# Управление версиями с помощью CVS

для CVS 1.10.8

Per Cederqvist et al
Перевод на русский язык — Алексей Махоткин

# 1 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.

## 1.1 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`, о котором подробно рассказывается в Приложение H [Ошибки в CVS], с. 161. Чтобы подписаться, напишите на `bug-cvs-request@gnu.org`.

## 1.2 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`.

Дополнительная информация о сборке проектов при использовании CVS находится в Глава 14 [Сборка проекта], с. 85.

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`' (см. Приложение C [Административные файлы], с. 133).

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 `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.

## 1.3 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 2 [Репозиторий], с. 7); 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`'.

### 1.3.1 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 RCCS: не требуется блокировать файлы, которые вы желаете отредактировать. См. Глава 10 [Несколько разработчиков], с. 63, где приведены дополнительные объяснения.

### 1.3.2 Фиксирование изменений

После того, как вы проверили, что компилятор все еще компилируется, вы решили создать новую версию '`backend.c`'. При этом в репозитории появится ваш новый '`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
```

### 1.3.3 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` (см. Раздел A.15 [Команда release], с. 115):

```
$ 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. См. Раздел C.10 [Файл history], с. 144.

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. См. Раздел C.9 [Файл cvsignore], с. 143, где можно найти информацию о том, как избежать этого предупреждения. См. Раздел A.15.2 [Сообщения команды release], с. 116, где находится полная информация о возможных сообщениях команды `release`.

'`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.

### 1.3.4 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
```

# 2 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 '`/usr/local/cvsroot`' на компьютере, на котором используется CVS. Другие методы доступа описаны в Раздел 2.9 [Сетевые репозитории], с. 19.

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.

## 2.1 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.

## 2.2 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 (см. Раздел 10.5 [Совместный доступ], с. 68) или если вам потребуется изменить права доступа к файлам в репозитории.

### 2.2.1 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 5.5 [Волшебные номера веток], с. 44. Имена меток, которые позволяет использовать CVS, являются подмножеством тех, что позволены в RCS; см. Раздел 4.4 [Метки], с. 34.

### 2.2.2 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 (см. Раздел 10.5 [Совместный доступ], с. 68).

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 (См. Глава 13 [Слежение за исходными текстами], с. 81. См. Раздел A.4 [Глобальные ключи], с. 90, где описан ключ ‘`-r`’; См. Приложение D [Переменные окружения], с. 147, в которой описана переменная `CVSREAD`).

Заметьте, что при использовании клиент-серверного CVS (см. Раздел 2.9 [Сетевые репозитории], с. 19) 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 2.9.3.1 [Сервер парольной аутентификации], с. 21.

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 Раздел 2.9.3.1 [Сервер парольной аутентификации], с. 21.

### 2.2.3 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).

### 2.2.4 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.

### 2.2.5 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 CVS 1.7 и выше, см. Раздел 10.6.5 [Совместимость слежений], с. 72.

### 2.2.6 Блокировки в репозитории

Видимое пользователем поведение блокировок CVS описано в Раздел 10.5 [Совместный доступ], с. 68. 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 CVS 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 `rcs_internal_lockfile` в исходном коде CVS, где находится дополнительное обсуждение и мотивация.

### 2.2.7 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 (см. Приложение H [Ошибки в CVS], с. 161). 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`'.

Информация о назначении разнообразных административных файлов находится в Приложение C [Административные файлы], с. 133.

## 2.3 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 Раздел 2.1 [Указание репозитория], с. 7.

'`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 If the first character is '`/`', then the format is:

*/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 Раздел 4.9 [Липкие метки], с. 38.

‘`Checkin.prog`’
‘`Update.prog`’
: В этих файлах хранятся имена программ, заданных опциями ‘`-i`’ и ‘`-u`’ в файле ‘`modules`’, соответственно.

‘`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:

  B*имя*/*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`’ (см. Раздел C.8 [Файл rcsinfo], с. 142). Он используется только клиентом; не-клиент-серверные варианты CVS напрямую обращаются к ‘`rcsinfo`’.

## 2.4 Административные файлы

Каталог ‘`$CVSROOT/CVSROOT`’ содержит несколько *административных файлов*. Полное их описание в См. Приложение C [Административные файлы], с. 133. 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. Полное описание всех доступных возможностей находится в См. Раздел C.1 [Файл modules], с. 133.

### 2.4.1 Редактирование административных файлов

Административные файлы можно редактировать точно так же, как и любой другой модуль. Используйте ‘`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.

## 2.5 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 2.1 [Указание репозитория], с. 7.

Серьезным преимуществом нескольких репозиториев является то, что они могут находиться на различных серверах. При использовании CVS 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.

## 2.6 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 Раздел 2.9.1 [Требования к серверу], с. 19.

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 (см. Раздел 10.5 [Совместный доступ], с. 68).

Чтобы создать репозиторий, выполните команду `cvs init`. Она создаст пустой репозиторий в корневом каталоге CVS, заданном обычным образом (см. Глава 2 [Репозиторий], с. 7). 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`. См. Раздел C.10 [Файл history], с. 144.

## 2.7 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. См. Раздел 10.5 [Совместный доступ], с. 68, где приводится дополнительная информация о блокировках CVS. 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.

## 2.8 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 2.3 [Хранение файлов в рабочем каталоге], с. 13, где приводится дополнительная информация о файлах '`CVS/Repository`' и '`CVS/Root`', но если вы не уверены, то, наверное, лучше не пытаться.

## 2.9 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 Раздел 2.9.2 [Соединение с помощью rsh], с. 20.

### 2.9.1 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 (если вы обнаружите, что это не так, дайте нам знать, как описано в Приложение H [Ошибки в CVS], с. 161, 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 Раздел 2.6 [Создание репозитория], с. 17.

### 2.9.2 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 так как 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.)

### 2.9.3 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.

#### 2.9.3.1 Настройка сервера для парольной аутентификации

Во-первых, вы, вероятно, хотите усилить права доступа к каталогам '`$CVSROOT`' и '`$CVSROOT/CVSROOT`'. См. Раздел 2.9.3 [Парольная аутентификация], с. 21, где описаны детали.

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 — 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 (отформатировано, чтобы влезло 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. В случае проблем с настройкой смотрите Раздел F.2 [Соединение], с. 157.

Так как клиент хранит и пересылает пароли практически открытым тестом (См. Раздел 2.9.3 [Парольная аутентификация], с. 21, где описаны детали), то может использоваться отдельный файл паролей для CVS, чтобы пользователи не раскрывали своих обычных паролей при доступе к репозиторию. Этот файл — '`$CVSROOT/CVSROOT/passwd`' (см. Раздел 2.4 [Введение в административные файлы], с. 16). 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 См. Раздел 2.10 [Доступ только для чтения], с. 27.

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`', см. Раздел C.12 [Файл config], с. 145). 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 2.9.3.3 [Безопасность парольной аутентификации], с. 24, 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. См. Раздел 2.2.7 [Каталог CVSROOT], с. 12.

### 2.9.3.2 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 2.9.2 [Соединение с помощью rsh], с. 20).

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`.

#### 2.9.3.3 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 2.9.3.1 [Сервер парольной аутентификации], с. 21) 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.

### 2.9.4 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 2.9.3.1 [Сервер парольной аутентификации], с. 21. 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
```

### 2.9.5 Прямое соединение с помощью Kerberos

Самый простой способ использования Kerberos — это kerberos `rsh`, что описано в Раздел 2.9.2 [Соединение с помощью rsh], с. 20. 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

```
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.

### 2.9.6 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.

## 2.10 Доступ к репозиторию только для чтения

Существует возможность предоставить публичный доступ к репозиторию только для чтения, используя сервер парольной аутентификации (см. Раздел 2.9.3 [Парольная аутентификация], с. 21). (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 2.9.3.1 [Сервер парольной аутентификации], с. 21).

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
jrandom
```

(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 2.9.3.1 [Сервер парольной аутентификации], с. 21), 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.

## 2.11 Временные каталоги на сервере

В процессе работы CVS-сервер создает временные каталоги. Они называются

`cvs-serv`*pid*

где *pid* — это номер процесса сервера. Они находятся в каталоге, указанном в переменной окружения `TMPDIR` (см. Приложение D [Переменные окружения], с. 147), 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-serv`*pid*’. Если нет сервера с номером процесса *pid*, то сделать это можно совершенно безопасно.

# 3 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 7.4 [Перемещение файлов], с. 54.

Дальнейшие действия зависят от конкретной ситуации.

## 3.1 Помещение файлов в репозиторий

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

### 3.1.1 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. См. Глава 13 [Слежение за исходными текстами], с. 81, за дальнейшей информацией.

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 2.2.2 [Права доступа к файлам], с. 10.

Если какие-то из файлов, которые нужно импортировать, являются бинарными, вам потребуется использовать *обертки*, чтобы указать, какие именно. См. Раздел C.2 [Обертки], с. 136.

### 3.1.2 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, найдите все RCS-файлы, 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 добавленным '`,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.

  RCS-файл 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 RCS-файлы в стандартном формате. Если ваша система умеет так делать, экспортируйте RCS-файлы 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`', конвертирующий 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.

### 3.1.3 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.

## 3.2 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. См. Раздел 2.4 [Введение в административные файлы], с. 16. Полное описание файла '`modules`' можно найти в См. Раздел C.1 [Файл modules], с. 133. 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
   ```

# 4 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.

## 4.1 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 на ветках (см. Глава 5 [Создание ветвей и слияние], с. 41); эти номера подробно описаны в Раздел 5.4 [Ветки и ревизии], с. 43.

## 4.2 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`'.

## 4.3 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 4.4 [Метки], с. 34). 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. Глава 5 [Создание ветвей и слияние], с. 41).

## 4.4 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’ (см. Раздел 8.3 [Настройка журналирования], с. 57).

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)
```

Полный синтаксис команды `cvs tag`, включая разнообразные ключи командной строки, описан в Приложение B [Вызов CVS], с. 121.

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. Глава 6 [Рекурсивное поведение], с. 49.)

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. См. Раздел A.7.1 [Ключи команды checkout], с. 100. Задав команде `checkout` ключ командной строки ‘`-r`’, следует остерегаться липких меток; см. Раздел 4.9 [Липкие метки], с. 38.

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
```

## 4.5 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!
```

## 4.6 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. См. Раздел A.5 [Стандартные ключи], с. 92, где приведено полное описание.

`-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
```

## 4.7 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`' (см. Раздел 8.3 [Настройка журналирования], с. 57).

Чтобы удалить метку, задайте ключ командной строки '`-d`' команде `cvs tag` или `cvs 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
```

## 4.8 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.

## 4.9 Липкие метки

Иногда ревизия, находящаяся в рабочем каталоге, содержит также дополнительную информацию о себе: например, она может находиться на ветке (см. Глава 5 [Создание ветвей и слияние], с. 41), 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 Раздел 5.3 [Доступ к веткам], с. 42. 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 Раздел 5.8 [Слияние двух ревизий], с. 46.

# 5 Создание ветвей и слияние

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.

## 5.1 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 4.4 [Метки], с. 34) 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.

## 5.2 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 5.3 [Доступ к веткам], с. 42, 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`’.

