекать старые версии вашего проекта, так как они, вероятно, зависят от имен каталогов.

Часть III. LAN-Crypto — Библиотека разработчика программного обеспечения защиты информации

Библиотека предназначена для разработчиков, использующих в своих приложениях защиту информации от несанкционированного доступа.

Глава 11. Введение

Библиотека разработки программных средств защиты информации от несанкционированного доступа включает в себя набор библиотек, обладающих средствами, позволяющими разработчикам повышать безопасность своих приложений. Она содержит функции кодирования и декодирования информации, контроля целостности данных и идентификации пользователей с использованием цифровых сертификатов.

Разработчик приложения, в которое встроена защита информации средствами библиотеки, должен быть знаком с основами криптографии. В документации по Библиотеке (файлы в формате `CHM`) даны краткие описания основных концепций, алгоритмов и задач криптографии:

Большинство криптографических операций используют секретные и открытые ключи.

Секретные ключи — это секретные компоненты, которыми обладают пользователи системы. Секретный ключ похож на пароль, но довольно длинный и содержится в специальном контейнере для секретного ключа. У пользователя нет прямого доступа к секретному ключу, он может использоваться только соответствующими программами.

Открытые ключи не являются тайными и представляют открытую и общедоступную часть ключа пользователя. Важно то, что, хотя пользователь открывает доступ к своему открытому ключу остальным пользователям системы, они не могут вычислить секретный ключ по открытому ключу. Однако обмен открытыми ключами между двумя пользователями позволяет установить между ними защищённое соединение.

Проблема защищенного обмена информацией через незащищённые компьютерные сети включает три основные области:

- секретность — предотвращение прочтения секретных данных любым лицом, кроме назначенного их получателя;
- целостность — гарантия, что полученные данные есть в точности те, которые были отправлены, т.е. что они не были изменены с момента отправки;
- аутентификация — установление личности или группы, с которыми устанавливается соединение.

Эти задачи решаются криптографическими технологиями при использовании разнообразных криптографических алгоритмов.

Для обеспечения секретности используются алгоритмы кодирования/декодирования. Для целостности — алгоритмы хэширования и цифровой подписи/проверки подписи. Для аутентификации пользователей используется система управления пользовательскими сертификатами.

Библиотека предоставляет набор алгоритмов кодирования — как разработанных *ЛАН Крипто*, так и стандартных алгоритмов, известных во всём мире.

Существуют разнообразные специальные алгоритмы для кодирования, хэширования и подписи данных. Качество алгоритма определить сложно. Алгоритмы, которые выглядят многообещающе, могут оказаться очень лёгкими для взлома определёнными методами атаки. При выборе алгоритмов представляется разумным выбирать те из них, которые использовались в течение некоторого времени, за которое они успешно отразили все атаки.

В таблице ниже приведены параметры алгоритмов — размер блока в битах, длина сеансового ключа в битах.

«ЛАН Крипто» рекомендует:

Недавно были разработаны надёжные и сверхбыстрые алгоритмы кодирования Wicker98 и Nush. Эти очень эффективные алгоритмы блочного кодирования могут быть успешно применены в любом приложении, имеющем операцию кодирования.

Wicker98 и Nush — прекрасные алгоритмы блочного кодирования, известные на мировом рынке.

Библиотека предоставляет набор алгоритмов подписи, как разработанных «ЛАН Крипто», так и других, широко известных во всём мире.

«ЛАН Крипто» рекомендует:

Алгоритм подписи LANS (Notary-S) позволяет делать длину цифровой подписи на 35% короче, чем в алгоритме DSA, и на 60% короче, чем у алгоритма GOSTR3410. Он уже принят в качестве нового корпоративного стандарта цифровой подписи целым рядом индустриальных и коммерческих концернов в России и СНГ. Он также был принят как отраслевой стандарт газовой промышленности России и стал национальным стандартом цифровой подписи в нескольких странах СНГ.

Библиотека предоставляет набор алгоритмов хэширования:

«ЛАН Крипто» рекомендует:

Алгоритм хэширования SHA1, широко используемый во всём мире. SHA1 используется вместе с алгоритмами подписи DSA и LANS.

Библиотека предоставляет несколько методов кодирования данных (режимов кодирования):

Поточный режим кодирования кодирует данные по биту за раз, и используется в поточных кодах (Веста-2M, RC4). При использовании блочных кодов (Wicker, TripleDES, т.п.) режим блочного кодирования кодирует данные по блоку за раз. Хотя это выполняется медленнее, чем в поточных алгоритмах, блочные более безопасны. В блочных алгоритмах используются четыре режима кодирования. В большинстве случаев предпочтительнее использование режима CBC.

EBC — электронная кодовая книга Простой режим кодирования, кодирующий блоки по отдельности и не использующий обратную связь.

CBC — сцепление блоков по шифр-тексту Режим блочного кодирования, в котором обратная связь представлена комбинированием шифр-текста и открытого текста.

CFB — обратная связь по шифр-тексту Режим блочного кодирования, преобразующий небольшие прибавки открытого текста в шифр-текст при использовании сдвигового регистра вместо преобразования за раз блока целиком.

OFB — обратная связь по выходу Режим блочного кодирования, использующий обратную связь похожим с CFB образом, но использующий другой метод заполнения сдвигового регистра.

Обычно сообщение и кодируется, и декодируется на общем сеансовом ключе и поиск способа, которым отправитель и получатель сообщения могут без опасности перехвата обменяться этим ключом, может стать отдельной проблемой. Это проблема обмена ключами.

Библиотека поддерживает два протокола обмена ключами, решающих эту проблему:

- протокол Диффи-Хеллмана (DH) — обмен открытыми ключами между двумя пользователями (отправителем и получателем);
- протокол групповой работы (WG) — используется для кодирования для нескольких получателей; этот протокол не требует использования секретного и открытого ключей отправителя.

Трудность использования открытых ключей для генерации сеансового ключа в том, как послать ваш открытый ключ остальным пользователям так, чтобы он не мог быть перехвачен и подменён. Обычно эту проблему решает Центр Сертификации — доверенная организация, которая подписывает, хранит и распространяет открытые ключи.

Таблица 11.1.

Алгоритм	**Размер блока**	**Длина ключа**	**Описание**
Nush	128	256	Nush — самая последняя разработка "ЛАН Крипто"
Wicker98	128	128	Wicker98 от "ЛАН Крипто" — наиболее эффективный и известный блочный код. В настоящее время он является лучшим по всем основным параметрам
Веста-2М	1	512	Веста-2М от "ЛАН Крипто" — поточный код, за последние 5 лет ставший практически стандартом де-факто для защиты электронных документов. Он используется целым рядом коммерческих компаний, банков и корпораций в России и СНГ.
RC4	1	64	Поточный код, хорошо известный в США
DES	64	64	Блочный код

Таблица 11.2.

Алгоритм	**Описание**	**Предпочтительный алгоритм хэширования**
LANS (Notary-S)	Алгоритм подписи, разработанный "ЛАН Крипто" и профессором Клаусом Шнорром (Германия). Включает основные элементы национального американского стандарта цифровой подписи, и более эффективную функцию сжатия.	Gas48, SHA1
DSA	Алгоритм подписи DSA (США)	SHA1
RSA	Алгоритм подписи RSA (США)	MD5
ГОСТ РФ 34.10-94	Алгоритм подписи. Стандарт РФ	ГОСТ РФ 34.11-94
LANS ECC	Алгоритм подписи LANS на эллиптических кривых	Gas48, SHA1
GOST ECC	Алгоритм подписи на эллиптических кривых. Стандарт РФ	ГОСТ РФ 34.11-01

Таблица 11.3.

Алгоритм	**Размер хэша (в байтах)**	**Описание**
SHA1	20	Алгоритм хэширования SHA1
SHA	20	Алгоритм хэширования SHA
MD5	16	Алгоритм хэширования MD5
GAS48	48	Алгоритм хэширования GAS48 был разработан в "ЛАН Крипто" как отраслевой стандарт для РАО "Газпром".
ГОСТ 34.11-94	32	Алгоритм хэширования ГОСТ РФ 34.11-94 — стандарт РФ. Должен использоваться совместно с алгоритмом подписи ГОСТ 34.10-94

Глава 12. Интерфейс разработки приложений (API) Библиотеки

API Библиотеки — это гибкий, мощный и современный интерфейс. Он разделен на несколько функциональных уровней: основные части API — ctBaseLib, ctMsgLib и ctCertStoreLib.

Составляющие библиотеки:

Каждая часть Библиотеки разработчика представляет уровень процедур защиты информации:

Таблица 12.1.

Библиотека	**Описание**
ctBaseLib	Защита информации от несанкционированного доступа низкого уровня, управление секретными ключами.
ctMsgLib	Работа с высокоуровневыми сообщениями и пользовательскими сертификатами.
ctCertStoreLib	Обеспечивает унифицированный интерфейс для хранения сертификатов в долгосрочных хранилищах различных типов.

Функции ctBaseLib используют для генерации секретных ключей в контейнеры секретных ключей. Также они могут быть использованы для реализации процедур защиты информации от несанкционированного доступа (таких, как кодирование или подписание) вместо высокоуровневых функций (например, для кодирования записей баз данных), или когда не требуется обмена открытыми ключами и определения отправителя или получателя.

CtMsgLib поддерживает управление пользовательскими сертификатами и высокоуровневыми сообщениями. Сертификат — это подписанный объект, содержащий открытый ключ пользователя, его имя и другие параметры. Он используется для передачи открытых ключей. Сообщение — это конверт стандартного формата, содержащий обработанные данные и другую информацию, зависящую от типа сообщения,

которая может включать параметры алгоритма и данные, необходимые для того, чтобы открыть сообщение. Функции этой библиотеки можно использовать для обработки файлов неограниченного размера.

Функции ctCertStoreLib используют для хранения сертификатов. Объекты-сертификаты могут храниться в хранилищах, расположенных на различного типа физических хранилищах (локальная или удалённая файловая система, база данных, LDAP или любое пользовательское устройство хранения). Отдельные модули — провайдеры физических хранилищ (Physical Store Providers — PSP) — управляют хранилищами определённых типов. Не следует обращаться к PSP напрямую, нужно только прикрепить (зарегистрировать) PSP к ctCertStoreLib и затем использовать унифицированный интерфейс для работы с различными хранилищами сертификатов. «ЛАН Крипто» распространяет PSP для файловых хранилищ (PSPFile), баз данных через ADO (PSPADO), LDAP-хранилищ сертификатов (PSPLDAP); любой может написать собственный PSP для использования в ctCertStoreLib.

Модель программирования Библиотеки:

Разнообразные объекты защиты информации от несанкционированного доступа (контейнеры секретных ключей, ключи, сертификаты, сообщения, хранилища и т.п.) представляются в Библиотеке с помощью контекстов. Контекст включает в себя данные, относящиеся к объекту, и с его помощью производится управление этими данными. Приложения могут осуществлять доступ к объектам по их описателям контекстов. В целом использование объекта защиты информации включает следующие шаги:

- открываем контекст объекта, получая его описательную часть;
- получаем или устанавливаем свойства объекта, выполняем некоторые операции с объектом, передавая его описательную часть контекста процедурам защиты информации;
- закрываем контекст объекта, делая недействительным его описательную часть.

Глава 13. Основная библиотека (ctBaseLib)

Функции базовой библиотеки защиты информации (ctBaseLib) являют собой наиболее гибкие и низкоуровневые средства для разработки приложений защиты информации.