## 5.3 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 5.2 [Создание ветви], с. 41).

```
$ 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 4.9 [Липкие метки], с. 38) — 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`’.

Смотри Раздел 5.4 [Ветки и ревизии], с. 43, где подробно описано, как устроены номера ветвей.

## 5.4 Ветки и ревизии

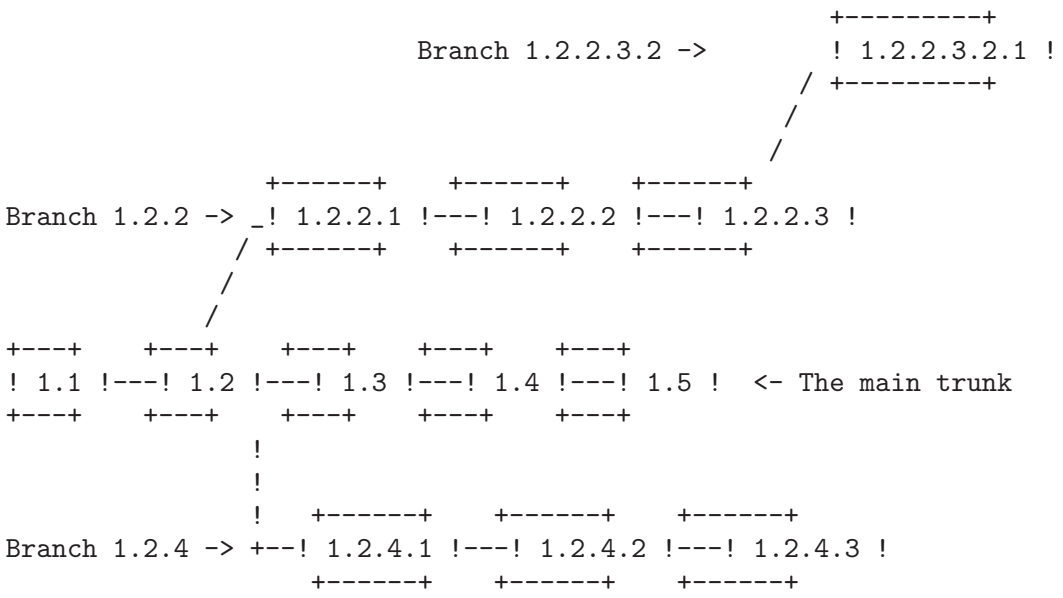
Обычно история ревизий файла — это линейная возрастающая последовательность номеров (см. Раздел 4.1 [Номера ревизий], с. 33):

```
+---+    +---+    +---+    +---+    +---+
! 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.




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 5.5 [Волшебные номера веток], с. 44). Ветка 1.1.1 имеет специальное значение. См. Глава 13 [Слежение за исходными текстами], с. 81.

## 5.5 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 4.1 [Номера ревизий], с. 33. 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`.
- 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).

## 5.6 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. См. Раздел 10.3 [Пример конфликта], с. 65.

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

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

## 5.7 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
```

## 5.8 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
$
```

## 5.9 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.

For example:

```
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.

# 6 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`’ (см. Раздел A.3 [~/.cvsrc], с. 90).

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

# 7 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.

## 7.1 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 1.3.1 [Получение исходного кода], с. 4.
- Создайте новый файл в рабочей копии каталога.
- Используйте '`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`' (см. Глава 9 [Двоичные файлы], с. 59).
- 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
```

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

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`. См. Раздел A.12 [Команда import], с. 110.

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 7.2 [Удаление файлов], с. 52, там находится пример.

Ключ командной строки '`-k`' задает способ по умолчанию, которым будут извлекаться файлы, дальнейшая информация находится в Глава 12 [Подстановка ключевых слов], с. 77.

Ключ командной строки '`-m`' задает описание файла. Описание появляется в журнале истории, если разрешено его использование, см. Раздел C.10 [Файл history], с. 144. 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`. См. Раздел A.6 [Команда admin], с. 95. 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. Глава 5 [Создание ветвей и слияние], с. 41). Вы можете позднее поместить добавления на другую ветку, если захотите (см. Раздел 5.9 [Слияние добавлений и удалений], с. 47).

## 7.2 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 1.3.4 [Просмотр изменений], с. 6, 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`.

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

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 Приложение B [Вызов CVS], с. 121.

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. Глава 5 [Создание ветвей и слияние], с. 41). Позже можно слить удаления на другую ветку, если захотите (см. Раздел 5.9 [Слияние добавлений и удалений], с. 47).

## 7.3 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.

## 7.4 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 7.5 [Перемещение каталогов], с. 55.)

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

### 7.4.1 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 смотри в Раздел 4.3 [Назначение номеров ревизий], с. 33.

### 7.4.2 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.

### 7.4.3 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 ‘`-r`*метка*’, а не ‘`-D`*дата*’.
- Журнал изменений остается в целости и сохранности.
- Номера ревизий не искажаются.

Недостатки:

- Нет способа легко увидеть историю файла до переименования.

## 7.5 Перемещение и переименование каталогов

Обычный способ переименовать или переместить каталог — переименовать или переместить каждый файл в нем, как описано в Раздел 7.4.1 [Снаружи], с. 54. Затем следует извлечь их заново, используя ключ командной строки ‘`-P`’, как описано в Раздел 7.3 [Удаление каталогов], с. 53.

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.

# 8 Просмотр истории

После того, как вы успели поиспользовать CVS для хранения информации об истории изменений: какие файлы, когда, как и кто изменил, вам потребуются разнообразные механизмы для просмотра истории.

## 8.1 Журнальные записи

Каждый раз, когда вы фиксируете изменения в файле, вам предлагается создать соответствующую журнальную запись.

Для того, чтобы просмотреть журнальные записи, соответствующие каждой зафиксированной ревизии, используйте команду `cvs log` (см. Раздел A.13 [Команда log], с. 112).

## 8.2 База истории

Вы можете использовать файл history (см. Раздел C.10 [Файл history], с. 144), чтобы журналировать разнообразные действия CVS. Чтобы извлечь информацию из файла history, используйте команду `cvs history` (см. Раздел C.10 [Файл history], с. 144).

## 8.3 Настройка журналирования

Вы можете настроить CVS для журналирования различных действий тем способом, который вам требуется. Это достигается выполнением определенного скрипта в определенные моменты времени. Скрипт может, например, добавить сообщение об изменении в конец какого-либо файла, послать почтовое сообщение группе разработчиков или, например, поместить сообщение в группу новостей. Для того, чтобы журналировать факты фиксирования, используйте файл '`loginfo`' (см. Раздел C.7 [Файл loginfo], с. 141).

Для журналирования фиксирований, извлечений, экспортов и меток можно использовать флаги '`-i`', '`-o`', '`-e`' и '`-t`' соответственно. Эти флаги находятся в файле модулей. Более гибким способом уведомления пользователей, требующим меньше усилий по поддержке централизованных скриптов, является команда `cvs watch add` (см. Раздел 10.6.2 [Получение уведомлений], с. 69); эта команда полезна, даже если вы не используете `cvs watch on`.

В файле '`taginfo`' перечисляются программы, которые нужно выполнить, когда кто-либо выполняет команды CVS `tag` или `rtag`. Файл '`taginfo`' имеет стандартный формат административных файлов (см. Приложение C [Административные файлы], с. 133), а каждая строка в нем содержит регулярное выражение, за которым следует команда, которую надо выполнить. Аргументы, которые передаются команде, это *имя-метки*, *операция* (`add` для `tag`, `mov` для `tag -F`, `del` для `tag -d`), *репозиторий*, а затем следует серия пар *имя-файла ревизия*. Ненулевой код завершения программы приведет к отмене операции с метками.

Вот пример использования '`taginfo`' для журналирования команд `tag` и `rtag`. В файле '`taginfo`' написано:

```
ALL /usr/local/cvsroot/CVSROOT/loggit
```

Здесь ‘`/usr/local/cvsroot/CVSROOT/loggit`’ является таким скриптом:

```
#!/bin/sh
echo "$@" >>/home/kingdon/cvsroot/CVSROOT/taglog
```

## 8.4 Команда annotate

**cvs annotate** [`-flR`] [`-r rev|-D date`] *files* . . . Команда

Для каждого файла в списке *files*, напечатать головную ревизию в стволе, а также информацию о последних изменениях в каждой строке. Например:

```
$ cvs annotate ssfile
Annotations for ssfile
***************
1.1          (mary     27-Mar-96): ssfile line 1
1.2          (joe      28-Mar-96): ssfile line 2
```

В файле ‘`ssfile`’ в настоящий момент содержит две строки. Строка `ssfile line 1` была зафиксирована `mary` двадцать седьмого марта. Затем, двадцать восьмого, `joe` добавил строки `ssfile line 2`, не изменяя строки `ssfile line 1`. В этом отчете ничего не сказано о том, что было удалено или заменено; используйте `cvs diff`, чтобы узнать это (см. Раздел A.9 [Команда diff], с. 105).

Ключи команды `cvs annotate` перечислены в Приложение B [Вызов CVS], с. 121, и могут использоваться для выбора файлов и их ревизий, которые нужно аннотировать. Ключи детально описаны в Раздел A.5 [Стандартные ключи], с. 92.

# 9 Обработка двоичных файлов

Основное применение для CVS – хранить текстовые файлы. При работе с текстовыми файлами CVS может объединять ревизии, показывать различия между ревизиями в доступном для человека формате, и совершать прочие подобные операции. Однако, если вы согласитесь отказаться от некоторых возможностей, то CVS может хранить двоичные файлы. Например, можно хранить в CVS целый web-сайт, причем как страницы, так и двоичные картинки.

## 9.1 Вопросы использования двоичных файлов

Если вы постоянно работаете с двоичными файлами, то необходимость их использования очевидна. Если же вы хотите хранить историю изменений таких файлов, возникают дополнительные вопросы.

Одной из основных функций контроля версий — просмотр различий между двумя ревизиями. Например, если кто-то еще создает новую ревизию файла, то вам хотелось бы взглянуть на то, что изменилось и выяснить, верно ли были сделаны изменения. Для текстовых файлов CVS обеспечивает такую функциональность с помощью команды `cvs diff`. Для двоичных файлов можно извлечь две ревизии и сравнить их с помощью внешнего инструмента (например, в текстовых процессорах обычно имеется такая возможность. Если такого инструмента не существует, нужно отслеживать изменения посредством с помощью других механизмов, например, заставляя людей писать хорошие журнальные сообщения, надеясь при этом, что они действительно сделали то, что намеревались сделать.

Другая возможность системы контроля версий — объединение двух ревизий. Для CVS это происходит в двух контекстах. Во-первых, пользователи редактируют файлы в различных рабочих каталогах (см. Глава 10 [Несколько разработчиков], с. 63). Во-вторых, объединения совершаются явно, используя команду ‘`update -j`’ (см. Глава 5 [Создание ветвей и слияние], с. 41).

В случае текстовых файлов CVS может объединить изменения, совершенные независимо друг от друга, и сигнализировать о конфликте, если нужно. В случае двоичных файлов лучшее, что может сделать CVS — выдать две различных копии файла и предоставить пользователю справиться с конфликтом. Пользователь может выбрать ту или иную копию, или использовать специальный инструмент для слияния файлов этого конкретного формата, если таковой инструмент существует. Заметьте, что необходимость слияния изменений вручную полностью зависит от аккуратности пользователя, поэтому может привести к ошибкам.