CtBaseLib управляет следующими объектами защиты информации: Контейнер секретных ключей, который ассоциирован с пользователем системы безопасности. Вы можете осуществлять доступ к контейнерам ключей через контексты защиты информации, к которым прикреплены контейнеры ключей.

- Задачи защиты информации (секретность, целостность, аутентификация), которые открываются внутри контекста защиты информации и представляют собой различные алгоритмы. Задача защиты информации управляет всеми данными, требуемыми для выполнения определённой операции защиты данных, например, кодирования или подписи.
- Открытые ключи, необходимые для большинства операций защиты информации. Они передаются среди пользователей системы безопасности в форме капсулы с открытым ключом. Для библиотеки ctBaseLib не имеет значения, каким образом происходит передача и обмен секретных ключей, равно и как гарантирование их подлинности.

Функции ctBaseLib

Алгоритмы защиты информации используют параметры A, P и Q, называемые триплетами APQ. CtBaseLib поддерживает хранение этих триплетов, что является гарантией надежной защиты информации.

Библиотека предоставляет полный набор функций для разработки приложений:

1. Функции для работы с параметрами провайдера включают в себя:

   - функцию,возвращающую значения следующих параметров провайдера ctBaseLib:
   - информация о версии библиотеки
   - информация об указанном алгоритме
   - информация обо всех поддерживаемых алгоритмах

- триплет APQ, ассоциированный с определенным индексом APQ и алгоритмом
- индекс триплета APQ и значения для указанного алгоритма (используя этот параметр, Вы можете установить специальный триплет APQ для данного приложения)
- максимальное количество хэшируемых триплетов APQ
- параметр для освобождения хэшированных триплетов APQ, которые не используются никакими задачами и контекстами
- отладочная информация об открытых контекстах
- отладочная информация о хэшированных триплетах APQ
- функцию, настраивающую параметры библиотеки провайдера, где вы можете изменять значения параметров, приведенных выше.

2. Функции контекста защиты информации (контейнеров секретных ключей):
   Они дают возможность:

   - добавить ссылки в описатель контекста защиты информации
   - создать копию контекста
   - возвратить или настроить параметры контекста
   - управлять операциями с секретным ключом внутри контейнера секретного ключа

3. Функции для управления задачами защиты информации и открытыми ключами, которые дают возможность:

   - создать копию задачи защиты информации
   - возвратить или настроить параметры задачи защиты информации - импортировать (экспортировать) открытый ключ в задачу защиты информации — кодировать (декодировать) буфер с текстом
   - вычислить хэш данных и подписать его значение — проверить значение подписи хэша

   Следующие объекты библиотеки определены как константы ctBaseLib, что дает возможность быстро и просто работать с библиотекой:

- типы алгоритмов : подписи\проверки подписи, кодирования\декодирования, хэширования, неопределенный тип.
- идентификаторы алгоритмов : для следующих алгоритмов предусмотрены свои идентификаторы:

Алгоритмы кодирования: Блочные коды Safer, DES, Triple DES, FEAL, ГОСТ РФ 28147, Wicker98, Nush; поточные коды Vesta-2M, RC4.
Алгоритмы подписи: ГОСТ РФ, DSA, LANS, RSA, Gost.
D1Алгоритмы хэширования: ГОСТ РФ, SHA, SHA-1, MD5, GAS48.
В некоторых ситуациях могут быть полезны следующие идентификаторы:

a. Использовать алгоритм подписи, принятый по умолчанию (хранящийся в контейнере ключа).
b. Использовать алгоритм кодирования, принятый по умолчанию (хранящийся в контейнере ключа).

- режимы кодирования : ECB, CBC, CFB, OFB, а также C, G, GB (в алгоритме ГОСТ 28147).
- стандартные индексы APQ — стандартные триплеты APQ для открытых ключей длиной 512, 1024, 2048, 4096 бит (A, P и Q — параметры, используемые алгоритмами защиты информации).
- принятые по умолчанию идентификаторы секретных ключей.
- параметры контекста защиты информации (напр., такие как информация об алгоритмах, разрешенных для использования в контейнере ключей, или заданные по умолчанию в контейнере ключей алгоритмы подписи),
- параметры задач защиты информации (напр., такие как идентификатор секретного ключа, который нужно использовать в данной задаче)
- коды возврата.

В документации к библиотеке приведены следующие примеры работы с ctBaseLib: перечисление алгоритмов

- получение открытого ключа

- кодирование и декодирование данных
- подписание и верификация данных
- и многие другие примеры.

Глава 14. Библиотека сертификатов и сообщений (ctMsgLib)

Библиотека сертификатов и сообщений (ctMsgLib) предоставляет высокоуровневый интерфейс к процедурам управления сертификатами и обработки сообщений.

Общий обзор:

Библиотека поддерживает все общеиспользуемые сертификационные объекты:

сертификат — подписанные данные, содержащие имя пользователя, открытый ключ, параметры алгоритма и другую информацию. Обычно сертификаты подписывает доверенная организация, называемая Certification Authority (CA) или Центром Сертификации (ЦС). Это обеспечивает секретный способ обмена открытыми ключами между пользователями системы безопасности.

запрос на сертификат — подписанные данные, содержащую информацию, требуемую для создания сертификата. Запрос подписывается пользователем и отправляется в ЦС на обработку. ЦС создаёт сертификат по информации запроса.

CMC запрос/ответ — этот объект используется для взаимодействия с ЦС. Запрос может содержать несколько запросов на сертификацию и набор дополнительных атрибутов. Ответ посылается Центром Сертификации на каждый запрос CMC.

список аннулированных сертификатов (Certificate revokation list, CRL) — подписанные данные, содержащие список сертификатов, которые были аннулированы (отменены) в ЦС и не могут использоваться с момента аннулирования. CRL подписывается ЦС и периодически рассылается пользователям.

список доверенных сертификатов (Certificate trust list, CTL) — капсула с данными, содержащими список сертификатов, которые являются доверенными на использование без проверки подписи. Он должен содержать, как минимум, самоподписанный сертификат ЦС. Каждый пользователь хранит CTL на своём компьютере.

цепь сертификатов — капсула с данными, содержащая несколько сертификатов, представляющими путь сертификации.

цепь объектов — капсула с данными, содержащая внутри набор любых объектов.

Сертификаты, используемые в настоящей библиотеке, полностью удовлетворяет международному стандарту X.509 (стандарт, используемый для хранения дополнительной информации в сертификационных объектах), включая его расширения версии 3.

конверт сообщений — сообщения, содержащие криптографически обработанные данные и связанную с ними информацию, необходимую для его открывания.

CtMsgLib поддерживает определённое количество общеиспользуемых типов (стандарта PKCS #7) содержимого сообщений для хэшированных, подписанных или закодированных данных.

Формат PKCS #7 описывает общий синтаксис сообщения. Этот синтаксис допускает рекурсию, при которой, например, один конверт может находиться внутри другого, или пользователь может подписать данные, хотя они были уже помещены в конверт другим пользователем. Также доступны произвольные расширения — атрибуты, такие как подписанное время, эти атрибуты подписываются вместе с содержимым конверта.

Функции библиотеки ctMsgLib предусматривают управление сертификатами и обработкой сообщений. Они предоставляют возможность:

-открыть новый контекст для объекта указанного типа -добавить ссылку в контекст

-закрыть контекст и освободить захваченную память -получить свойства контекста объекта

-установить свойства контекста объекта

-передавать данные между контекстами, добавляя один объект ко другому -фиксировать и подписывать данные объекта -проверить подлинность полей и подписи объекта -получить информацию о последней ошибке

Следующие структуры ctMsgLib содержат всю необходимую для работы информацию:

- информация об алгоритме сертификата
- информация об OID (уникальный идентификатор алгоритма. Используется для описания алгоритмов с открытым ключом и подписи в сертификационных объектах формата X.509)
- информация о свойстве объекта-сертификата
- информация о результате последнего вызова функции
- информация о сертификате пользователе

К константам ctMsgLib относятся:

- параметры провайдера (напр., информация о версии библиотеки, или о значении OID, возвращаемая по идентификатору OID)
- параметры контекста объекта-сертификата (включает статус контекста, информацию о пользователе, информацию об открытом ключе)
- типы и поля объектов: сертификат, запрос на сертификат, CMC запрос/ответ, список аннулированных сертификатов, список доверенных сертификатов, цепь сертификатов, цепь объектов, конверт сообщений
- типы содержимого сообщений: данные, хэшированный тип, подписанный тип, закодированный тип, закодированный с паролем, заархивированный
- типы полей объектов: булево, большое целое переменной длины, битовая строка, номер, идентификатор объекта (OID), дата и время, текстовая строка, двоичные данные, составное поле
- информация о пользователе
- расширения X.509 (здесь хранится дополнительная информация о сертификационных объектах)
- идентификаторы алгоритмов, значения OID
- коды возврата

С помощью ctMsgLib Вы можете свободно работать с сертификационными объектами и сообщениями. В документации (.chm) к .Библиотеке приведены примеры:
создания сертификата по запросу
создания законченного сообщения
и другие примеры.

- создания запроса на сертификат

Глава 15. Библиотека хранения сертификатов (ctStoreLib)

Библиотека хранения сертификатов Certificate Store Library (ctCertStoreLib) предназначена для вызова функций хранения сертификатов.

Общие принципы:

export: Command not found. Пользователь системы может собрать на своем компьютере множество сертификатов. Как правило, у пользователя хранятся сертификаты издателя (CA), его собственные сертификаты и сертификаты других пользователей. Для каждого пользователя может существовать несколько сертификатов. Для каждого сертификата должна существовать цепочка проверяющих сертификатов, которая обеспечивает подлинность сертификата и правильность данных в нем.

Хранилища сертификатов используются для хранения самих сертификатов и всех объектов, относящихся к этим сертификатам. Функции ctCertStoreLib позволяют создавать хранилища сертификатов, добавлять/находить объекты в/из хранилища. Для выборочного нахождения объекта используется запрос по заданным свойствам объекта.

Хранилища сертификатов могут быть встроены в различные типы физических хранилищ (пример: файловая система, база данных или LDAP). Для каждого типа хранилища существует соответствующий провайдер физического хранилища (Physical Store Provider, PSP) прикрепленный к ctCertStoreLib.

Функции для работы с хранилищами сертификатов предоставляют широкие возможности для работы с хранилищами сертификатов. В библиотеке предусмотрены возможности:

-создать новое или открыть существующее хранилище для последующего доступа -добавить ссылку в контекст хранилища

-освободить описатель хранилища, завершающий текущий доступ к хранилищу -получить параметры библиотеки и хранилища -установить параметры библиотеки или контекста -добавить объект сертификата в открытое хранилище -добавить хранилище в список хранилищ

Константы ctCertStoreLib предусматривают всевозможные параметры для работы с хранилищем сертификатов. Они включают в себя:

параметры провайдера:

информация о версии библиотеки

параметр для регистрации или удаления из списка зарегистрированных типов PSP выходная информация об открытых контекстах хранилища при отладке параметры хранилища:

тип PSP

параметры PSP предопределенные или определенные некоторым PSP отладочная информация об открытых контекстах хранилищ

параметры запроса:

число найденных объектов

разрешение итерации результатов запроса UID в хранилище текущего объекта

коды возврата

В документации к Библиотеке описаны примеры:

- создания хранилища сертификатов
- добавления объектов к хранилищу
- поиск в хранилище

В настоящее время Библиотека является самым надежным и удобным средством разработки приложений, использующих защиту информации.

Часть IV. Редактор шрифтов Pfaedit

Глава 16. Вступление

Pfaedit — редактор шрифтов, который позволяет создавать и редактировать шрифты в различных форматах, а также преобразовывать их из одного формата в другой.

При запуске программы без параметров появляется диалог выбора файла, дающий возможность выбрать файл(ы) шрифтов для редактирования или создать новый.

Форматы шрифтов, поддерживаемые Pfaedit, можно разделить на две группы: растровые и векторные.

К векторным относятся *PostScript* Type1 (PFA/PFB + AFM; GSF), *PostScript* Type0 (PS), TrueType (TTF) и OpenType (OTF), к растровым — BDF. Для хранения разрабатываемых шрифтов Pfaedit использует свой собственный формат Structured Font Database (SFD). Кроме этого, он позволяет импортировать файлы в формате PCF, в нескольких форматах `Mac OS` и растровые шрифты TeX (PK + TFM).

Глава 17. Виды окон

При работе со шрифтами Pfaedit использует несколько различных видов окон: окно шрифта, окно редактируемого символа и окно метрик. Каждое окно содержит меню, содержимое которого, а также результат действия команд различаются в зависимости от ряда условий.

Окно шрифта

В окне шрифта (17.1) отображаются все символы редактируемого шрифта. Оно предназначено для работы со всем шрифтом или с отдельными символами целиком. Наиболее часто выполняемые в этом окне операции — выделение символов и открытие окна символа для редактирования.

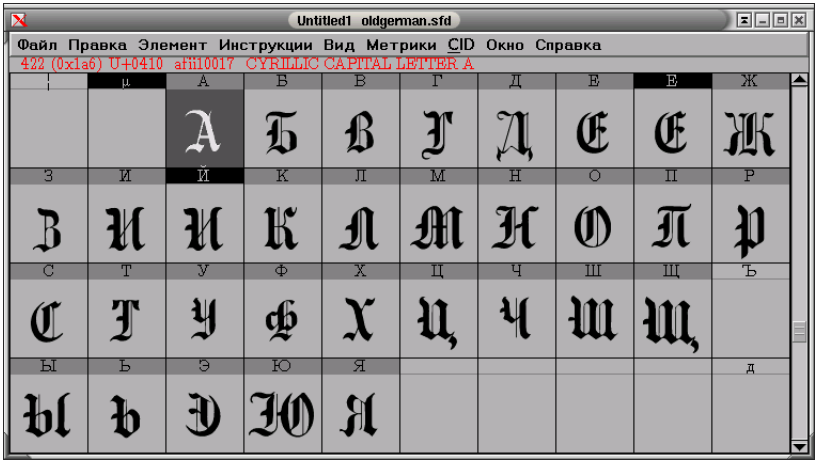


Рисунок 17.1. Окно шрифта

Для выделения одного символа достаточно щёлкнуть на нём левой клавишей мыши; двойной щелчок приведёт к открытию окна редактирования символа. Для выделения нескольких символов подряд надо щёлкнуть на первом из выделяемых символов левой клавишей мыши и, удерживая её нажатой, переместить курсор к последнему выделяемому символу.

Добавлять символы к выделению можно, удерживая нажатой клавишу **Shift**. Повторный щелчок на уже выделенном символе при нажатой клавише **Shift** приведёт к исключению этого символа из выделения.

Окно символа

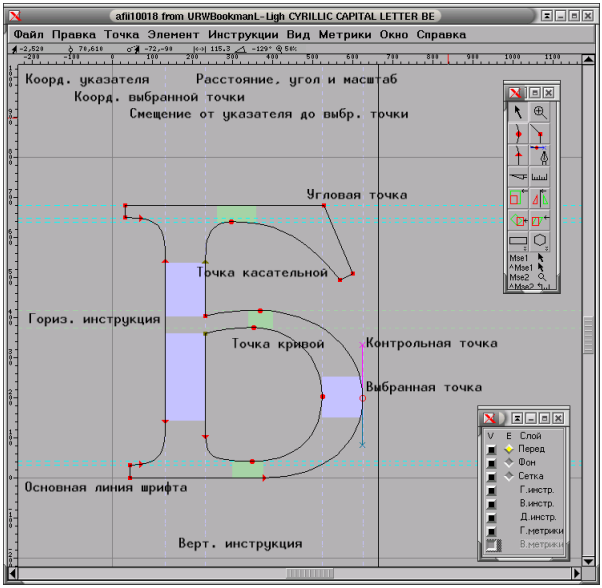


Рисунок 17.2. Окно редактируемого символа (контур)

Окно редактируемого символа содержит контур (17.2) или растр (17.3) символа и позволяет изменять как символ целиком, так и его отдельные части. Вместе с этим окном открываются две панели: со слоями и с инструментами. Состав этих панелей для векторных и растровых шрифтов различен (дальнейшее описание относится к векторным шрифтам).

Панель слоёв позволяет включать/выключать отображение слоёв и выбирать, к какому из них будут относиться последующие команды. Поддерживаемые слои: переднего плана, фона, направляющих, «инструкций» (Type1 hinting или TrueType instructions) разных типов.

Для слоёв инструкций можно только включать/выключать отображение.

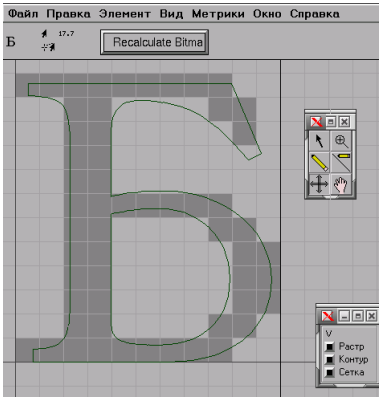


Рисунок 17.3. Окно редактируемого символа" (растровый шрифт)

Панель инструментов содержит 14 инструментов, кратко описанных в следующем подразделе.

Инструменты

«Указатель» позволяет выделять и перемещать точки контура или сами контуры символа. Выделение производится щелчком на нужных точках или контурах или выделением при нажатой левой клавише мыши области содержащей нужные точки. Также щелчком можно выделить правую границу символа. Добавление/исключение точек производится при нажатой клавише **Shift**.

Группу выделенных точек можно:

- перемещать, нажав левой клавишей мыши на одну из них и переместив курсор в нужное место;
- перемещать с помощью клавиш управления курсором;
- подвергать различным геометрическим преобразованиям с помощью других инструментов или выбрав способ преобразования в диалоговом окне пункта «Преобразовать» меню «Элемент» (**Ctrl-**);
- изменять тип (см.ниже);
- удалять, разрывая контур, или «упрощать», сохраняя контур между окружающими «упрощённую» точками.

Щелчок левой клавишей мыши на рабочем поле окна символа уберёт выделение со всех точек.

«Лупа» позволяет изменять масштаб в окне символа. Увеличение производится щелчком в окне символа, уменьшение — щелчком при нажатой клавише **Alt**.

Pfaedit активно использует кнопки мыши с клавиатурными модификаторами. Независимо от выбранного инструмента, левая кнопка мыши в сочетании с нажатой клавишей **Ctrl** работает как «Указатель», средняя — как «Лупа», а средняя в сочетании с нажатой клавишей **Ctrl** — как «Измеритель» (см.ниже). Щелчок правой кнопкой в окне символа вызывает всплывающее меню, позволяющее выбрать один из инструментов.

«Кривая», «угловая» и «касательная» точки и «Перо» позволяют добавлять точки контура в окне символа и различаются типом добавляемой точки («Перо» добавляет точку кривой с контрольными точками). Тип точки можно изменить, выделив её и выбрав подходящий пункт в меню «Точка» или же нажав **Ctrl-2**, **Ctrl-3** или **Ctrl-4** для преобразования в точку кривой, угловую или касательную соответственно.

Нож позволяет разрезать контур между точками. На образующихся концах контура создаются две новые точки кривых, между которыми нет соединения. Чтобы разрезать контур, надо провести курсор в точке разреза, удерживая нажатой левую клавишу мыши. (Часто более удобным оказывается добавить точку на контур и удалить её. Контур при этом оказывается разорван.)

«Измеритель» позволяет измерять расстояние, угол и смещение по осям X и Y от точки где в была отпущена левая клавиша мыши до текущего положения курсора.

Масштабирование позволяет масштабировать выделенную часть символа «на глаз». Масштабирование выполняется перетаскиванием курсора при нажатой левой клавише мыши. Точка, в которой была нажата клавиша, будет являться началом координат для преобразования. Для одинакового масштабирования по обеим осям или для масштабирования только по одной оси необходимо удерживать нажатой клавишу **Shift**.

Зеркало, как следует из названия, позволяет отразить выделенную часть символа по горизонтали или по вертикали. Так же, как и для «Масштабирования», точка, в которой была нажата клавиша, будет являться началом координат для преобразования.

Вращение позволяет вращать выделенную область.

Наклон позволяет наклонять выделенную область.

Для точного масштабирования, вращения и наклона удобнее использовать соответствующий параметр в диалоге «Преобразовать...» меню «Элемент».

Прямоугольник/Эллипс и Многоугольник/Звёздочка позволяют нарисовать соответствующие фигуры. Двойной щелчок на инструменте открывает окно диалога для выбора параметров инструмента (закругление углов прямоугольника, прямоугольник или эллипс, число углов многоугольника/звёздочки, многоугольник или звёздочка).

Окно метрик

Окно метрик (17.4) позволяет изменять различные параметры, относящиеся к сочетаниям символов, а также изменять метрики каждого символа в отдельности.

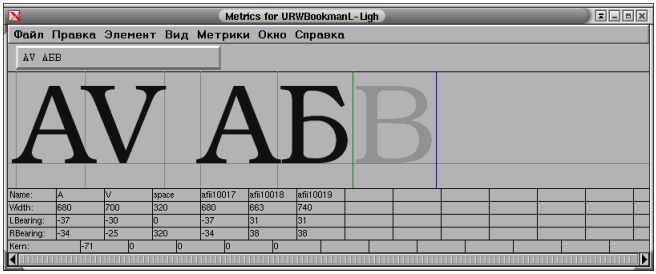


Рисунок 17.4. Окно метрик

Изменить левую/правую границы символа или его ширину, а также задать значение апроша для любой пары символов можно, либо введя значение в соответствующей клетке, либо перемещая мышью символ или его границы. Для того, чтобы упростить работу с кириллическим алфавитом, имеет смысл указать в настройках Pfaedit используемый для ввода набор символов («Файл»→«Настройки...»→«Локальная кодировка», например KOI8-R). Это позволит вводить кириллические символы в любых строках ввода, в частности в окне метрик[3].

[3]По умолчанию строка в окне метрик содержит символы, выделенные в окне шрифта.

Глава 18. Создание шрифта

Чтобы создать новый шрифт, надо при запуске Pfaedit выбрать пункт Создать в окне выбора шрифта (или выбрать пункт «Создать» меню «Файл», или запустить **pfaedit-new**). При этом откроется окно шрифта не содержащее контуров символов. Набор символов вновь создаваемого шрифта можно указать в диалоге «Настройки...» меню «Файл». Если вас не устраивает текущий набор символов, то его можно изменить в диалоге «Информация о шрифте» меню «Элемент» (см. 18.1). В этом же диалоге можно установить название шрифта, его семейство, угол наклона, комментарий и многое другое.