Если вышеописанный процесс нежелателен, лучшим выходом было бы отказаться от автоматического слияния. Чтобы избежать слияний, являющихся результатом работы в разных рабочих каталогах, посмотрите обсуждение блокирующих извлечений (блокировок файлов) в Глава 10 [Несколько разработчиков], с. 63. Чтобы избежать слияний, образующихся в результате использования ветвей, ограничьте использование ветвей.

## 9.2 Как хранить двоичные файлы

При хранении двоичных файлов встает два вопроса. Первый: CVS по умолчанию преобразует коды конца строк между канонической формой, в которой они хранятся в репозитории (только символ новой строки) и формой, соответствующей операционной системе клиента (например, возврат каретки, за которым следует перевод строки под Windows NT).

Второй вопрос — в двоичном файле могут оказаться данные, похожие на ключевое слово (см. Глава 12 [Подстановка ключевых слов], с. 77), так что эта подстановка должна быть выключена.

Ключ командной строки '`-kb`', доступный при использовании некоторых команд CVS, позволяет убедиться, что ни преобразование концов строки, ни подстановка ключевых слов не производятся.

Вот пример создания нового файла с использование флага '`-kb`':

```
$ echo '$Id$' > kotest
$ cvs add -kb -m"A test file" kotest
$ cvs ci -m"Первое фиксирование; файл содержит ключевое слово" kotest
```

Если файл случайно будет добавлен без '`-kb`', можно использовать команду `cvs admin` для восстановления. Например,

```
$ echo '$Id$' > kotest
$ cvs add -m"A test file" kotest
$ cvs commit -m"Первое фиксирование; содержит ключевое слово" kotest
$ cvs admin -kb kotest
$ cvs update -A kotest
# Для не-UNIX систем:
# Скопировать откуда-нибудь хорошую копию файла
$ cvs commit -m "Сделать файл двоичным" kotest
```

Когда вы извлечете файл '`kotest`', он не станет двоичным, потому что вы не зафиксировали его как двоичный. Команда `cvs admin -kb` устанавливает метод подстановки ключевых слов по умолчанию для этого файла, но не изменяет рабочую копию файла, которая у вас есть. Если вам нужно справиться с символами конца строки (при использовании CVS на не-UNIX системах), вам требуется зафиксировать новую копию файла, как показано выше в команде `cvs commit`. Под UNIX достаточно выполнить `cvs update -A`.

Однако, используя `cvs admin -k` для изменения режима подстановки ключевых слов, знайте, что этот режим не подчиняется контролю версий. Это означает, что если, скажем, в старых версиях пакета какой-то файл был текстовым, а затем в новых версиях появился двоичный файл с тем же именем, то CVS не обеспечивает способа извлечь файл в двоичном или текстовом режиме в зависимости от версии пакета, которую вы извлекаете. Для обхода этой проблемы хорошего решения не существует.

Вы можете установить значения по умолчанию, которые используют команды `cvs add` и `cvs import`, для выяснения, является ли файл текстовым или двоичным, основываясь на его имени. Например, можно гарантировать, что файлы, чьи имена заканчиваются на '`.exe`' являются двоичными. См. Раздел C.2 [Обертки], с. 136. В настоящий момент нет способа для определения, является ли файл текстовым или двоичным, в

зависимости от его содержимого. Основная трудность при разработке такой возможности — неясно, как различить такие файлы, потому что способы сильно различаются в разных операционных системах.

# 10 Несколько разработчиков

Когда над программным проектом работает более одного человека, ситуация резко усложняется. Зачастую два разработчика одновременно пытаются редактировать один и тот же файл. Одно из решений, известное как *блокировка файлов* или *блокированное извлечение*, — в каждый момент времени позволять редактировать файл только одному человеку. Это — единственное решение при использовании некоторых систем контроля версий, включая RCS и SCCS. В настоящее время обычным способом совершить блокированное извлечение рабочей копии с помощью CVS — использовать команду `cvs admin -l` (см. Раздел A.6.1 [Ключи команды admin], с. 95). Эта возможность не столь красиво интегрирована в CVS, как функции слежения, описанные ниже, но кажется, что всех, кому требуется блокированное извлечение, эта команда устраивает.

Во избежание одновременного редактирования файла двумя разработчиками можно также использовать возможность слежения, описанную ниже, вместе с соответствующими административными процедурами (не контролируемыми с помощью программы).

Модель, используемая в CVS по умолчанию, называется *неблокированные извлечения*. В этой модели разработчики редактируют свои собственные *рабочие копии* файла. Первый, зафиксировавший свои изменения, не может автоматически узнать, что второй также начал редактировать файл. Второй получит сообщения об ошибке, когда попытается зафиксировать файл. Он должен использовать соответствующие команды CVS, чтобы его рабочая копия соответствовала свежайшей ревизии, находящейся в репозитории. Весь этот процесс проходит почти автоматически.

CVS также поддерживает механизмы различных способов коммуникации, никак не настаивая на выполнении каких-либо правил, в отличие от блокированных извлечений.

Оставшаяся часть главы описывает, как работают все эти различные модели, а также некоторые вопросы, связанные с выбором того или иного варианта.

## 10.1 Статус файла

Основываясь на операциях, которые производятся над извлеченным файлом, а также на операциях, которые производятся над этим файлом в репозитории, можно классифицировать несколько состояний файла. Команда `status` рапортует об этих состояниях. Они таковы:

Up-to-date
: Файл идентичен последней ревизии в репозитории, находящейся на используемой ветке.

Locally Modified
: Вы редактировали этот файл и еще не зафиксировали изменения.

Locally Added
: Вы добавили этот файл с помощью `cvs add`, и еще не зафиксировали изменения.

Locally Removed
: Вы удалили файл с помощью `cvs remove` и еще не зафиксировали изменения.

Needs Checkout
: Кто-то еще поместил новую ревизию в репозиторий. Название немного сбивает с толку, потому что требуется использовать команду `cvs update`, а не `cvs checkout`, чтобы получить свежайшую версию.

Needs Patch
: Похоже на "Needs Checkout", но CVS-сервер пошлет заплату, а не целый файл. В принципе это приведет к тому же самому результату.

Needs Merge
: Кто-то еще поместил новую ревизию в репозиторий, а вы также изменили этот файл.

File had conflicts on merge
: Похоже на "Locally Modified", только последняя выполненная команда `cvs update` обнаружила конфликт. Если вы еще не исправили его, сделайте это, как описано в Раздел 10.3 [Пример конфликта], с. 65.

Unknown
: CVS ничего не знает об этом файле. Например, вы создали новый файл и еще не выполнили `cvs add`.

Чтобы уточнить состояние файла, `cvs status` также сообщает о `Working revision`, являющейся ревизией, на основе которой создан файл в рабочем каталоге, и `Repository revision`, являющейся свежайшей ревизией в репозитории, находящейся на используемой ветке.

Ключи команды `status` перечислены в Приложение B [Вызов CVS], с. 121. Информация о `Sticky tag` и `Sticky date` находится в Раздел 4.9 [Липкие метки], с. 38. Информация о `Sticky options` находится в описании флага ‘`-k`’ в Раздел A.16.1 [Ключи команды update], с. 117.

Команды `status` и `update` можно рассматривать как соответствующие друг другу. `update` используется для извлечения самых свежих файлов, а `status` — для выяснения, что же произойдет, если выполнить `update` (конечно, состояние репозитория может измениться до того, как вы выполните `update`). В действительность, если вы хотите узнать состояние файлов в более краткой форме, выполните

```
$ cvs -n -q update
```

Ключ командной строки ‘`-n`’ указывает не выполнять обновление, а просто сообщить о состоянии файлов; ‘`-q`’ не печатает имена каждого каталога. Прочую информацию о команде `update` можно найти в Приложение B [Вызов CVS], с. 121.

## 10.2 Извлечение свежей ревизии файла

Если вы хотите получить новую ревизию файла или объединить его с другой ревизией, используйте команду `update`. Если вы имеете старую ревизию файла, то эта команда эквивалентна `checkout`: свежая ревизия извлекается из репозитория и помещается в рабочий каталог.

Ваши изменения в файле не теряются, когда вы используете `update`. Если более свежей ревизии не существует, выполнение `update` ничего не сделает. Если вы редактировали файл, а в репозитории появилась его более новая ревизия, изменения будут объединены с вашей рабочей копией.

Например, представим себе, что вы извлекли ревизию 1.4 и начали редактировать ее. В это время кто-то еще поместил в репозиторий ревизию 1.5 и, вскорости, ревизию 1.6. Если теперь вы выполните команду `update`, CVS внедрит изменения между ревизиями 1.4 и 1.6 в ваш файл.

Если изменения между ревизиями 1.4 и 1.6 случились слишком близко к вашим изменениям, происходит *пересечение*. В таких случаях на экран выдается предупреждение, а в результирующем файле оказываются обе версии пересекающихся строк, разделенные специальными маркерами. См. Раздел A.16 [Команда update], с. 117, где полностью описана команда `update`.

## 10.3 Пример конфликта

Предположим, что ревизия 1.4 файла ‘`driver.c`’ содержит такой код:

```
#include <stdio.h>

void main()
{
    parse();
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(nerr == 0 ? 0 : 1);
}
```

Ревизия 1.6 файла ‘`driver.c`’ содержит такой код:

```
#include <stdio.h>

int main(int argc,
         char **argv)
{
    parse();
    if (argc != 1)
    {
        fprintf(stderr, "tc: No args expected.\n");
        exit(1);
    }
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(!!nerr);
}
```

Ваша рабочая копия файла ‘driver.c’, основанная на ревизии 1.4, перед выполнением ‘cvs update’ содержит такой код:

```
#include <stdlib.h>
#include <stdio.h>

void main()
{
    init_scanner();
    parse();
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
}
```

Вы выполняете ‘cvs update’:

```
$ cvs update driver.c
RCS file: /usr/local/cvsroot/yoyodyne/tc/driver.c,v
retrieving revision 1.4
retrieving revision 1.6
Merging differences between 1.4 and 1.6 into driver.c
rcsmerge warning: overlaps during merge
cvs update: conflicts found in driver.c
C driver.c
```

CVS сообщает, что вы встретились с конфликтами. Ваш исходный рабочий файл сохранен в ‘.#driver.c.1.4’. Новая версия ‘driver.c’ содержит такой код:

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc,
         char **argv)
{
    init_scanner();
    parse();
    if (argc != 1)
    {
        fprintf(stderr, "tc: No args expected.\n");
        exit(1);
    }
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
<<<<<<< driver.c
    exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
=======
    exit(!!nerr);
>>>>>>> 1.6
```

```
}
```

Заметьте, что непересекающиеся модификации включены в вашу рабочую копию, а пересекающаяся секция четко обозначена строками '<<<<<<<', '=======' and '>>>>>>>'.

Разрешить конфликт можно, отредактировав файл, удалив маркеры и неверный вариант. Предположим, в результате у вас получился такой файл:

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc,
         char **argv)
{
    init_scanner();
    parse();
    if (argc != 1)
    {
        fprintf(stderr, "tc: No args expected.\n");
        exit(1);
    }
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
}
```

Теперь вы можете поместить этот файл в репозиторий в качестве ревизии 1.7.

```
$ cvs commit -m "Initialize scanner. Use symbolic exit values." driver.c
Checking in driver.c;
/usr/local/cvsroot/yoyodyne/tc/driver.c,v  <-  driver.c
new revision: 1.7; previous revision: 1.6
done
```

Чтобы защитить вас, CVS откажется фиксировать файл, если в нем произошел конфликт и вы с ним не справились. В настоящий момент для разрешения конфликта нужно изменить дату модификации файла. В предыдущих версиях CVS вам также требовалось убедиться, что файл не содержит маркеров конфликта. Так как ваш файл действительно может содержать маркеры конфликтов (символы '>>>>>>>>' в начале строки, не обозначающие конфликта), то в текущей версии CVS выдает предупреждение и фиксирует файл.