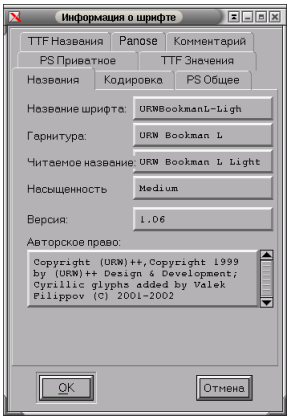


Рисунок 18.1. Информация о шрифте

Если ни один из предлагаемых наборов символов не покрывает всего диапазона необходимых вам символов, то можно выбрать наиболее близкий диапазон, увеличить число символов в шрифте (лист «Кодировка» в диалоге «Информация о шрифте») и дать названия всем дополнительным символам. Чтобы назвать (или переименовать) символ, надо вызвать диалог «Информация о символе» меню «Элемент» (или нажать **Ctrl-I**), в появившемся окне можно ввести символ, его название в соответствии с *Unicode* или его код по *Unicode*. После того, как все необходимые символы названы, можно сохранить получившийся набор, нажав кнопку Создать из шрифта на листе «Кодировка» окна «Информация о шрифте».

Все наборы символов, созданные пользователем, сохраняются в файле `~/.Pfaedit/Encoding.ps`, поэтому для создания нового набора симво-

лов часто бывает удобнее добавлять названия символов прямо в этот файл по образцу.

Глава 19. Создание символа

Для создания символа наиболее часто используются три способа: копирование символа (или создание ссылки на него), автоматическое или ручное оконтуривание изображения (трассировка) и модификация подходящего символа.

Копирование символа; акцентированные символы

Часть символов может быть создана копированием уже существующих.

Например, если вы добавляете в шрифт кириллические глифы, то латинские символы «АВСЕНМОРТХасеорух» могут быть скопированы в соответствующие кириллические. Шрифты *Type1* и *TrueType* поддерживают механизм ссылок, который позволяет не копировать одинаковые символы, а помещать ссылки на уже существующие. Замена копий символа ссылками позволяет уменьшить размер шрифта, а также обеспечить совпадение всех таких «копий» с оригиналом в случае его изменения.

Кроме того, такие символы, как «ЁёЙй», являющиеся акцентированными версиями других символов, могут быть созданы из соответствующих символов и акцентов. Чтобы создать акцентированные символы, достаточно выделить соответствующие ячейки в окне шрифтов и выбрать пункт «Сделать акцентированный символ» меню «Элемент» (или нажать **Ctrl-Shift-A**). Так как шрифты европейских языков содержат достаточно большое количество символов с различными акцентами, то один раз нарисовать каждый акцент, а затем сделать акцентированные символы не только быстрее, но и правильнее, так как акценты у соответствующих символов будут одинаковы.

Оконтуривание вручную

Для того, чтобы обвести какое-либо изображение по контуру, его необходимо сначала импортировать в слой фона символа. Для этого надо открыть окно символа и выбрать пункт «Импортировать...» меню «Файл». Pfaedit позволяет импортировать изображения в форматах BMP, GIF, PNG, JPEG, EPS и XFig. После того как изображение импортировано в слой фона, можно выбрать в качестве рабочего инструмента какой-либо тип точки (удобнее начинать работу с точкой кривой) и последовательно добавить необходимые точки. В том

случае, когда требуется создать более одного контура (например, у О есть внешний и внутренний контуры), надо снять выделение с последней добавленной точки (нажать **Ctrl**, чтобы переключить инструмент на Указатель, и щёлкнуть на свободном месте в окне символа) после того, как текущий контур будет замкнут — и перейти к обводке следующего контура.

Эту операцию можно выполнить автоматически, если в вашей системе установлена программа autotrace.

Автотрассировка

Для автоматического оконтуривания импортируйте изображение и выберите пункт «Автотрассировка» из меню «Элемент». Для того, чтобы получить результат, приемлемый для последующей обработки, необходимо импортировать изображение с достаточно высоким разрешением.

Модификация символов

В большинстве шрифтов многие символы содержат одинаковые или схожие элементы, а некоторые символы даже частично совпадают друг с другом. Например нижние половины у Б, Ь, Ы и Ъ во многих шрифтах будут одинаковы, а Р и Ь можно будет получить из друг друга зеркальным отражением. Кроме того, у латинских, греческих и кириллических шрифтов вертикальные основные элементы должны быть абсолютно одинаковы, в противном случае возможно возникновение очень заметных искажений при растрировании.

Таким образом, «склеивание» символов из соответствующих элементов других символов не только удобно, но и позволяет достичь более правильного с технической точки зрения результата (в отдельных случаях элементы, которые кажутся одинаковыми, должны быть разными из соображений дизайна шрифта; рассмотрение вопросов дизайна выходит за рамки данного текста). Для создания нового символа путём модификации уже существующего необходимо скопировать модифицируемый символ через буфер обмена: выделить исходный символ, выбрать «Скопировать» в меню «Правка» или нажать **Ctrl-C**; выделить ячейку создаваемого символа, выбрать «Вставить» в меню «Правка» или нажать **Ctrl-V**. Если к уже существующим частям символа необходимо добавить части какого-то другого символа, то вставку необходимо производить в окне создаваемого символа, а не в окне шрифта, иначе уже имеющиеся части создаваемого символа будут заменены вставляемыми.

Другие способы создания символов

Помимо описанных, возможно непосредственное рисование символов.

Для этого Pfaedit предлагает следующие функции: «Расширить штрих» и «Удалить пересечения» в меню «Элемент». Первая позволяет заменить созданную из точек кривую повторяющим её форму контуром заданной ширины, а вторая — объединить пересекающиеся контуры.

Обработка символов

Созданные символы можно (и нужно) дополнительно обработать. Pfaedit предлагает следующие функции обработки: «Упростить», «Добавить экстремумы», «Исправить направление». Все они доступны в меню «Элемент». Любые изменения можно контролировать, скопировав контур символа в слой фона (пункт «Скопировать передний план в фон» в меню «Правка» или **Ctrl-Shift-C**).

Pfaedit позволяет добавлять инструкции (hints) для растеризатора *Type1* и *TrueType* в ручном или автоматическом режиме. Для запуска автоматической генерации инструкций выберите пункт «Автоинструкции» меню «Инструкции». В настоящий момент качество автоматически генерируемых инструкций для *TrueType* недостаточно высокое. Для создания инструкций *TrueType* предназначена программа ttfmod (в настоящий момент позволяет только просматривать существующие инструкции).

Pfaedit позволяет проверить, что шрифт (или выбранные символы) не содержат часто возникающих ошибок. Проверка запускается при выборе пункта «Найти проблемы» меню «Элемент».

Глава 20. Скрипты

Уникальной чертой Pfaedit является предлагаемая им возможность писать сценарии на собственном языке, похожем на смесь C, языка программирования командной оболочки (bash) и команд из меню Pfaedit.

Из скрипта можно выполнить любую функцию, относящуюся к шрифту (открыть, перекодировать, добавить символы, сгенерировать шрифт) или к символу (скопировать, отразить, повернуть, изменить границы и так далее). Таким образом с помощью сценариев можно автоматизировать большую часть выполняемых со шрифтами действий.

Технологии разработки WEB-проектов с использованием серверов приложений

История переиздания
Издание 0.1 22 Oct 2002
Отформатирована в DocBook версия статьи, взятая с сайта http://www.neural.ru - последняя актуальная версия автора.

Общие принципы

Программное обеспечение интернет-проектов должно удовлетворять ряду требований, таких как максимально возможное разделение трех составляющих любого интернет-проекта: исполняемого кода, дизайна веб-интерфейса и информационного наполнения ресурса, что дает возможность изменения любой из компонент в достаточно широких пределах, не подвергая опасности работоспособность ресурса в целом.

Разделение подразумевает не только возможность раздельной разработки, но и возможность отчуждения любой составляющей, что обеспечивает выполнение следующего требования - повторного использования решений. В самом деле, какими бы уникальными ни были ресурсы и бизнес-процессы, они строятся из базовых элементов, ряд которых практические не изменяется от проекта к проекту.

Необходимость повторного использования влечет за собой требование функциональной полноты и замкнутости решения, что означает полное покрытие программным кодом возможных управляющих воздействий и решение всех задач управления решением в рамках самого этого решения.

Используемые инструментальные средства

В последее время, появился ряд инструментальных средств, ориентированных на поддержку разработки интернет-проектов при соблюдении вышеприведенных требований. Хотя среди их производителей можно упомянуть такие известные названия как Software AG и

Oracle, производящие хорошо масштабируемые и интенсивно разрабатываемые продукты, необходимо отметить: для 90% российского рынка интернет-проектов использование настолько масштабных решений неоправдано, ибо влечет за собой существенные финансовые затраты без получения существенных преимуществ. Бесспорно, Oracle - более мощная база данных, чем MySQL. Но не менее бесспорно то, что для поддержки базы данных из сорока таблиц длиной не более нескольких тысяч записей MySQL более чем достаточен. При этом накладные расходы на использование MySQL (лицензирование, контракты с хостинг-провайдерами) существенно ниже.

На сегодняшний день, в зависимости от требований клиента можно предложить один из трех вариантов архитектуры интернет-ресурса, основанных на свободно-распространяемом ПО. Это CGI-скрипты и сервера приложений - Midgard и Zope

Пакет CGI-скриптов, обеспечивающих необходимую функциональность

В целях пояснения дальнейшего изложения преимуществ серверов приложений, опишем гипотетический универсальный модуль языка Perl, позволяющий создавать и поддерживать интернет-ресурс как тройку вида

<оформление, информационное наполнение, метод доступа> .

Информационное наполнение хранится в реляционной базе данных (как правило, MySQL), откуда извлекается по запросу и заполняет слоты темплейтов, формирующих концепцию оформления. Практически, модуль может представлять собой удачное объединение трех общеизвестных модулей: HTML::Template, CGI, MySQL, допускающее использование ряда взаимозаменямых модулей, реализующих стандартные концепции веб-интерфейсов.

Идея модуля следующая: разобрать http-запрос и выполнить соответствующие действия, разобрать темплейт, определить свободные слоты, запросить заполнение слотов из базы данных, заполнить слоты и отобразить веб-страницу. Реализация существенно сложнее этого краткого описания, но это вполне работоспособное решение, хорошо зарекомендовавшее себя во многих проектах.

Возвращаясь к взаимозаменяемым модулям отметим следующее: использование такого модуля в новом проекте сводится лишь к дизайну нового интерфейса, затраты на переработку самого модуля близки к нулю. Могут быть созданы такие модули как "Гостевая книга", "Доска объявлений", "Каталог услуг", "Корзина покупателя" и другие.

Несмотря на изрядный возраст технологии CGI-скриптов, такой подход дает возможность создавать удачные интернет-ресурсы при минимальных требованиях к контракту между пользователем и хостинг-провайдером - даже самые дешевые варианты контрактов подразумевают предоставление CGI-каталога.

Сервер приложений Midgard

Технология PHP , используемая данным продуктом, имеет ряд неотъемлемых преимуществ, таких как высокая скорость обработки запросов и ориентированность на создание веб-приложений. Однако, из-за невозможности декомпозиции на дизайн и програмный код, PHP больше подходит для индивидуальных разработчиков разовых проектов, а не корпоративного применения. Несмотря на недостатки использования PHP в чистом виде, его использование в составе таких средств, как сервер приложений Midgard, может быть крайне эффективным. Midgard содержит встроенные средства управления информационным наполнением и динамической генерации страниц, что дает возможность легкой для пользователя поддержки крупных контент-проектов. Встроенный язык программирования - PHP - позволяет решать достаточно сложные задачи представления информации, причем оригинальная архитектура Midgard снимает большую часть присущих PHP проблем.