Если вы используете `pcl-cvs` (оболочка к CVS для Emacs) версии 1.04 или позже, вы можете использовать пакет emerge, помогающий разрешать конфликты. Смотрите документацию по `pcl-cvs`.

## 10.4 Информирование коллег о фиксировании ревизий

Часто бывает полезно информировать своих коллег, когда вы фиксируете новую ревизию файла. Можно использовать ключ '`-i`' в файле '`modules`' или файле '`loginfo`' для автоматизации этого процесса. См. Раздел C.1 [Файл modules], с. 133. См.

Раздел C.7 [Файл loginfo], с. 141. Можно использовать эти возможности CVS для того, чтобы указать CVS, например, отсылать почтовые сообщения всем разработчикам или помещать сообщение в локальную группу новостей.

## 10.5 Совместный доступ нескольких разработчиков к CVS

Если несколько разработчиков попытаются одновременно выполнить CVS, один из них получит такое сообщение:

```
[11:43:23] waiting for bach's lock in /usr/local/cvsroot/foo
```

CVS попытается повторить операцию каждые 30 секунд, и либо успешно выполнит ее, либо опять напечатает сообщение, если опять нужно ждать. Если блокировка сохраняется слишком долго, найдите того, кто создал ее и спросите его, что за команду он выполняет. Если он не выполняет команд CVS, загляните в каталог репозитория, упомянутый в сообщении, и удалите файлы, чьи имена начинаются с '`#cvs.rfl`', '`#cvs.wfl`', или '`#cvs.lock`', принадлежащие указанному пользователю.

Заметьте, что эти блокировки предназначаются для защиты внутренних структур данных CVS и не имеют никакого отношения к слову *блокировка* в том смысле, который используется в RCS и означает блокированные извлечения (см. Глава 10 [Несколько разработчиков], с. 63).

Неограниченное количество пользователей может одновременно читать репозиторий; только когда кто-либо пишет туда, блокировки препятствуют прочим как читать, так и писать.

Можно было бы надеяться, что верно следующее утверждение:

**Если кто-либо фиксирует несколько изменений одной командой CVS, то команда update, выполненная кем-то еще, получит либо все изменения, либо ни одно из них.**

К сожалению, при работе с CVS это утверждение *неверно*. Например, если имеются файлы

```
a/one.c
a/two.c
b/three.c
b/four.c
```

и кто-то выполняет

```
cvs ci a/two.c b/three.c
```

а затем кто-то еще в то же самое время выполняет `cvs update`, то он может получить изменения для '`b/three.c`', и не получить изменения в '`a/two.c`'.

## 10.6 Как отследить, кто редактирует файлы?

Для большинства групп использования CVS в режиме по умолчанию совершенно достаточно. Пользователи иногда могут, попытавшись зафиксировать изменение, обнаружить, что мешает другое изменение, но они справятся с этим и все же зафиксируют свое изменение. В других группах предпочитают знать, кто какие файлы редактирует, так что если два человека захотят редактировать один и тот же файл, то они предпочтут договориться друг с другом, кто что будет делать, чтобы не получать проблем

при каждом фиксировании. Возможности, описанные в этой главе, помогают такой координации, сохраняя возможность редактирования одного файла двум разработчикам одновременно.

Для получения максимального преимущества разработчики должны использовать `cvs edit` (а не `chmod`), чтобы сделать файлы доступными для чтения/записи, и `cvs release` (а не `rm`) для удаления рабочего каталога, который более не используется. CVS не может вынудить их делать так.

### 10.6.1 Как с помощью CVS следить за определенными файлами?

Для того, чтобы включить использование функции слежения, сначала укажите, за какими файлами нужно следить.

**cvs watch on** [`-lR`] *файлы . . .* Команда

Команда означает, что разработчикам нужно выполнить команду `cvs edit` перед редактированием *файлов*. Для того, чтобы напомнить об этом разработчикам, CVS создаст рабочие копии *файлов* в режиме только для чтения.

Если среди *файлов* есть имя каталога, CVS включает слежение за всем файлами, находящимися в соответствующем каталоге репозитория и автоматически включает режим слежения за всеми файлами, которые будут в дальнейшем добавлены в каталог; это позволяет пользователю задать стратегию уведомлений для каждого каталога. Содержимое каталога обрабатывается рекурсивно, если только не задан ключ командной строки `-l`. Ключ `-R` позволяет включить рекурсивное поведение, если в файле '`~/.cvsrc`' оно было выключено с помощью ключа `-l` (см. Раздел A.3 [~/.cvsrc], с. 90).

Если список файлов пропущен, по умолчанию обрабатывается текущий каталог.

**cvs watch off** [`-lR`] *файлы . . .* Команда

Команда означает, что при извлечении не нужно создавать *файлы* в режиме только для чтения; таким образом, разработчики не будут получать напоминания о необходимости использования `cvs edit` и `cvs unedit`. CVS будет извлекать *файлы* в обычном режиме, если только на файле не установлены специальные права доступа, разрешенные с помощью ключевого слова `PreservePermissions` в административном файле '`config`'; см. Глава 15 [Специальные файлы], с. 87, а также См. Раздел C.12 [Файл config], с. 145.

*Файлы* и ключи командной строки обрабатываются точно так же, как и для `cvs watch on`.

### 10.6.2 CVS может посылать вам уведомления

Вы можете сказать CVS, что хотели бы получать уведомления о разнообразных действиях, совершенных с файлом. В принципе вы можете сделать это без использования `cvs watch on`, но обычно все же будете использовать как раз `cvs watch on`, чтобы другие разработчики использовали команду `cvs edit`.

**cvs watch add** [`-a` *действие*] [`-lR`] *файлы* . . . Команда

Добавить текущего пользователя в список лиц, которые будут получать уведомления о действиях, совершавшихся с *файлами*.

Ключ командной строки `-a` задает тип событий, о которых следует посылать уведомления. *действие* — это

`edit` Другой пользователь выполнил для файла команду `cvs edit` (описанную ниже).

`unedit` Другой пользователь выполнил команду `cvs unedit` (описанную ниже) или команду `cvs release`, или удалил файл и позволил команде `cvs update` создать его заново.

`commit` Другой пользователь зафиксировал изменения в файле.

`all` Все эти действия.

`none` Ни одно из этих действий. (Это полезно вместе с `cvs edit`, описанной ниже.)

Ключ `-a` можно указать несколько раз или вообще не указывать, в этом случае по умолчанию используется `all`.

*Файлы* и ключи командной строки обрабатываются так же, как и в команде `cvs watch`.

**cvs watch remove** [`-a` *действие*] [`-lR`] *файлы* . . . Команда

Удалить запрос на уведомление, созданный с помощью `cvs watch add`; аргументы те же самые. Если присутствует ключ командной строки `-a`, то только удаляются только слежения за указанными действиями.

Когда требуется отправить уведомление, CVS обращается к административному файлу ‘`notify`’. Этот файл можно отредактировать точно так же, как и другие административные файл (см. Раздел 2.4 [Введение в административные файлы], с. 16). Синтаксис этого файла подобен другим административным файлам (см. Раздел C.3.1 [Синтаксис], с. 137), где каждая строка состоит из регулярного выражения и команды, которую надо выполнить. Команда должна содержать одно единственное упоминание символов ‘`%s`’, которые будут заменены на имя пользователя, которого нужно уведомить; остальная информация передается этой команде на стандартный вход. Обычно в файл ‘`notify`’ помещается такая строка:

```
ALL mail %s -s \"CVS notification\"
```

В результате всего этого пользователи получают уведомления по электронной почте.

Заметьте, что если вы настроите все именно так, как рассказано выше, то пользователи будут получать уведомления на сервере. Конечно же, можно написать скрипт ‘`notify`’, который перенаправляет уведомления на другой адрес, но, для простоты, CVS позволяет задать адрес, по которому следует отсылать уведомления пользователю. Для этого создайте в ‘`CVSROOT`’ файл ‘`users`’, в котором каждая строка имеет вид

*пользователь:адрес*. Тогда вместо того, чтобы использовать имя пользователя, CVS будет использовать *адрес*.

CVS не уведомляет вас о ваших собственных изменениях. В настоящий момент проверка производится, основываясь на имени пользователя, который совершает действия, приводящие к отсылке уведомления. Вообще, функция слежения каждый раз сообщает только об одном изменении, сделанном одним пользователем. Вероятно, было бы более полезно, если бы отдельно отслеживались целые рабочие каталоги, поэтому такое поведение было бы полезно изменить.

### 10.6.3 Как редактировать файлы, за которыми наблюдают?

Так как файл, за которым следит кто-либо, извлекается в режиме только для чтения, то вы не можете просто взять и отредактировать его. Для того, чтобы сделать его доступным для записи и сообщить остальным, что вы планируете отредактировать этот файл, используйте команду `cvs edit`. Некоторые системы называют это *извлечение*, но пользователи CVS уже используют этот термин в смысле "получение копии исходных текстов" (см. Раздел 1.3.1 [Получение исходного кода], с. 4), а эту операцию, в свою очередь, другие системы называют *взять*.

**cvs edit** [*ключи*] *файлы* . . . Команда

Подготовить для редактирования рабочие файлы. CVS делает *файлы* доступными для чтения и записи и уведомляет пользователей, которые уведомления о редактировании какого-нибудь из указанных файлов.

Команда `cvs edit` принимает такие же ключи командной строки, что и команда `cvs watch add`, и устанавливает временное слежение за *файлами* для пользователя; CVS прекратит слежение, когда будет выполнена команда `unedit` или команда `commit`. Если пользователь не хочет получать уведомления, он должен указать ключ `-a none`.

*Файлы* и ключи командной строки обрабатываются точно так же, как и для команды `cvs watch`.

**Предупреждение:** если в репозитории разрешена опция `PreservePermissions` (см. Раздел C.12 [Файл config], с. 145), то CVS не будет менять прав доступа к *файлам*. Причина этого изменения — убедиться, что `cvs edit` не мешает хранению прав доступа к файлам в CVS-репозитории.

Обычно, когда вы закончите редактирование файлов, используйте команду `cvs commit`, которая проверит ваши изменения и вернет файлы, за которыми производилось слежение, в обычное состояние только для чтения. Если же вы вместо этого решите отменить изменения, или просто не станете ничего менять, используйте команду `cvs unedit`.

**cvs unedit** [`-lR`] *files* . . . Command

Отбросить все изменения в рабочих файлах *files* и привести их в соответствие с ревизией в репозитории. Если кто-либо запросил уведомление об изменениях каких-либо файлов, то CVS делает эти файлы доступными только для чтения. CVS уведомляет пользователей, которые запросили уведомление о команде `unedit`.

Ключи командной строки и список файлов обрабатываются точно так же, как для команды `cvs watch`.

Если слежение не используется, команда `unedit`, вероятно, не работает, и единственный способ вернуть файл в то состояние, в котором он находится в репозитории — удалить его и использовать `cvs update` для получения новой копии. Семантика этой операции идентична команде `unedit`: удаление и обновление может внести также и изменения, которые были помещены в репозиторий с тех пор, как вы в последний раз обновляли свою рабочую копию.

При использовании сетевого CVS вы можете использовать команды `cvs edit` и `cvs unedit`, даже если CVS не смогла успешно соединиться с сервером. Уведомления будут посланы при следующем успешном выполнении какой-либо команды CVS.

### 10.6.4 Информация о том, кто следит и кто редактирует

**cvs watchers** [`-lR`] *files . . .* Команда

Выдает список пользователей, которые отслеживают изменения в *files*. Сообщаются имена файлов и почтовые адреса каждого следящего.

Ключи командной строки и список файлов обрабатываются так же, как и в команде `cvs watch`.

**cvs editors** [`-lR`] *files . . .* Команда

Выдает список пользователей, которые в текущий момент работают над файлами *files*. Сообщаются почтовые адреса пользователей, время, когда пользователь начал работу с файлом, а также машина и рабочий каталог на ней, в котором находится каждый файл.

Список файлов и ключи командной строки обрабатываются точно так же, как и в команде `cvs watch`.

### 10.6.5 Использование слежений со старыми версиями CVS

Если вы используете возможность слежения за репозиторием, то в нем создаются каталоги '`CVS/`', в которых хранится информация о слежениях за файлами из соответствующего каталога. Если вы попытаетесь использовать в этом репозитории CVS версии 1.6 и ранее, вы получите такое сообщение об ошибке:

```
cvs update: cannot open CVS/Entries for reading:
No such file or directory
```

Выполняемая команда, скорее всего, будет прервана. Для использования возможности слежения обновите все копии CVS, которые используют этот репозиторий в локальном или серверном режиме. Если вы не можете совершить обновление, используйте команды `watch off` и `watch remove`, чтобы удалить все слежения, а затем восстановите репозиторий в состояние, с которым может работать CVS 1.6.

## 10.7 Выбор между блокированными и неблокированными извлечениями

Блокированные и неблокированные извлечения имеют свои "за" и "против". Достаточно сказать, что это в основном вопрос мнения, или же принятого в группе стиле работы. Впрочем же, вот краткое описание некоторых возникающих вопросов. Существует множество способов организовать команду разработчиков. CVS не пытается насильно внедрить какой-либо способ. CVS — это просто инструмент, который можно использовать различными способами.

Блокированные извлечения могут оказывать отрицательное влияние на производительность. Если два человека хотят отредактировать различные части файла, какие могут быть причины помешать кому-нибудь из них сделать это? Обычным делом также является заблокировать файл, предполагая поредактировать его, а затем забыть снять блокировку.

Люди, обычно те, кто хорошо знаком с блокированными извлечениями, обычно спрашивают, как часто случаются конфликты при использовании неблокированных извлечений, и сколь сложно их разрешить. Опыт разнообразных групп показал, что такие конфликты случаются редко и обычно их можно разрешить относительно спокойно.

Редкость серьёзных конфликтов может удивить, пока не осознаешь, что они случаются только когда два разработчика расходятся во мнениях о должном дизайне определенного куска кода; такое расхождение свидетельствует о том, что команда, прежде всего, не общалась друг с другом должным образом. Для того, чтобы сотрудничать в условиях *любой* системы контроля исходных текстов, разработчики должны не иметь разногласий по поводу общего дизайна системы; при отсутствии таковых, пересекающиеся изменения обычно разрешаются напрямую.

В некоторых случаях неблокированные извлечения совершенно точно являются неподходящими. Если для файлов, которые вы редактируете, не существует инструмента для слияния (например, файлы, созданные текстовым процессором, или же файлы, созданные с помощью CAD-системы), а переход на программу, которая использует формат файлов с возможностью слияния, нежелателен, то разрешение конфликтов, скорее всего, будет столь неприятным, что будет проще избежать их, используя блокированные извлечения.

Возможность слежения, описанная выше, в главе Раздел 10.6 [Слежения], с. 68, может считаться промежуточной моделью между блокированными и неблокированными изменениями. Когда вы редактируете файл, можно узнать, кто ещё редактирует его. Система, вместо того, чтобы просто запретить двум людям редактировать файл, может описать ситуацию и позволить вам самому решить, является ли этот конкретный случай проблемой или нет. Таким образом, для некоторых групп механизм слежения может считаться объединением лучших черт блокированных и неблокированных изменений.

# 11 Управление ревизиями

Если вы дочитали до этого места, вы, вероятно, уже достаточно хорошо понимаете, что может сделать для вас CVS. В этой главе рассказывается ещё немного о том, что вам предстоит решить для себя.

Если вы занимаетесь разработкой в одиночку, используя CVS, вы, вероятно, можете пропустить эту главу. Вопросы, поднимаемые в этой главе, становятся важны, когда с репозиторием работает более одного пользователя.

## 11.1 Когда фиксировать изменения?

В вашей группе должны решить, какую политику применять по отношению к фиксированию изменений. Возможно несколько подходов, и вы, вероятно, найдете тот, что устраивает вас, по мере наращивания опыта.

Если вы фиксируете изменения слишком быстро, вы можете зафиксировать файлы, которые даже не будут компилироваться. Если ваш партнер обновит свою рабочую копию, в ней появится ваш файл с ошибкой, и он не сможет скомпилировать проект. С другой стороны, если вы будет фиксировать изменения очень редко, то другие участники не смогут воспользоваться вашими улучшениями, и конфликты станут появляться чаще.

Обычно изменения фиксируются, убедившись, по крайней мере, что измененные файлы компилируются. В некоторых организациях требуют, чтобы файлы прошли серию тестов. Подобную политику можно вести с помощью файла ‘`commitinfo`’ (см. Раздел C.4 [Файл commitinfo], с. 137), но следует дважды подумать, прежде чем установить такое требование. Чрезмерно увеличив контроль над разработкой, можно добиться отрицательного воздействия на процесс достижения цели, то есть написание работающего продукта.

# 12 Подстановка ключевых слов

Пока вы редактируете исходные файлы в рабочем каталоге, вы всегда можете узнать их статус с помощью ‘`cvs status`’ и ‘`cvs log`’. Как только вы экспортируете файлы из вашей среды разработки, становится гораздо сложнее узнать, какую ревизию имеют эти файлы.

Для того, чтобы помочь в идентификации файлов, CVS может использовать механизм, известный как *подстановка ключевых слов* (или *замена ключевых слов*). Строки вида `$`*ключевое_слово*`$` и `$`*ключевое_слово*`:...$` в файле заменяются строками вида `$`*ключевое_слово*`:`*значение*`$` каждый раз, когда вы получаете новую ревизию файла.

## 12.1 Список ключевых слов

Вот список ключевых слов:

`$Author$` Имя пользователя, который поместил ревизию в репозиторий.

`$Date$` Дата и время (в UTC), когда была зафиксирована ревизия.

`$Header$` Стандартный заголовок, содержащий полное имя RCS-файла, номер ревизии, дату в UTC, имя автора, состояние и имя пользователя, заблокировавшего этот файл (если файл заблокирован). Файлы обычно не блокируются при использовании CVS.

`$Id$` Точно так же, как `$Header$`, только имя RCS-файла указано без полного пути.

`$Name$` Имя метки, использованной при извлечении этого файла. Это ключевое слово подставляется, только если при извлечении было явно задано имя метки. Например, при выполнении команды `cvs co -r first` это ключевое слово заменяется на ‘`Name: first`’.

`$Locker$` Имя пользователя, который заблокировал эту ревизию (пустое, если файл не заблокирован, как обычно и бывает, если не использовалась команда `cvs admin -l`).

`$Log$` Журнальное сообщение, которое было введено во время фиксации изменений, перед которым идет имя RCS-файла, номер ревизии, имя автора и дата в UTC. Существующие журнальные сообщения *не* заменяются. Вместо этого, новое журнальное сообщение добавляется после `$Log:...$`. Каждая новая строка содержит в начале ту же самую строку, которая находится перед ключевым словом `$Log$`. Например, если в файле находится

```
/* Here is what people have been up to:
 *
 * $Log: frob.c,v $
 * Revision 1.1  1997/01/03 14:23:51  joe
 * Add the superfrobnicate option
 *
 */
```

то перед дополнительными строками, которые добавляются при замене ключевого слова `$Log$`, будет находиться ‘ * ’. В отличие от предыдущих версий CVS и RCS, *префикс комментария* из RCS-файла не используется. Ключевое слово `$Log$` полезно при накоплении полного журнала изменений в исходном файле, но по нескольким причинам это может привести к определенным проблемам. См. Раздел 12.5 [Ключевое слово Log], с. 80.

`$RCSfile$`
: Имя RCS-файла без полного пути.

`$Revision$`
: Номер ревизии.

`$Source$` Полное имя RCS-файла.

`$State$` Состояние, присвоенное ревизии. Состояния могут назначаться с помощью `cvs admin -s` — см. Раздел A.6.1 [Ключи команды admin], с. 95.

## 12.2 Использование ключевых слов

Для того, чтобы поместить в файл ключевое слово, вы просто пишете в нём, например, `$Id$`, а затем фиксируете файл. CVS автоматически заменит ключевое слово во время операции фиксирования.

Обычной практикой является помещение строки `$Id$` в исходные файлы, чтобы они оказались в скомпилированных объектных файлах. Например, если вы управляете исходными текстами программы, вы можете создать переменную, в которую при инициализации попадает строка с `$Id$`. Некоторые компиляторы языка C поддерживают директиву `#pragma ident`. Система управления документами может обеспечивать способ для передачи этой строки в результирующие файлы.

Программа `ident`, являющаяся частью пакета RCS, может использоваться для извлечения из файла ключевых слов и их значений. Это полезно и для работы с текстовыми файлами, но особенно полезно для извлечения ключевых слов из двоичных файлов.

```
$ ident samp.c
samp.c:
     $Id: samp.c,v 1.5 1993/10/19 14:57:32 ceder Exp $
$ gcc samp.c
$ ident a.out
a.out:
     $Id: samp.c,v 1.5 1993/10/19 14:57:32 ceder Exp $
```

SCCS — еще одна популярная система контроля ревизий. В её состав входит программа `what`, очень похожая на `ident` и использующаяся в тех же целях. Во многих местах, где не установлен пакет RCS, стоит SCCS. Так как `what` ищет последовательность символов `@(#)`, то можно довольно просто вставлять ключевые слова, которые обнаруживаются обеими программами. Просто поместите перед ключевым словом в стиле RCS волшебную фразу в стиле SCCS, например:

```
static char *id="@(#) $Id: ab.c,v 1.5 1993/10/19 14:57:32 ceder Exp $";
```

## 12.3 Как избежать подстановки

Подстановка ключевых слов имеет свои недостатки. Иногда бывает нужно, чтобы строка '`$Author$`' появилась в файле без того, чтобы CVS интерпретировала её как ключевое слово и заменила на что-нибудь типа '`$Author: ceder $`'.

К сожалению, нельзя выборочно отключить подстановку ключевых слов. Можно лишь использовать ключ командной строки '`-ko`' (см. Раздел 12.4 [Режимы подстановки], с. 79), чтобы полностью выключить эту подстановку.