Midgard представляет интернет-ресурс как три древовидных пространства имен: Стиль (дизайн), Адресное пространство (оформление и размещение), Публикуемые материалы (информационное наполнение). В первых двух пространствах могут быть определены стилевые и страничные элементы, причем такой элемент вставляется в веб-страницу посредством синтаксиса вида "<[" + <ИМЯ_ЭЛЕМЕНТА> + ">". Midgard предоставляет средства управления видимостью элементов, позволяя легко варьировать оформление страниц перекрытием имен. С другой стороны, любой элемент оформления пишется один и только один раз, что дает возможность быстрого изменения дизайна сайта переопределением элементов оформления.

Доступ к публикуемым материалам осуществляется посредством PHP-скриптов на страницах Адресного Пространства, формируя тем самым развитый аналог технологии, описанной выше для случая CGI-скриптов. Публикуемые материалы имеют строго фиксированную структуру (дерево каталогов типизированных статей в ранних версиях Midgard, к которым в более поздних версиях добавились прикрепленииия к статьям и отношения между статьями), поэтому после достижения договоренности о размещении материалов, с точки зрения

контент-менеджера поддержка интернет-ресурса созданного на базе Midgard укладывается в привычную абстракцию, сходную с размещением данных в дереве каталогов, а добавление или изменение материалов не представляет сложностей.

К сожалению, декомпозиция проекта на програмный код, информационное содержание и оформление представляет скорее соглашение, чем неотъемлемую часть архитектуры, что приводит к серъезным трудозатратам на планирование интерфейса.

Midgard навязывает разработчикам древовидную модель данных и накладывает промежуточный уровень абстракции на интерфейс с базой данных, что ограничивает возможность формировать запросы и размещать данные произвольного формата, поэтому при поддержке информационных ресурсов, связанных с достаточно сложным поиском данных, использование Midgard не дает существенных преимуществ: практически, если информационное наполнение интернет-сайта не удается отобразить на диктуемую Midgard модель, то его использование едва ли облегчает разработку и поддержку ресурса. Но, если рассматривать Midgard не как универсальное средство поддержки интернет-ресурсов, а как средство расширения возможностей PHP, обеспечивающее единый интерфейс для редактирования информационного наполнения и управление отображением данных, то Midgard - безусловно, очень удачное решение, облегчающее эксплуатацию однажды созданного интернет-ресурса.

Сервер приложений Zope

Сегодня существует более продвинутое решение в области серверов приложений - Z Object Publishing Environment (ZOPE). Zope представлет собой мощную и чрезвычайно гибкую среду разработки и поддержки, которая содержит коннекторы к подавляющему большинству распространенных баз данных, средства автоматического контроля версий, мощную схему разделения доступа, возможность отчуждения и репликации любых фрагментов сайтов, встроенные средства поиска и индексирования содержимого и многое другое, что позволяет делать на его основе все. К сожалению, для российской действительности, практически неразрешимая для Zope проблема - размещение сервера у хостинг-провайдера.

Проведенный опрос московских провайдеров дал неутешительные результаты: в тот период, когда python уже входил в десятку первых проектов на www.sourceforge.net, являлся частью процедуры инсталляции RedHat и использовался при конфигурировании ядра Linux, на

вопрос о возможности использование python подавляющее большинство провайдеров дало ответы типа: "наверное, вам нужна NT, а у нас Linux RedHat 6.1". Похоже, единственный вариант контракта с провайдером, допускающим возможность использования Zope - услуга co-location, что приемлимо лишь для достаточно крупных и дорогостоящих проектов.

Технически Zope представляет собой объектно-ориентированную среду, реализующую поверх Python все ту же базовую идею заполнения слотов темплейта данными, извлеченными по запросу пользователя. В качестве языка темплейтов используется DTML, которого в подавляющем большинстве случаев достаточно для извлечеия и отображения данных. В остальных случаях можно создавать массивные процедуры на Python и импортировать их в Zope . Как и в случае Midgard , проектирование интернет ресурса требует хорошей начальной проработки для определения списка используемых объектов и отношений наследования между ними: любое информационное наполнение в Zope представляется как объекты определенного типа, такие как "DTML-документ", "DTML-метод", "Рисунок", "Каталог" или типы, определяемые пользователем (скажем, для логотипов или баннеров отображаемых на страницах, можно определить типы "Logo" и "Banner"). Далее следует не менее сложная стадия программирования и определения пользовательских типов: но результат оправдывает затраченные усилия. Тип Logo, Banner, ZDiscuss (доска объявлений) - это готовые, легко тиражируемые решения, затраты на повторное использование которых близки к нулю.

Выводы

Изложив общие декларации, попробуем точно ответить на следующий вопрос: что дает использование такого подхода к разработке интернет-ресурсов?

1. Декомпозиция проекта на исполняемый код, дизайн и информационное наполнение - означает в первую очередь то, что ваша секретарша решив исправить название поля в элементе диалога с "От:" на "Отправитель:" не нарушит работоспособность интернет-ресурса, забыв поставить точку с запятой после оператора;
2. Возможность повторного использования - означает, что каждый последующий ресурс будет создаваться быстрее, качественнее и дешевле чем предыдущий, ибо многие его составляющие будут уже готовы;

3. Адекватность решений поставленной задаче - означает, что клиент любого уровня, получит работоспособное решение именно тех проблем, которые его интересуют по приемлемой цене;

ПРИЛОЖЕНИЕ А

Приведенные в статье общие размышления можно проиллюстрировать реализацией простого примера с использованием различных средств. Примером будет список логотипов выводимых в каком-либо месте страницы - например, вдоль нижнего края. Каждый логотип характеризуется следующими атрибутами:

1. Изображение
2. Ссылка

В результате нужно получить на странице фрагмент HTML-кода примерно следующего содержания:

```
<P>
      <A HREF="ссылка-1"> <IMG SRC="URL-изображения-1"> </A>
      ... несколько повторений ...
</P>
```

Причем должна быть возможность легкого изменения дизайна приведенного фрагмента и списка логотипов.

CGI-скрипт & HTML::Template

За отсутствием серьезных альтернатив для более масштабных задач, будем хранить логотипы в базе данных *MySQL*в таблице, определяемой строкой:

```
CREATE TABLE logo (
      id int primary key auto_increment,
      img varchar(32),
      ref varchar(32)
);
```

Для отображения будем использовать следующий фрагмент кода HTML::Template :

```
..... какое-то начало ....

<P>
<TMPL-LOOP name="logo_list">
       <A HREF="%ref%"> <IMG SRC="%img%"> </A>
</TMPL-LOOP>
</P>
..... какой-то конец ....
```

Для получения данных из базы создадим perl-модуль под названием LOGO, который будет содеражать следующий фрагмент кода:

```
if ($self->query(logo_list)) {
$self->htmltable("select ref,img from logo",logo_list,ref,img)
}
```

(Позже мы поясним почему нужен "целый модуль" для одной строки кода)

Все, задача практически решена: после небольшого тестирования HTML::Template-код можно отдать дизайнеру, который из вышеприведенного незамысловатого фрагмента сделает конфетку. Как нетрудно видеть, дизайнер не выходит за пределы привычной ему концепции HTML-тегов: просто появились два новых тега. Нарушить программный код дизайнер не может, т.к. не имеет к нему доступ.

Но за этими победными словами кроется подводный камень: приведенный фрагмент кода функционально не полон. Как разместить новые логотипы в списке? Открыть Unix-shell, запустить консольный клиент MySQL и начать писать SQL-запросы? Вполне возможный вариант, хотя на самом деле, вышеупомянутый "целый модуль" придется существенно доработать, добавив средства редактирования таблицы логотипов. Это нетрудно - но выходит за рамки нашего примера. Результатом будет функционально полное решение, позволяющее вставлять в CGI-скрипт возможность отображать строку логотипов, написав команду вида:

```
$logo = LOGO->new(tablename=>"logo", edit=>"1");
```

Как видим, объект здесь - строка логотипов, методы которого - отображение, редактировние и т.п.

Midgard

Хотя можно хранить список логотипов в таблице базы данных, но, здесь есть лучшая альтернатива: создать каталог в пространстве "Публикуемых материалов", статьи которого будут определениями отдельных логотипов. Пусть это будет каталог No.54 (в новых версиях *Midgard*возможна ссылка по имени). В каталоге 54 разместим статьи, причем в атрибут extra1 этой статьи запишем url изображения, а в атрибут extra2 - url ссылки. Чбы отличать логотипы от всех других статей в этом каталоге, присвоим им тип номер 12. Несмотря на то, что при этом используется интерфейс Midgard и который позволяет создать статьи в каталоге, на самом деле вносятся данные в аналогичную таблицу MySQL, но ее поля были предопределены разработчиками (если не хватило предопределенных полей - то об этом отдельный разговор, начиная с версии Bifrost).

Для отображения указаного списка логотипов требуется в адресном пространстве поместить следующий фрагмент кода:

```
<P>
<?
      $article = mgd_list_topic_articles(54,"order",12);
      if ($article)
            while($article->fetch()) {
            $article_item = mgd_get_article($article->id);
?>         <A HREF="&(article_item.extra2);">
            <IMG SRC="&(article_item.extra1);"> </A> <?
      }
?>
</P>
```

Как видим, о декомпозиции кода и дизайна речи пока не идет. К счастью, есть выход: определить участки HTML-кода в виде стилевых элементов и разместить их в пространстве Стилей. Тогда фрагмент кода изменится:

```
<[LOGO_head]>
```

```
<?
       $article = mgd_list_topic_articles(54,"order",12);
       if ($article) while($article->fetch()) {
              $article_item = mgd_get_article($article->id);
?>          <[LOGO_item]>  <?
       }
?>

<[LOGO_tail]>
```

А в таблице стилей появятся следующие элементы:

```
LOGO_head -- <p>

LOGO_item -- <A HREF="&(article_item.extra2);">
<IMG SRC="&(article_item.extra1);"> </A>

LOGO_tail -- <p>
```

Кажется, нужно еще сказать что-то о повторном использовании? Хорошо, создадим страничный элемент LOGO_list, следующего содержания:

```
<[LOGO_head]>

<?
       $article =
              mgd_list_topic_articles($article_topic,"order",12);
       if ($article) while($article->fetch()) {
       $article_item = mgd_get_article($article->id);
?>      <[LOGO_item]>  <?
       }
?>

<[LOGO_tail]>
```

А на самой странице разместим следующий код:

```
<? $article_topic = 54; ?>
```

```
<[LOGO_list]>
```

Вот, в принципе готово. Объяснив дизайнеру какие элементы можно редактировать и зачем, можно ожидать что дизайнер сделает дизайн.

Несмотря на технологические затруднения при разделении труда разработчиков, необходимо отметить отсутствие необходимости создания обвязки для достижения функциональной полноты. Все уже есть - контент-менеджер может спокойно положить статью в каталог и забыть о ней. Причем интерфейс для контент-менеджера вполне удобный, это не командная строка mysql....