Во многих случаях вам нужно избежать использования ключевых слов в исходном тексте, даже несмотря на то, что они появятся в конечном продукте. Например, исходный текст этого руководства содержит '`$@asis{}Author$`' везде, где должна появиться строка '`$Author$`'. При использовании `nroff` и `troff` можно поместить в ключевое слово нулевой символ `\&`, чтобы добиться подобного эффекта.

## 12.4 Режимы подстановки

Вместе с каждым файлом хранится его режим подстановки ключевых слов по умолчанию, и каждая копия файла в рабочем каталоге также имеет режим подстановки. Режим по умолчанию задается с помощью ключа '`-k`' команд `cvs add` и `cvs admin`; режим в рабочем каталоге задается с помощью ключей '`-k`' и '`-A`' команд `cvs checkout` и `cvs update`. Команда `cvs diff` также имеет ключ '`-k`'. Некоторые примеры приведены в Глава 9 [Двоичные файлы], с. 59.

Доступные режимы таковы:

'`-kkv`' Генерировать строки из ключевых слов стандартным образом, то есть из ключевого слова `Revision` получается `$Revision: 5.7 $`.

'`-kkvl`' Подобно '`-kkv`', но только всегда указывается имя заблокировавшего пользователя, если данная ревизия в настоящий момент заблокирована. Имя блокировщика имеет смысл только если используется `cvs admin -l`.

'`-kk`' Генерировать только имена ключевых слов и опускать их значения. Например, для ключевого слова `Revision` получается строка `$Revision$`, а не `$Revision: 5.7 $`. Этот ключ полезен для игнорирования изменений, возникающих из-за ключевых слов, при сравнении разных ревизий файла.

'`-ko`' Генерирует старую строку, присутствовавшую в рабочем файле перед тем, как он был зафиксирован. Например, для ключевого слова `Revision` генерируется строка `$Revision: 1.1 $` вместо `$Revision: 5.7 $`, если она была записана именно так, когда файл был помещен в репозиторий.

'`-kb`' Подобно '`-ko`', но также предотвращает преобразование символов конца строк между канонической формой, в которой они хранятся в репозитории (только символ перевода строки), и формой, принятой в используемой операционной системе. Для UNIX-подобных систем, в которых для завершения строк используется символ перевода строки, этот режим совпадает с '`-ko`'. Дополнительная информация о двоичных файлах находится в Глава 9 [Двоичные файлы], с. 59.

‘`-kv`’ Генерирует только значения ключевых строк. Например, для ключевого слова `Revision` генерируется строка `5.7` вместо `$Revision: 5.7 $`. Это может помочь при генерации файлов на языках программирования, в которых сложно извлечь из строки разделители ключевых слов, такие как как `$Revision: $`. Однако, дальнейшая подстановка ключевых слов не может быть осуществлена, когда удалены ключевые слова, поэтому этот ключ нужно использовать осторожно.

Часто бывает полезно использовать ‘`-kv`’ совместно с командой `cvs export` — см. Раздел A.10 [Команда export], с. 107. Помните только, что этот ключ некорректно экспортирует двоичные файлы.

## 12.5 Проблемы с ключевым словом $Log$.

Ключевое слово `$Log$` довольно-таки спорно. Пока вы работаете над проектом, информация легко доступна даже без использования ключевого слова `$Log$`: просто вызовите `cvs log`. Когда вы экспортируете файл, информация об его истории в любом случае практически бесполезна.

Более серьёзным обстоятельством является то, что CVS не слишком хорошо справляется с пунктами `$Log$`, когда ветка объединяется с основным стволом. В результате такого объединения часто возникают конфликты.

Люди часто стараются "исправить" журнальные записи в файле, исправляя орфографические и даже фактические ошибки. В результате информация от `cvs log` не совпадает с информацией в файле. Это может стать (а может и не стать) проблемой при реальной работе.

Звучали рекомендации помещать ключевое слово `$Log$` в конец файла (если вообще нужно использовать это слово). В этом случае длинный список сообщений об изменениях не будет мешать чтению исходного файла.

# 13 Слежение за чужими исходными текстами

Если вы изменяете программу, чтобы она лучше соответствовала вашим нуждам, то вам, вероятно, захочется еще раз сделать те же самые изменения, когда появляется новая версия программы. CVS поможет вам с этой задачей.

В терминологии, используемой в CVS, лицо или организация, предоставившая вам текст программы, называется *поставщиком*. Неизмененное дерево каталогов из комплекта поставки помещается на отдельную ветку, которая называется *ветка производителя*. CVS резервирует для этой цели ветку с номером '`1.1.1`'.

Когда вы изменяете исходный текст и фиксируете изменения, то они оказываются в основном стволе. Когда поставщик выпускает новую версию программы, вы помещаете её на ветку производителя и копируете изменения в основной ствол.

Используйте команду `import` для создания и обновления ветки производителя. Когда вы импортируете новый файл, ветка производителя становится "головной" ревизией (`HEAD`), поэтому все, кто извлекает копию файла, получают эту ревизию. Когда локальные модификации фиксируются, они помещаются в основной ствол и становятся "головной" (`HEAD`) ревизией.

## 13.1 Начальный импорт

Используйте команду `import` для начального помещения исходных текстов в репозиторий. Для отслеживания чужих исходников полезно использовать *метки поставщика* и *метки релизов*. *Метка поставщика* — это символическое имя ветки (всегда имеющей номер '`1.1.1`', если вы не использовали флаг '`-b` *ветка*'. См. Раздел 13.6 [Несколько веток поставщика], с. 83. *Метки релизов* — это символические имена конкретной версии продукта, например, '`FSF_0_04`'.

Заметьте, что `import` *не* изменяет содержимое каталога, в котором вызвана эта команда. В частности, этот каталог не становится рабочим каталогом CVS; если вы хотите работать с исходными текстами, сначала импортируйте их, затем извлеките их в другой каталог (см. Раздел 1.3.1 [Получение исходного кода], с. 4).

Предположим, что у вас имеются исходные тексты программы, которая называется `wdiff`, находящиеся в каталоге '`wdiff-0.04/`', и вы хотите сделать изменения для себя, и сохранить их, когда появится новая версия. Начните с импорта исходных текстов в репозиторий:

```
$ cd wdiff-0.04
$ cvs import -m "Import of FSF v. 0.04" fsf/wdiff FSF_DIST WDIFF_0_04
```

В вышеприведенном примере метка поставщика называется '`FSF_DIST`', а единственная присвоенная метка релиза — '`WDIFF_0_04`'.

## 13.2 Обновление с помощью импорта

Когда появляется новая версия исходных текстов, вы импортируете их в репозиторий с помощью такой же команды `import`, которую вы использовали для начального импорта в репозиторий. Единственное различие — теперь вы используете другую метку релиза.

```
$ tar xfz wdiff-0.05.tar.gz
$ cd wdiff-0.05
$ cvs import -m "Import of FSF v. 0.05" fsf/wdiff FSF_DIST WDIFF_0_05
```

Для файлов, которые не были локально изменены, новые созданные ревизии становятся головными ревизиями. Если же локальные изменения были сделаны, команда `import` предупредит вас, что вы должны слить изменения в основной ствол и посоветует использовать для этого команду '`checkout -j`'.

```
$ cvs checkout -jFSF_DIST:yesterday -jFSF_DIST wdiff
```

Вышеуказанная команда извлечет последнюю версию '`wdiff`', объединяя в рабочую копию изменения, сделанные на ветке поставщика '`FSF_DIST`' со вчерашнего дня. Если в процессе слияния появляются конфликты, то их нужно разрешить обычным способом (см. Раздел 10.3 [Пример конфликта], с. 65), затем измененные файлы можно зафиксировать.

Использование даты, как предлагается выше, предполагает, что вы импортируете не более одной версии продукта в день. Если же это не так, вы всегда можете использовать что-нибудь типа такого:

```
$ cvs checkout -jWDIFF_0_04 -jWDIFF_0_05 wdiff
```

В этом случае вышеприведенные команды эквивалентны.

## 13.3 Возврат к последней версии от поставщика

Вы также можете полностью отменить локальные изменения и вернуться к последней версии от поставщика, установив головную ревизию в ветку поставщика для всех файлов. Например, если у вас есть извлеченная копия исходников в '`~/work.d/wdiff`', и вы хотите вернуть все файлы в этом каталоге в то состояние, в котором вам предоставил их поставщик, наберите:

```
$ cd ~/work.d/wdiff
$ cvs admin -bWDIFF .
```

Вы должны написать '`-bWDIFF`' без пробела после '`-b`'. См. Раздел A.6.1 [Ключи команды admin], с. 95.

## 13.4 Как обрабатывать двоичные файлы при импорте в CVS

Используйте ключ '`-k`', чтобы сказать команде '`import`', что файлы являются двоичными. См. Раздел C.2 [Обертки], с. 136.

## 13.5 Как обрабатывать замену ключевых слов при импорте в CVS

Исходные тексты, которые вы импортируете, могут содержать ключевые слова (см. Глава 12 [Подстановка ключевых слов], с. 77). Например, поставщик мог использовать CVS или какую-нибудь другую систему, которая использует похожий синтаксис подстановки ключевых слов. Если вы просто импортируете файлы, то содержимое ключевых слов будет заменены вашим CVS. Вероятно, более удобно будет оставить

ключевые слова в том виде, в котором они были у вашего поставщика, чтобы сохранить информацию об исходных текстах.

Чтобы сохранить содержимое ключевых слов, добавьте к команде `cvs import` при первом импорте ключ '`-ko`'. Это полностью выключит замену ключевых слов для этого файла, поэтому если хотите, то можете выборочно использовать разные ключи '`-k`' с командами `cvs update` и `cvs admin`.

## 13.6 Несколько веток поставщика

Все примеры до этого момента подразумевали, что есть только один поставщик, от которого вы получаете исходные тексты. В некоторых ситуациях вы можете получать эти тексты из разных мест. Например, предположим, что вы работаете в проекте, в котором много различных людей и команд изменяют исходный код продукта. Существуют различные способы работы в такой ситуации, но иногда у вас есть несколько деревьев исходников, лежащих рядом, и всё, что вы хотите сделать — это положить их под CVS, чтобы они по крайней мере находились в одном месте.

Для обработки ситуаций, в которых поставщиков больше одного, вы можете задать команде `cvs import` ключ '`-b`'. Этот ключ принимает в качестве параметра номер ветки поставщика. По умолчанию используется '`-b 1.1.1`'.

Например, предположим, что есть две команды: красная и синяя, и обе посылают вам исходные тексты. Вы хотите импортировать результаты работы красной команды в ветку '`1.1.1`' и использовать метку поставщика '`RED`'. Вы также хотите импортировать работу синей команды на ветку '`1.1.3`' и использовать метку поставщика '`BLUE`'. В этом случае можно использовать такие команды:

```
$ cvs import dir RED RED_1-0
$ cvs import -b 1.1.3 dir BLUE BLUE_1-5
```

Заметьте, что если метка поставщика не соответствует ключу '`-b`', то CVS не обратит на это внимания! Например,

```
$ cvs import -b 1.1.3 dir RED RED_1-0
```

Будьте осторожны; такие ошибки совершенно точно приведут, в лучшем случае, к недоразумениям. Не могу придумать полезной цели для возможности намеренного несовпадения в этом месте, а если вы придумаете такой способ, то не используйте его. Скорее всего, в следующих версиях CVS это станет ошибкой.

# 14 Как ваша система сборки взаимодействует с CVS

Как упоминалось во введении, CVS не содержит средств для сборки вашего проекта из исходных текстов. В этой части описывается, как CVS взаимодействует с разными аспектами вашей системы сборки.

Обычным вопросом, особенно для людей, знакомых с RCS, является "как сделать так, чтобы проект компилировался из самых свежих исходных текстов". Ответ на этот вопрос неоднозначен. Во-первых, так как CVS может рекурсивно обходить дерево каталогов, то не требуется изменять ваши файлы '`Makefile`' (или тот файл, что используется вашими инструментами для сборки), чтобы убедиться, что каждый файл является самым свежим. Вместо этого просто используйте две команды, сначала `cvs -q update`, а затем `make` или ту команду, которую вы выполняете, чтобы запустить вашу систему сборки. Во-вторых, вам не обязательно требуется получать копии изменений, которые сделал кто-то другой, пока вы не закончили свою собственную работу. Рекомендуется сначала обновить ваши исходные тексты, затем сделать изменение, собрать проект, проверить его, а затем зафиксировать изменения (сначала опять обновив исходники, если требуется). Периодически (в промежутке между изменениями, используя описанный подход) обновляя всё дерево исходных текстов, вы остаётесь в уверенности, что ваши исходники достаточно свежи.

Обычно необходимо записывать, какие ревизии исходных файлов использовались для построения конкретной версии продукта. Такая функциональность иногда называется *списком использованных материалов*. Лучший способ добиться этого с помощью CVS — использовать команду `tag` (см. Раздел 4.4 [Метки], с. 34).

Используя CVS простейшим способом, каждый разработчик будет иметь копию всего дерева исходников, которые использовались в конкретных версиях проекта. Если дерево небольшое, или если разработчики удалены друг от друга географически, то это — предпочтительное решение. В действительности ещё одним подходом к большим проектам является разбить их на меньшие подсистемы, компилируемые по отдельности, и обустроить механизм внутренних релизов, чтобы каждый разработчик мог извлекать те подсистемы, над которыми он активно работает.

Другой подход — создать структуру, позволяющую разработчикам иметь собственные копии некоторых файлов, а за остальными файлами обращаться в центральное хранилище. Многие предлагали подобные системы, используя такие возможности, как символические ссылки, существующие во многих операционных системах, или механизм `VPATH`, поддерживаемый многими версиями программы `make`. Например, одним из инструментов для сборки проекта, разработанным для этого, является Odin (см. `ftp://ftp.cs.colorado.edu/pub/distribs/odin`).

# 15 Специальные файлы

В обычных условиях CVS работает только с обычными файлами. Каждый файл в проекте считается устойчивым: его можно открыть, прочесть и закрыть, и делать это несколько раз. CVS также игнорирует права доступа и владельцев файлов, предоставляя разработчику решать такие вопросы во время инсталляции. Другими словами, нельзя "поместить" устройство в репозиторий; если устройство не может быть открыто, CVS откажется обрабатывать его. Файлы также могут потерять права доступа и владельцев во время транзакций в репозитории.

Если в репозитории установлена переменная `PreservePermissions` (см. Раздел C.12 [Файл config], с. 145), то CVS сохранит в репозитории такие характеристики файлов:

- пользователь- и группа-владелец файла
- права доступа
- основной и вспомогательный номер устройства
- символические ссылки
- структуру жестких ссылок

Использование опции `PreservePermissions` влияет на поведение CVS в нескольких местах. Во-первых, некоторые новые операции, поддерживаемые CVS не доступны всем пользователям. В частности, владельцы файла и характеристики специальных файлов могут изменяться только суперпользователем. Таким образом, когда переменная конфигурации `PreservePermissions` установлена, пользователи должны стать пользователем `root`, чтобы выполнять операции с CVS.

Когда используется `PreservePermission`, некоторые операции CVS (такие, как '`cvs status`'), не смогут определить структуру жестких ссылок файла, и будут выдавать предупреждения о несовпадающих жестких ссылках. Причиной этого является то, что внутренние структуры CVS не обеспечивают простого способа собрать всю информацию о жестких ссылках, поэтому они проверяют конфликты файлов, пользуясь неточными данными.

Более тонким различием является то, что CVS считает, что файл изменился, только если изменилось его содержимое (особенно если время модификации рабочего файла не совпадает с временем модификации файла в репозитории). Таким образом, если у файла изменились только права доступа, владелец, или основной и вспомогательный номера устройства, то CVS не заметит этого. Для того, чтобы принудительно поместить такое изменение в репозиторий, используйте ключ '`-f`' команды '`cvs commit`'. Это также означает, что если права доступа файла изменились, а файл в репозитории новее, чем рабочая копия, то выполнение '`cvs update`' просто изменит права доступа на рабочей копии.

Изменение жестких ссылок в репозитории CVS является особенно тонкой процедурой. Предположим, что файл '`foo`' был связан с файлом '`old`', а затем связан с файлом '`new`'. Вы можете попасть в необычную ситуацию, когда несмотря на то, что у '`foo`', '`old`' и '`new`' изменилось количество жестких ссылок, только файлы '`foo`' и '`new`' были изменены, поэтому '`old`' не является кандидатом на фиксирование в репозитории. Таким образом можно прийти к результатам, несоответствующим действительности. Если необходимо хранить в репозитории жесткие ссылки, мы рекомендуем применять

команду `touch` ко всем файлам, чьи ссылки или статус изменились со времени последней фиксации. В действительности, было бы правильным выполнять `touch *` перед каждой фиксацией, если в каталоге находится сложная система ссылок на файлы.

Стоит заметить, что только обычные файлы могут быть объединены, по причинам, которые, надеемся, очевидны. Если ‘`cvs update`’ или ‘`cvs checkout -j`’ попытаются объединить символическую ссылку с обычным файлом, или два файла устройств друг с другом, то CVS сообщит о конфликте и откажется производить объединение. В то же самое время ‘`cvs diff`’ не сообщит о различиях между этими файлами, потому что с файлами, не содержащими текста, нельзя совершать текстуальные сравнения.

Параметр `PreservePermissions` не работает с клиент-серверной версией CVS. Другим ограничением является то, что жесткие ссылки должны быть между файлами, находящимися в одном каталоге; жесткие ссылки между разными каталогами не поддерживаются.

# Приложение A Руководство по командам CVS

В этом приложении описывается общая структура команд CVS, а некоторые команды описываются детально; краткий справочник по командам CVS находится в см. Приложение B [Вызов CVS], с. 121.

## A.1 Общая структура команд CVS

Общий формат всех команд CVS таков:

```
cvs [ опции_cvs ] команда_cvs [ опции_команды ] [ аргументы_команды ]
```

`cvs` Имя исполняемого файла CVS.

`cvs_options`
Некоторые ключи командной строки влияют на все подкоманды CVS. Такие ключи описаны ниже.

`cvs_command`
Одна из нескольких подкоманд. Некоторые команды имеют синонимы, которые указываются в справочнике по этой команде. Есть только две ситуации, в которых вы можете пропустить '`команду_cvs`': '`cvs -H`' выдает список доступных команд, а '`cvs -v`' отображает версию CVS.

`command_options`
Ключи командной строки, специфичные для каждой команды.

`command_args`
Аргументы команд.

К сожалению, есть небольшая путаница между **опциями_cvs** и **опциями_команды**. Ключ '`-l`', когда он используется в качестве опции CVS, воздействует только на некоторые команды. Когда этот ключ используется как опция команды, у него появляется другое значение, и он используется с бóльшим количеством команд. Другими словами, не придавайте вышеописанной категоризации слишком большого значения, а обращайтесь вместо этого к документации.

## A.2 Код выхода CVS

CVS может сообщить вызывающей программе, успешно ли завершилась операция или нет, возвращая тот или иной *код выхода*. Точный способ проверки кода выхода зависит от операционной системы. Например, в скриптах оболочки UNIX переменная '`$?`' содержит ноль, если последняя команда возвратила код успешного выхода, или же больше нуля, если выполнение программы завершилось с ошибкой.

Если CVS выполняется успешно, то возвращает код успешного завершения; в случае ошибки программа печатает сообщение об ошибке и возвращает код неуспешного завершения. Исключением является команда `cvs diff`. Она возвращает код успешного завершения, если не обнаружила различий, или же код неудачного завершения, если были обнаружены различия или произошла ошибка. Так как такое поведение не обеспечивает простого способа обнаружения ошибок, в будущем, вероятно, команда `cvs diff` будет изменена, чтобы вести себя подобно прочим командам CVS.

## A.3 Ключи по умолчанию и файл ~/.cvsrc

Имеются определённые ключи команд CVS, которые используются столь часто, что вы захотите настроить для них что-то типа синонима. Основным примером (именно он и привел к поддержке файла '`~/.cvsrc`') является то, что многим не нравится стандартная форма выдачи изменений, которая используется в команде '`diff`', и они предпочитают контекстную или унифицированную выдачу изменений, которые выглядят значительно лучше.

Файл '`~/.cvsrc`' — это способ установить ключи по умолчанию для команд CVS, не используя синонимов, скриптов оболочки и т. п.

Формат файла '`~/.cvsrc`' прост. В нем ищется строка, чье начало совпадает с именем выполняемой команды CVS. Если совпадающая строка найдена, то остаток строки расщепляется на ключи командной строки и добавляется к командной строке *перед* ключами из настоящей командной строки.

Если у команды есть два имени (например, `checkout` и `co`), то для поиска используется официальное имя, не обязательно совпадающее с тем, что использовалось при вызове CVS. Таким образом, если содержимое файла '`~/.cvsrc`' таково:

```
log -N
diff -u
update -P
checkout -P
```

то к аргументам команды '`cvs checkout foo`' добавится ключ '`-P`', и точно то же самое произойдет с командой '`cvs co foo`'.

При использовании вышеприведенного файла команда '`cvs diff foobar`' будет выдавать изменения в унифицированном формате. '`cvs diff -c foobar`' будет, как обычно, выдавать контекстные изменения. Получение изменений в "старом" формате чуть более сложно, потому что у команды `diff` нет способа задать выдачу в "старом" формате, поэтому вам потребуется использовать '`cvs -f diff foobar`'.

Вместо имени команды вы можете использовать `cvs`, чтобы задать глобальные ключи (см. Раздел A.4 [Глобальные ключи], с. 90). Например, такая строка в файле '`~/.cvsrc`' включит использование шестого уровня компрессии:

```
cvs -z6
```

## A.4 Глобальные ключи командной строки

Вот список имеющихся ключей командной строки CVS (те из них, что задаются слева от имени команды):

`-allow-root=`*rootdir*
: Задает разрешенный каталог `CVSROOT`. См. Раздел 2.9.3.1 [Сервер парольной аутентификации], с. 21.

`-a`
: Аутентифицировать всё общение между клиентом и сервером. Влияет только на клиента. В настоящее время это реализуется только при использовании соединения GSSAPI (см. Раздел 2.9.4 [Аутентификация с помощью GSSAPI], с. 25). Аутентификация предотвращает определённые виды атак,

использующих перехват TCP-соединения. Включение аутентификации не включает шифрование.