Zope

Теперь все тоже самое, только в *Zope*. Опять-таки, будем хранить объект в рамках самого *Zope*, правда, на этот раз это будет действительно объект - объект, для которого помимо свойств определим метод: tag. Не останавливаясь на подробностях создания объекта, скажем лишь что класс объекта называется Logo. В отличие от двух предыдущих случаев будем хранить изображение в самой базе данных, породив класс Logo от встроенного класса image. Хотя метод render можно написать на языке Python, в данном случае будет проще воспользоваться самим DTML. Назначение метода - вернуть строку, отображающую логотип. Вот он:

```
<A HREF="<dtml_var ref>"> <dtml_var "tag(border=0)">" </A>
```

(Нам не потребовался тег img, потому что сам по себе объект Image уже делает операцию отображения изображения используя метод tag)

Создадим для объекта Logo вид по умолчанию, вызывающий этот метод. Теперь в каталоге Logo создадим несколько объектов типа Logo, а в нужном месте вставим следующий DTML-код:

```
<dtml-in "Logo.objectItems()">
      <dtml-var sequence-item>
</dtml-in>
```

Все, готово. Нашему дизайнеру, как и в *Midgard*, нужно лезть в разные фрагменты, однако здесь при сохранении всех преимуществ

Midgard, мы получаем еще одно: инкапсуляцию. Что будет если в каталог Logo мы положим объект класса Image ? Мы получим картинку без ссылки. А объект класса DTML-document? Вставленный текст документа. Кроме того, мы можем аналогично Logo определить более сложный тип Banner, причем, это происходит незаметно для дизайнера (он лишь определяет метод render()) и прозрачно - для контент-менеджера (в списке классов объектов, которые он может добавить в каталог появляется еще один). Заметим еще одну немаловажную деталь: несколькими манипуляциями мышью мы можем экспортировать определение объекта и сам объект в виде переносимого формата (XML или ZEXP) и импортировать в другой сайт - вот и повторное использование. Функциональная полнота обеспечивается за счет наследования при создании типа объекта.

Часть V. Разработка программ для PalmOS под PalmOS

Глава 21. Введение

В этой статье я хотел бы остановиться подробнее на разработке программ для PalmOS в PalmOS. Я использую дистрибутив *ALT Linux* Master 2.0 с настроенным репозитарием пакетов *Sisyphus*[4], откуда можно легко установить большинство необходимых программ. Конечно, все используемые программы можно скачать в интернете, и использовать бесплатно.

Я не буду подробно описывать технологию программирования в PalmOS — для этого много документации можно найти в Интернете. Главными справочниками разработчика являются *Palm OS Companion and Reference*[5]. На том же сайте вы можете найти много полезной документации, предназначенной для разработчиков: http://www.palmos.com/dev/support/docs/. Неплохой форум, посвященный разработке под PalmOS можно найти на сайте http://pascal.sources.ru/cgi-bin/forum/YaBB.cgi?board=palmos. Из конференций (*news-groups*) могу порекомендовать сервер news://news.falch.net — конференция pilot.programmer.gcc (на английском). Хороший каталог ресурсов по разработке для Palm на *OpenNet*[9].

Кратко перечислю используемые программы:

PalmOS SDK

библиотеки и заголовочные файлы;

prc-tools

собственно кросс-компилятор gcc и другие инструменты для компиляции и сборки программ под PalmOS;

pose

эмулятор Palm. Позволяет отлаживать программы с помощью m68k-palmos-gdb, без риска потерять все данные на реальном устройстве;

pilrc

компилятор ресурсов для PalmOS. Преобразует rcp (текстовый файл описания ресурсов) в бинарные файлы формата PalmOS

[4] http://www.altlinux.ru/index.php?module=sisyphus
[5] http://www.palmos.com/dev/support/docs/palmos50-html.zip
[9] http://www.opennet.ru/prog/sml/71.shtml

guikachu

графический редактор ресурсов `PalmOS` (на базе gtk/gnome). Позволяет «рисовать» интерфейс приложений `PalmOS`

ddd

графический frontend к отладчику gdb, включая m68k-palmos-gdb;

gvim

редактор исходных текстов. Подойдет любой привычный редактор или среда;

Глава 22. PalmOS SDK

Для работы с gcc надо на странице `http://www.palmos.com/dev/tools/sdk/` перейти в раздел нужной версии SDK и скачать архив в разделе «Unsupported Palm OS SDK 5.0 for PRC-Tools». Я надеюсь, что rpm установит файлы куда надо, а для архива `tar.gz` надо делать так:

1. создать каталог `/usr/local/palmdev/sdk35` и линк на неё — `/usr/local/palmdev/sdk` (что-бы легко переключаться между версиями SDK):
   mkdir /usr/local/palmdev/sdk35
   ln -s /usr/local/palmdev/sdk35 /usr/local/palmdev/sdk
2. Из архива скопировать содержимое каталога `Palm OS 3.5 Support/Incs` в каталог `/usr/local/palmdev/sdk35/include`, и `Palm OS 3.5 Support/GCC Libraries` — в `/usr/local/palmdev/sdk35/lib`

Документацию можно найти на в разных форматах. Нам больше подойдет вариант в HTML или PDF.

Глава 23. Prc-tools

Скачать последнюю версию можно с домашней страницы *prc-tools*[11]. На текущий момент это версия `2.2`, которая поддерживает последние модели Palm-based устройств на ARM-процессорах и PalmOS SDK 5.

В пакет `prc-tools` входят программы:

build-prc	строит из бинарных файлов и ресурсов файл .prc готовый к установки на Palm;
m68k-palmos-ar	позволяет создавать свои библиотеки
m68k-palmos-c++	для Palm можно писать программы на *C++* (это тема отдельного разговора и многочисленных споров);
m68k-palmos-gdb	отладчик программ. Можно использовать линк с реальным устройством или эмулятором POSE;

Другие инструменты

`prc-tools` во многом повторяет стандартный пакет `gcc`, и изначально являлся хаком `2.95` версии gcc. Поэтому команды и ключи `prc-tools` не покажутся необычными программисту, работавшему с gcc. Специфичные для `PalmOS` опции можно посмотреть на *соответсвующей*[12] странице.

[11] `http://prc-tools.sourceforge.net/`
[12] `http://prc-tools.sourceforge.net/doc/prc-tools.html`

Глава 24. POSE

POSE это эмулятор PalmOS-based машин. Существует версия для UNIX, MacOS, Windows. Исходники можно найти на *соответствущей*[13] странице. Для дистрибутивов *ALT Linux* достаточно дать команду **apt-get install pose**. Так-же доступны пакеты pose-skins и pose-doc. Пакет pose-skins нужен для корректной работы с конкретными моделями Palm. Доступны шкурки для Palm, Handspring, Workpad, HandEra разных моделей.

Для работы понадобиться ещё и пакет rom-image, который по лицензионным ограничениям распространяться в составе дистрибутива не может. Для получения *ROM Image Files* необходимо подписаться на *Palm OS Developer Program*[14]. Также можно скопировать ROM с самого устройства Palm. Для этого надо:

1. Установить и настроить пакет pilot-link (см. соответсвующий раздел в «Руководстве пользователя»);
2. Установить на Palm программу getrom2.prc:
   pilot-xfer -i /usr/share/pilot-link/getrom2.prc
   Для старых моделей, с объёмом ROM в 512K6 можно использовать getrom.prc;
3. Запустить на «пальме» установленную программу getrom и нажать кнопку Start. В PalmOS запустить программу pi-getrom;
4. Полученный pilot.rom можно загружать в POSE.

> **Замечание**
>
> На последних моделях Palm getrom2.prc использовать нельзя. Достаточно нажать кнопку синхронизации и запустить pi-getrom.

Теперь можно запустить и настроить POSE. В открывшемся окне POSE нажатием правой кнопки мыши вызывается меню. Выберите пункт «New» и заполните поля:

- В поле «ROM file» надо открыть пункт Other и выбрать файл, скачанный с помощью pi-getrom;
- В поле «Device» появиться номер модели;
- В поле «Skin» выберите шкурку;

[13] http://www.palmos.com/dev/tools/emulator/
[14] http://www.palmos.com/dev/tools/emulator/#roms

- В поле «RAM size» — размер памяти вашего виртуального «палма»;

Вот как выглядит у меня POSE, настроенный на Palm m505:

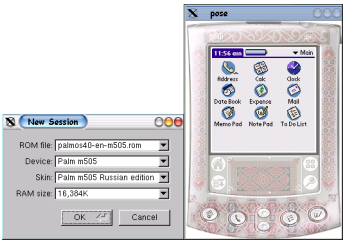


Рисунок 24.1. POSE, настроенный на Palm m505

Теперь можно устанавливать и отлаживать программы.

Глава 25. Guickachu и pilrc

Программа для PalmOS состоит из ресурсов. Сам код является специфичным типом ресурса. Также как и текст программы, ресурсы описываются текстовым файлом (стандартное расширение .rcp), который можно редактировать напрямую в текстовом редакторе.

Для визуального редактирования ресурсов существуют такие программы как pilrcedit и guikachu. *Pilrcedit*[15] — это *Java*-приложение, и мне показалось не слишком удобным. Guikachu написан под *Gtk+/GNOME*, использует XML, как формат сохраняемых файлов, и активно развивается. Последняя стабильная версия на текущий момент — 1.2.3. Скачать можно с *домашней страницы*[16]. Для дистрибутивов *ALT Linux*, guikachu можно установить командой **apt-get install guikachu**. Чтобы преобразовать исходный файл guikachu в RCP-файл, достаточно дать команду **guikachu2rcp name.guikachu** или выбрать пункт «Экспорт RCP» из меню «Файл».

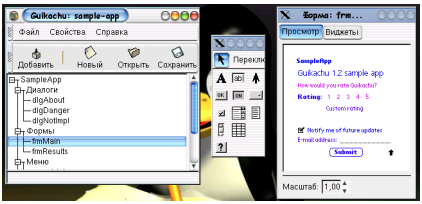


Рисунок 25.1.

Guikachu поддерживает следующие типы ресурсов:

- Форма;
- Диалог;
- Меню;
- Строка;
- Списки строк;
- Текстовые поля;

В формы можно добавлять: статический текст, поля, кнопки, списки, таблицы, пользовательские элементы. Интерфейс достаточно понятен и удобен.

[15] http://www.wn.com.au/rnielsen/pilrcedit/
[16] http://cactus.rulez.org/projects/guikachu

После получения rcp файла, его необходимо преобразовать в бинарный формат. Для этого служит программа *pilrc*[17] (пользователи *Sisyphus* могут ее установить командой **apt-get install pilrc**). Синтаксис достаточно прост: **pilrc -H name.h name.rcp**. По этой команде создадутся необходимые .bin-файлы, и заголовочный файл с определениями идентификаторов.

Замечание

Все идентификаторы ресурсов в `PalmOS` должны быть числовыми, но в программе удобнее пользоваться говорящими именами. Ключ *-H* позволяет автоматически сгенерировать заголовочный файл с "#define Имя Число", давая возможность в программе использовать конструкции типа "case MY_BEST_BUTTON:" вместо "case 9991:".

[17] `http://www.ardiri.com/index.php?redir=palm&cat=pilrc`

Глава 26. Отладчик DDD

m68k-palmos-gdb является версией стандартного gdb, пропатченного под Palm, поэтому можно использовать графические оболочки к gdb. Здесь я расскажу про отладку под ddd.

Чтобы программа собралась с отладочной информацией, надо добавить ключ *-g* в вызове m68k-palmos-gcc. Размер окончательного `.prc`-файла увеличится мало — добавиться только заглушка, передающая управлению внешнему отладчику на хосте. Вся отладочная информация записывается в объектный модуль, который и надо загружать в отладчик.

Для запуска ddd в качестве frontend к m68k-palmos-gdb надо дать ключ *--debugger m68k-palmos-gdb*:

ddd --debugger m68k-palmos-gdb

Первым делом стоит загрузить (**Ctrl-O**) объектник с отладочной информацией (тот файл, который указан в опции *-o m68k-palmos-gcc*). Чтобы стартовать сессию отладки в окне gdb надо дать команду **target pilot localhost:6414**. По указанному порту POSE общается с отладчиком.

> ## Замечание
>
> Для упрощения, можно создать в ddd кнопку. Для этого надо открыть меню «Commands»→«Edit Buttons» и в пункте «Console Buttons» ввести: `target pilot localhost:6414`. После этого над окном отладчика появиться кнопка с введенной командой.

Теперь отладчик ждет, когда на pose запустят программу. При старте в эмуляторе программы, управление немедленно передастся в ddd, и экран POSE останется черный, пока не пойдет вывод из программы. Теперь можно как обычно устанавливать точки прерывания, просматривать значения переменных и т.д. Вот как выглядит сессия отладки:

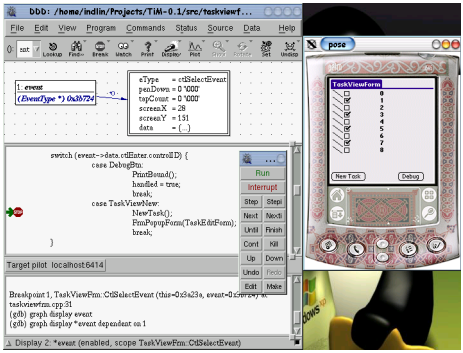

DDD: /home/indin/Projects/TiM-0.1/src/taskviewf...
File Edit View Program Commands Status Source Data Help
pose
TaskViewForm
New Task
Debug
Run
Interrupt
Step
Stepi
Next
Nexti
Until
Finish
Cont
Kill
Up
Down
Undo
Redo
Edit
Make
Target pilot localhost:6414
(gdb) graph display event
(gdb) graph display *event dependent on 1
Display 2: *event (enabled, scope TaskViewFrm::CtlSelectEvent)

Рисунок 26.1.

Глава 27. Среда разработки

Под PalmOS можно собрать для себя эффективную среду разроботки. Выбор довольно широк: vi, emacs, Anjuta, KDevelop. Все эти программы можно настроить для разработки под PalmOS. Универсальным и мощным средством работы является make. .

Перевод на русский язык лицензии GNU на свободную документацию

История переиздания
Издание 0.1 9 Aug 2001

Текст GFDL на английском языке вы можете прочитать здесь: `http://www.gnu.org/copyleft/fdl.html`

GNU Free Documentation License

История переиздания
Издание 1.1 март 2001 г.

Каждый вправе копировать и распространять экземпляры настоящей Лицензии без внесения изменений в ее текст.

0. Преамбула

Цель настоящей Лицензии — сделать свободными справочник, руководство пользователя или иные документы в письменной форме, т.е. обеспечить каждому право свободно копировать и распространять как с изменениями, так и без изменений, за вознаграждение или бесплатно указанные документы. Настоящая Лицензия также позволяет авторам или издателям документа сохранить свою репутацию, не принимая на себя ответственность за изменения, сделанные третьими лицами.

Настоящая Лицензия относится к категории «copyleft» [19]. Это означает, что все произведения, производные от документа, должны быть свободными в соответствии с концепцией «copyleft». Настоящая Лицензия дополняет General Public License GNU, которая является лицензией «copyleft», разработанной для свободного программного обеспечения.

Настоящая Лицензия разработана для применения ее к документации на свободное программное обеспечение, поскольку свободное программное обеспечение должно сопровождаться свободной документацией. Пользователь должен обладать теми же правами в отношении руководства пользователя, какими он обладает в отношении свободного программного обеспечения. При этом действие настоящей Лицензии не распространяется только на руководство пользователя. Настоящая Лицензия может применяться к любому текстовому произведению независимо от его темы или от того, издано ли данное произведение в виде печатной книги или нет. Настоящую Лицензию рекомендуется применять для произведений справочного или обучающего характера.

1. Сфера действия, термины и их определения

Условия настоящей Лицензии применяются к любому руководству пользователя или иному произведению, которое в соответствии с уведомлением, помещенным правообладателем, может распространяться на условиях настоящей Лицензии. Далее под термином «Документ» понимается любое подобное руководство пользователя или произведение. Лицо, которому передаются права по настоящей Лицензии, в дальнейшем именуется «Лицензиат».

«Модифицированная версия Документа» — любое произведение, содержащее Документ или его часть, скопированные как с изменениями, так и без них и/или переведенные на другой язык.

[19] Термин «copyleft» используется авторами проекта GNU Free Software Foundation в качестве одного из основных понятий в концепции свободного программного обеспечения (free software). Данный термин образуется за счет замены в английском языке термина «copyright» (авторское право) на «copyleft». Как указывают авторы проекта, «copyleft» — это наиболее общий способ сделать программное обеспечение свободным и обеспечить соблюдение условий, в соответствии с которыми все измененные и распространяемые версии программного обеспечение также сохраняли бы статус свободного программного обеспечения. Более подробно о концепции «copyleft» вы можете прочитать здесь: `http://www.gnu.org/copyleft/copyleft.html`. *(прим. перев.)*

«Второстепенный раздел» — имеющее название приложение или предисловие к Документу, в котором отражено исключительно отношение издателей или авторов Документа к его содержанию в целом, либо к вопросам, связанным с содержанием Документа. Второстепенный раздел не может включать в себя то, что относится непосредственно к содержанию Документа. (Например, если часть Документа является учебником по математике, во Второстепенном разделе не может содержаться что-либо имеющее отношение непосредственно к математике). Во Второстепенных разделах могут быть затронуты вопросы истории того, что составляет содержание или что связано с содержанием Документа, а также правовые, коммерческие, философские, этические или политические взгляды относительно содержания Документа.

«Неизменяемые разделы» — определенные Второстепенные разделы, названия которых перечислены как Неизменяемые разделы в уведомлении Документа, определяющем лицензионные условия.

«Текст, помещаемый на обложке» — определенные краткие строки текста, которые перечислены в уведомлении Документа, определяющем лицензионные условия, как текст, помещаемый на первой и последней страницах обложки.

«Прозрачный» экземпляр Документа — экземпляр Документа в машиночитаемой форме, представленный в формате с общедоступной спецификацией при условии, что документ может просматриваться и редактироваться непосредственно с помощью общедоступных текстовых редакторов или общедоступных программ для векторной или растровой графики (в случае, если в документе содержатся изображения векторной или растровой графики). Указанный формат должен обеспечить ввод текста Документа в программы форматирования текста или автоматический перевод Документа в различные форматы, подходящие для ввода текста Документа в программы форматирования текста. Экземпляр Документа, представленный в ином формате, разметка которого затрудняет или препятствует внесению в Документ последующих изменений пользователями, не является Прозрачным. Такой экземпляр документа называется *«Непрозрачным»*.

Форматы, в которых может быть представлен Прозрачный экземпляр Документа, включают простой формат ASCII без разметки, формат ввода Texinfo, формат ввода LaTeX, SGML или XML с использованием общедоступного DTD, а также соответствующий стандартам простой формат HTML, предназначений для внесения модификаций человеком. «Непрозрачные» форматы включают в себя PostScript, PDF, форматы, которые можно прочитать и редактировать только с

помощью текстовых редакторов, права на использование которых свободно не передаются, форматы SGML или XML, для которых DTD или инструменты для обработки не являются общедоступными, а также генерируемый машиной HTML, который вырабатывается некоторыми текстовыми редакторами исключительно в целях вывода.

«Титульный лист» — для печатной книги собственно титульный лист, а также следующие за ним страницы, которые должны содержать сведения, помещаемые на титульном листе в соответствии с условиями настоящей Лицензии. Для произведений, формат которых не предполагает наличие титульного листа, под Титульным листом понимается текст, который помещен перед началом основного текста произведения, после его названия, напечатанного наиболее заметным шрифтом.

2. Копирование без внесения изменений

Лицензиат вправе воспроизводить и распространять экземпляры Документа на любом носителе за вознаграждение или безвозмездно при условии, что каждый экземпляр содержит текст настоящей Лицензии, знаки охраны авторских прав, а также уведомление, что экземпляр распространяется в соответствии с настоящей Лицензией, при этом Лицензиат не вправе предусматривать иные лицензионные условия дополнительно к тем, которые закреплены в настоящей Лицензии. Лицензиат не вправе использовать технические средства для воспрепятствования или контроля за чтением или последующим изготовлением копий с экземпляров, распространяемых Лицензиатом. Лицензиат вправе получать вознаграждение за изготовление и распространение экземпляров Документа. При распространении большого количества экземпляров Документа Лицензиат обязан соблюдать условия пункта 3 настоящей Лицензии.

Лицензиат вправе сдавать экземпляры Документа в прокат на условиях, определенных в предыдущем абзаце, или осуществлять публичный показ экземпляров Документа.

3. Тиражирование

Если Лицензиат издает печатные экземпляры Документа в количестве свыше 100, и в соответствии с уведомлением Документа, определяющем лицензионные условия, Документ должен содержать Текст,

помещаемый на обложке, Лицензиат обязан издавать экземпляры Документа в обложке с напечатанными на ней ясно и разборчиво соответствующими Текстами, помещаемыми на обложке: Тексты, помещаемые на первой странице обложки — на первой странице, Тексты, помещаемые на последней странице — соответственно на последней. Также на первой и последней странице обложки экземпляра Документа должно быть ясно и разборчиво указано, что Лицензиат является издателем данных экземпляров. На первой странице обложки должно быть указано полное название Документа без пропусков и сокращений, все слова в названии должны быть набраны шрифтом одинакового размера. Лицензиат вправе поместить прочие сведения на обложке экземпляра. Если при издании экземпляров Документа изменяются только сведения, помещенные на обложке экземпляра, за исключением названия Документа, и при этом соблюдаются требования настоящего пункта, такие действия приравниваются к копированию без внесения изменений.

Если объем текста, который должен быть помещен на обложке экземпляра, не позволяет напечатать его разборчиво, Лицензиат обязан поместить разумную часть текста непосредственно на обложке, а остальной текст на страницах Документа, следующих сразу за обложкой.

Если Лицензиат издает или распространяет Непрозрачные экземпляры Документа в количестве свыше 100, Лицензиат обязан к каждому такому экземпляру приложить Прозрачный экземпляр этого Документа в машиночитаемой форме или указать на каждом Непрозрачном экземпляре Документа адрес в компьютерной сети общего пользования, где содержится Прозрачный экземпляр без каких-либо добавленных материалов, полный текст которого каждый пользователь компьютерной сети общего пользования вправе бесплатно, не называя своего имени и не регистрируясь, записать в память компьютера с использованием общедоступных сетевых протоколов. Во втором случае Лицензиат обязан предпринять разумные шаги с тем, чтобы доступ к Прозрачному экземпляру Документа по указанному адресу сохранялся по крайней мере в течение одного года после последнего распространения Непрозрачного экземпляра Документа данного тиража, независимо от того, было ли распространение осуществлено Лицензиатом непосредственно или через агентов или розничных продавцов.

Прежде чем начать распространение большого количества экземпляров Документа Лицензиату заблаговременно следует связаться с авторами Документа, чтобы они имели возможность предоставить Лицен-

зиату обновленную версию Документа. Лицензиат не обязан выполнять данное условие.

4. Внесение изменений

Лицензиат вправе воспроизводить и распространять Модифицированные версии Документа в соответствии с условиями пунктов 2 и 3 настоящей Лицензии, при условии что Модифицированная версия Документа публикуется в соответствии с настоящей Лицензией. В частности, Лицензиат обязан передать каждому обладателю экземпляра Модифицированной версии Документа права на распространение и внесение изменений в данную Модифицированную версию Документа, аналогично правам на распространение и внесение изменений, которые передаются обладателю экземпляра Документа. При распространении Модифицированных версий Документа Лицензиат обязан:

A. поместить на Титульном листе и на обложке при ее наличии название Модифицированной версии, отличающееся от названия Документа и названий предыдущих версий. Названия предыдущих версий при их наличии должны быть указаны в Документе в разделе «История». Лицензиат вправе использовать название предыдущей версии Документа с согласия издателя предыдущей версии;
B. указать на Титульном листе в качестве авторов тех лиц, которые являются авторами изменений в Модифицированной версии, а также не менее пяти основных авторов Документа либо всех авторов, если их не более пяти;
C. указать на Титульном листе наименование издателя Модифицированной версии, с указанием, что он является издателем данной Версии;
D. сохранить все знаки охраны авторского права Документа;
E. поместить соответствующий знак охраны авторского права на внесенные Лицензиатом изменения рядом с прочими знаками охраны авторского права;
F. поместить непосредственно после знаков охраны авторского права уведомление, в соответствии с которым каждому предоставляется право использовать Модифицированную Версию в соответствии с условиями настоящей Лицензии. Текст уведомления приводится в Приложении к настоящей Лицензии;
G. сохранить в уведомлении, указанном в подпункте F, полный список Неизменяемых разделов и Текста, помещаемого на обложке, перечисленных в уведомлении Документа;

H. включить в Модифицированную версию текст настоящий Лицензии без каких-либо изменений;

I. сохранить в Модифицированной версии раздел «История», включая его название, и дополнить его пунктом, в котором указать так же, как данные сведения указаны на Титульном листе, название, год публикации, наименования новых авторов и издателя Модифицированной версии. Если в Документе отсутствует раздел «История», Лицензиат обязан создать в Модифицированной версии такой раздел, указать в нем название, год публикации, авторов и издателя Документа так же, как данные сведения указаны на Титульном листе Документа и дополнить этот раздел пунктом, содержание которого описано в предыдущем предложении;

J. сохранить в Модифицированной версии адрес в компьютерной сети, указанный в Документе, по которому каждый вправе осуществить доступ к Прозрачному экземпляру Документа, а также адрес в компьютерной сети, указанный в Документе, по которому можно получить доступ к предыдущим версиям Документа. Адреса, по которым находятся предыдущие версии Документа, можно поместить в раздел «История». Лицензиат вправе не указывать адрес произведения в компьютерной сети, которое было опубликовано не менее чем за четыре года до публикации самого Документа. Лицензиат вправе не указывать адрес определенной версии в компьютерной сети с разрешения первоначального издателя данной версии;

K. сохранить без изменений названия разделов «Благодарности» или «Посвящения», а также содержание и стиль каждой благодарности и/или посвящения;

L. сохранить без изменений названия и содержание всех Неизменяемых разделов Документа. Нумерация данных разделов или иной способ их перечисления не включается в состав названий разделов;

M. удалить существующий раздел Документа под названием «Одобрения». Такой раздел не может быть включен в Модифицированную версию;

N. не присваивать существующим разделам Модифицированной версии название «Одобрения» или такие названия, которые повторяют название любого из Неизменяемых разделов.

Если в Модифицированную версию включены новые предисловия или приложения, которые могут быть определены как Второстепенные разделы и которые не содержат текст, скопированный из Документа,

Лицензиат вправе по своему выбору определить все или некоторые из этих разделов как Неизменяемые. Для этого следует добавить их названия в список Неизменяемых разделов в уведомлении в Модифицированной версии, определяющем лицензионные условия. Названия данных разделов должны отличаться от названий всех остальных разделов.

Лицензиат вправе дополнить Модифицированную версию новым разделом «Одобрения» при условии, что в него включены исключительно одобрения Модифицированной версии Документа третьими сторонами, например оценки экспертов или указания, что текст Модифицированной версии был одобрен организацией в качестве официального определения стандарта.

Лицензиат вправе дополнительно поместить на обложке Модифицированной версии Текст, помещаемый на обложке, не превышающий пяти слов для первой страницы обложке и 25 слов для последней страницы обложки. К Тексту, помещаемому на обложке, каждым лицом непосредственно или от имени этого лица на основании соглашения с ним может быть добавлено только по одной строке на первой и на последней страницах обложки. Если на обложке Документа Лицензиатом от своего имени или от имени лица, в интересах которого действует Лицензиат, уже был помещен Текст, помещаемый на обложке, Лицензиат не вправе добавить другой Текст. В этом случае Лицензиат вправе заменить старый текст на новый с разрешения предыдущего издателя, который включил старый текст в издание.

По настоящей Лицензии автор(ы) и издатель(и) Документа не передают право использовать их имена и/или наименования в целях рекламы или заявления или предположения, что любая из Модифицированных Версий получила их одобрение.

5. Объединение документов

Лицензиат с соблюдением условий п. 4 настоящей Лицензии вправе объединить Документ с другими документами, которые опубликованы на условиях настоящей Лицензии, при этом Лицензиат должен включить в произведение, возникшее в результате объединения, все Неизменяемые разделы из всех первоначальных документов без внесения в них изменений, а также указать их в качестве Неизменяемых разделов данного произведения в списке Неизменяемых разделов, который содержится в уведомлении, определяющем лицензионные условия для произведения.

Произведение, возникшее в результате объединения, должно содержать только один экземпляр настоящей Лицензии. Повторяющиеся в

произведении одинаковые Неизменяемые разделы могут быть заменены единственной копией таких разделов. Если произведение содержит несколько Неизменяемых Разделов с одним и тем же названием, но с разным содержанием, Лицензиат обязан сделать название каждого такого раздела уникальным путем добавления после названия в скобках уникального номера данного раздела или имени первоначального автора или издателя данного раздела, если автор или издатель известны Лицензиату. Лицензиат обязан соответственно изменить названия Неизменяемых разделов в списке Неизменяемых разделов в уведомлении, определяющем лицензионные условия для произведения, возникшего в результате объединения.

В произведении, возникшем в результате объединения, Лицензиат обязан объединить все разделы «История» из различных первоначальных Документов в один общий раздел «История». Подобным образом Лицензиат обязан объединить все разделы с названием «Благодарности» и «Посвящения». Лицензиат обязан исключить из произведения все разделы под названием «Одобрения».

6. Сборники документов

Лицензиат вправе издать сборник, состоящий из Документа и других документов, публикуемых в соответствии с условиями настоящей Лицензии. В этом случае Лицензиат вправе заменить все экземпляры настоящей Лицензии в документах одним экземпляром, включенным в сборник, при условии, что остальной текст каждого документа включен в сборник с соблюдением условий по осуществлению копирования без внесения изменений.

Лицензиат вправе выделить какой-либо документ из сборника и издать его отдельно в соответствии с настоящей Лицензией, при условии, что Лицензиатом в данный документ включен текст настоящей Лицензии и им соблюдены условия Лицензии по осуществлению копирования без внесения изменений в отношении данного документа.

7. Подборка документа и самостоятельных произведений

Размещение Документа или произведений, производных от Документа, с другими самостоятельными документами или произведениями на одном устройстве для хранения информации или носителе не

влечет за собой возникновения Модифицированной версии Документа, при условии, что Лицензиат не заявляет авторских прав на осуществленный им подбор или расположение документов при их размещении. Такое размещение называется *«Подборкой»*, при этом условия настоящей Лицензии не применяются к самостоятельным произведениям, размещенным вышеуказанным способом вместе с Документом, при условии, что они не являются произведениями, производными от Документа.

Если условия пункта 3 настоящей Лицензии относительно Текста, помещаемого на обложке, могут быть применены к экземплярам Документа в Подборке, то в этом случае Текст с обложки Документа может быть помещен на обложке только собственно Документа внутри подборки при условии, что Документ занимает менее четвертой части объема всей Подборки. Если Документ занимает более четвертой части объема Подборки, в этом случае Текст с обложки Документа должен быть помещен на обложке всей Подборки.

8. Перевод

Перевод является одним из способов модификации Документа, в силу чего Лицензиат вправе распространять экземпляры перевода Документа в соответствии с пунктом 4 настоящей Лицензии. Замена Неизменяемых разделов их переводами может быть осуществлена только с разрешения соответствующих правообладателей, однако Лицензиат вправе в дополнение к оригинальным версиям таких Неизменяемых разделов включить в текст экземпляра перевод всех или части таких Разделов. Лицензиат вправе включить в текст экземпляра перевод настоящей Лицензии при условии, что в него включен также и оригинальный текст настоящей Лицензии на английском языке. В случае разногласий в толковании текста перевода и текста на английском языке предпочтение отдается тексту Лицензии на английском языке.

9. Расторжение лицензии

Лицензиат вправе воспроизводить, модифицировать, распространять или передавать права на использование Документа только на условиях настоящей Лицензии. Любое воспроизведение, модификация, распространение или передача прав на иных условиях являются недействительными и автоматически ведут к расторжению настоящей Лицензии и прекращению всех прав Лицензиата, предоставленных ему

настоящей Лицензией. При этом права третьих лиц, которым Лицензиат в соответствии с настоящей Лицензией передал экземпляры Документа или права на него, сохраняются в силе при условии полного соблюдения ими настоящей Лицензии.

10. Пересмотр условий лицензии

Free Software Foundation может публиковать новые исправленные версии GFDL. Такие версии могут быть дополнены различными нормами, регулирующими правоотношения, которые возникли после опубликования предыдущих версий, однако в них будут сохранены основные принципы, закрепленные в настоящей версии (смотри `http://www/gnu.org/copyleft/`).

Каждой версии присваивается свой собственный номер. Если указано, что Документ распространяется в соответствии с определенной версией, т.е. указан ее номер, или любой более поздней версией настоящей Лицензии, Лицензиат вправе присоединиться к любой из этих версий Лицензии, опубликованных Free Software Foundation (при условии, что ни одна из версий не является проектом Лицензии). Если Документ не содержит такого указания на номер версии Лицензии Лицензиат вправе присоединиться к любой из версий Лицензии, опубликованных когда-либо Free Software Foundation (при условии, что ни одна из версий не является Проектом Лицензии).

Порядок применения условий настоящей Лицензии к вашей документации

Чтобы применить условия настоящей Лицензии к созданному вами документу, вам следует включить в документ текст настоящей Лицензии, а также знак охраны авторского права и уведомление, определяющее лицензионные условия, сразу после титульного листа документа в соответствии с нижеприведенным образцом:

© имя (наименование) автора или иного правообладателя, год первого опубликования документа
Каждый имеет право воспроизводить, распространять и/или вносить
изменения в настоящий Документ в соответствии с условиями GNU Free
Documentation License, Версией 1.1 или любой более поздней версией,

опубликованной Free Software Foundation;
Данный Документ содержит следующие Неизменяемые разделы, (указать
названия Неизменяемых разделов); данный документ содержит следующий
Текст, помещаемый на первой странице обложки (перечислить),
данный документ содержит следующий Текст, помещаемый на последней
странице обложки (перечислить).
Копия настоящей Лицензии включена в раздел под названием "GNU Free
Documentation License".

Если документ не содержит Неизменяемых разделов, укажите «Данный документ не содержит Неизменяемых разделов». Если документ не содержит Текста, помещаемого на первой или последней страницах обложки, укажите «Данный документ не содержит Текста, помещаемого на первой странице обложки», соответственно укажите для последней страницы обложки.

Если ваш документ содержит имеющие существенное значение примеры программного кода, мы рекомендуем вам выпустить их отдельно в соответствии с условиями одной из лицензий на свободное программное обеспечение, например GNU General Public License, чтобы их можно было использовать как свободное программное обеспечение.

Содержание

Список иллюстраций

Список таблиц

Список примеров