`-b` *bindir* В CVS 1.9.18 и более старых версиях этот ключ задавал каталог, в котором находятся программы RCS. Текущие версии CVS не выполняют программ RCS; этот ключ оставлен только для обратной совместимости.

`-T` *tempdir*
Использовать *tempdir* в качестве каталога, в котором расположены временные файлы. Переопределяет содержимое переменной среды `$TMPDIR` и каталог, заданный при компиляции. Этот параметр должен задавать полный путь.

`-d` *cvs_root_directory*
Использовать *cvs_root_directory* в качестве корневого каталога репозитория. Переопределяет содержимое переменной окружения `$CVSROOT`. См. Глава 2 [Репозиторий], с. 7.

`-e` *editor* Использовать *editor*, чтобы ввести журнальное сообщение. Переопределяет содержимое переменных окружения `$CVSEDITOR` и `$EDITOR`. За дальнейшей информацией обращайтесь к Раздел 1.3.2 [Фиксирование изменений], с. 4.

`-f` Не читать файл '`~/.cvsrc`'. Этот ключ чаще всего используется из-за неортогональности набора ключей CVS. Например, команда '`cvs log`' имеет ключ '`-N`' (отключить отображение имен меток), но не имеет соответствующего ключа, чтобы включить такое отображение. Поэтому если у вас в файле '`~/.cvsrc`' для команды '`log`' имеется ключ '`-N`', вам может потребоваться '`-f`', чтобы отобразить имена меток.

`-H`
`-help` Выдать информацию об указанной команде CVS (но не выполнять её). Если вы не укажете имя команды, то '`cvs -H`' выдаёт общую информацию об использовании CVS, включая список других ключей помощи.

`-l` Выполнить команду, не журналируя её в файле истории команд. См. Раздел A.11 [Команда history], с. 108.

`-n` Не изменять ничего на диске. Попытаться выполнить команду CVS, но только выдавать отчёт; не удалять, обновлять или объединять существующие файлы, не создавать новых.

Заметьте, что CVS не обязательно выдаст те же самые сообщения, что и без использования '`-n`', потому что в некоторых случаях CVS пропустит часть работы.

`-Q` Команда вообще не будет выдавать сообщений, за исключением сообщений о серьезных проблемах.

`-q` Команда будет выдавать только некоторые сообщения; например, информация о продвижении по дереву каталогов выдаваться не будет.

`-r` Делать новые рабочие файлы доступными только для чтения. Тот же самый эффект достигается установкой переменной окружения `$CVSREAD` (см. Приложение D [Переменные окружения], с. 147). По умолчанию рабочие

файлы создаются доступными для записи, если только не включено слежение (см. Глава 13 [Слежение за исходными текстами], с. 81).

`-s` *variable*=*value*
: Установить переменную пользователя (см. Раздел C.11 [Переменные], с. 144).

`-t`
: Отслеживать выполнение программы; отображать сообщения, сопровождающие различные шаги CVS. Особенно полезно совместно с '`-n`', чтобы изучить работу незнакомой команды.

`-v`

`-version`
: Отображает версию CVS и информацию об авторских правах.

`-w`
: Делает новые рабочие файлы доступными для чтения и записи. Переопределяет содержимое переменной окружения `$CVSREAD`. Файлы по умолчанию создаются для чтения и записи, если только не был установлен `$CVSREAD` или же не использовался ключ '`-r`'.

`-x`
: Шифровать всё взаимодействие между клиентом и сервером. Воздействует только на клиента CVS. В текущей версии реализовано только при использовании соединения с GSSAPI (см. Раздел 2.9.4 [Аутентификация с помощью GSSAPI], с. 25) или соединения с Kerberos (см. Раздел 2.9.5 [Аутентификация с помощью Kerberos], с. 26). Включение шифрования подразумевает, что в канале связи будет также включена аутентификация. Поддержка шифрования по умолчанию недоступна; при компиляции CVS используйте специальный ключ командной строки к `./configure --enable-encryption`.

`-z` *gzip-level*
: Установить уровень компрессии. Влияет только на клиента CVS.

## A.5 Стандартные ключи командной строки

В этой главе описываются '`ключи_команды`', доступные для использования с несколькими командами CVS. Эти ключи всегда задаются справа от имени '`команды_CVS`'. Не все команды поддерживают эти ключи, но лишь те, для которых ключ имеет смысл. Однако, если команда имеет один из этих ключей, вы можете быть уверены в одинаковом поведении этих ключей с разными командами. (Другие ключи команд, описанные вместе с отдельными командами, могут иметь различное поведение с разными командами CVS).

**Предупреждение:** команда '`history`' является исключением, она поддерживает различные ключи, конфликтующие даже со стандартными ключами.

`-D` *дата*
: Использовать самую свежую ревизию, сделанную не позже чем *дата*. В данном случае *дата* — это одиночный аргумент, являющийся описанием прошедшей даты.

  Указанная дата становится *липкой*, когда вы используете её, чтобы сделать копию файла с исходным текстом, то есть, когда вы извлекаете рабочий файл, используя '`-D`', то CVS запоминает указанную дату, так что

последующие команды обновления, выполненные в том же каталоге, будут использовать ту же дату (дальнейшая информация по липким меткам и датам находится в см. Раздел 4.9 [Липкие метки], с. 38).

Ключ '`-D`' доступен совместно с командами `checkout`, `diff`, `export`, `history`, `rdiff`, `rtag` и `update`. (Команда `history` использует этот ключ немного отличающимся способом; см. Раздел A.11.1 [Ключи команды history], с. 108).

CVS поддерживает большое множество форматов даты. Самыми стандартными являются ISO-8601 (от Международной Организации по Стандартизации) и стандарт электронной почты (описанные в RFC822, с поправками в RFC1123).

Даты в формате ISO-8601 имеют множество вариантов, но вот несколько примеров:

```
1972-09-24
1972-09-24 20:05
```

Вероятно, вы совсем не желаете увидеть перечисление *полного* списка форматов, описанных в ISO8601 :-).

Вдобавок к датам, разрешенным в электронной почте в Интернет, CVS также позволяет пропускать некоторые поля. Например:

```
24 Sep 1972 20:05
24 Sep
```

Считается, что дата находится в местной временно́й зоне, если только таковая не задана явно.

Предпочтительными являются два формата представления данных. Однако же, CVS в настоящее время поддерживает широкий диапазон других форматов представления даты. Они нарочно не документируются здесь, а будущие версии CVS могут уже не поддерживать их.

Одним из таких форматов является *месяц/день/год*. Такой взаимный порядок дня и месяца может смутить некоторых, например, '`1/4/96`' — это четвертое января, а не первое апреля.

Не забудьте написать аргумент команды '`-D`' в кавычках, чтобы ваша оболочка не посчитала пробелы разделителями аргументов. Команда, использующая ключ '`-D`', может выглядеть так:

```
$ cvs diff -D "1 hour ago" cvs.texinfo
```

`-f` Когда вы задаёте команде CVS конкретную дату или метку, то эта команда обычно игнорирует файлы, не содержащие заданной метки (или не существовавшие на указанный момент времени). Используйте ключ '`-f`', если вы хотите, чтобы файлы извлекались, даже если они не совпадают с меткой или со временем, в этом случае будет использована самая свежая ревизия файла.

'`-f`' доступна с командами `annotate`, `checkout`, `export`, `rdiff`, `rtag`, и `update`.

**Предупреждение:** Команды `commit` и `remove` также имеют ключ '`-f`', но он имеет другое поведение. См. Раздел A.8.1 [Ключи команды commit], с. 103, а также Раздел 7.2 [Удаление файлов], с. 52.

`-k` *kflag* Изменить обработку ключевых слов по умолчанию. См. Глава 12 [Подстановка ключевых слов], с. 77, о значении *kflag*. Указанное значение *kflag* становится *липким*, когда вы создаёте личную копию файла. Это означает, что когда вы используете этот ключ вместе с командами `checkout` или `update`, то CVS связывает значение *kflag* с файлом, и использует это значение при последующих командах обновления этого файла, если вы не укажете обратного.

Ключ '`-k`' доступен с командами `add`, `checkout`, `diff`, `import` и `update`.

`-l` Не обходить дерево каталогов, работать только в текущем рабочем каталоге.

**Предупреждение:** это не тот глобальный ключ '`-l`', который вы указываете слева от команды CVS!

Доступен с командами `annotate`, `checkout`, `commit`, `diff`, `edit`, `editors`, `export`, `log`, `rdiff`, `remove`, `rtag`, `status`, `tag`, `unedit`, `update`, `watch`, и `watchers`.

`-m "`*сообщение*`"`
Использовать "*сообщение*" в качестве журнальной записи, вместо того, чтобы запустить редактор.

Флаг доступен с командами `add`, `commit` и `import`.

`-n` Не выполнять соответствующие программы при выполнении команд '`checkout`', '`commit`' и '`tag`'. (В базе данных модулей могут быть указаны программы, которые нужно выполнить при выполнении одной из этих команд, а этот ключ используется для того, чтобы избежать этого).

**Предупреждение:** этот флаг — не тот глобальный флаг '`cvs -n`', который задаётся слева от команды CVS!

Флаг доступен с командами `checkout`, `commit`, `export` и `rtag`.

`-P` Удалять пустые каталоги. См. Раздел 7.3 [Удаление каталогов], с. 53.

`-p` Выдать файлы, извлеченные из репозитория, на стандартный вывод, а не записывать их в текущем каталоге. Флаг доступен с командами `checkout` и `update`.

`-R` Рекурсивно обрабатывать каталоги. Включено по умолчанию. Доступно с командами `annotate`, `checkout`, `commit`, `diff`, `edit`, `editors`, `export`, `rdiff`, `remove`, `rtag`, `status`, `tag`, `unedit`, `update`, `watch` и `watchers`.

`-r` *метка* Использовать ревизию, указанную в параметре *метка*, вместо *головной* ревизии (`HEAD`) по умолчанию. Помимо меток, созданных с помощью команд `tag` и `rtag`, всегда доступны две специальные метки: '`HEAD`' ссылается на самую свежую ревизию, находящуюся в репозитории, а '`BASE`' ссылается на ревизию, которую вы извлекли в текущий рабочий каталог.

Указанная метка становится липкой, если вы используете `checkout` или `update`, чтобы создать собственную копию файла: CVS запоминает метку и продолжает использовать её при дальнейших командах обновления, пока вы не укажете обратного (См. Раздел 4.9 [Липкие метки], с. 38, где можно найти дополнительную информацию о о липких метках/датах). Метка может быть номером ревизии или именем. См. Раздел 4.4 [Метки], с. 34.

Задание глобального ключа '`-q`' вместе с ключом '`-r`' часто бывает полезным, чтобы избежать предупреждающих сообщений о том, что RCS-файл не содержит указанной метки.

**Предупреждение:** не перепутайте этот ключ с глобальным ключом '`cvs -r`', который вы пишете слева от команды CVS!

Ключ '`-r`' доступен с командами `checkout`, `commit`, `diff`, `history`, `export`, `rdiff`, `rtag` и `update`.

`-W` *spec* Задаёт имена файлов, которые нужно отфильтровать. Этот ключ можно использовать в командной строке несколько раз. *spec* может быть шаблоном имени файла, таким же, который можно использовать в файле '`.cvswrappers`'. Ключ доступен с командами `import` и `update`.

## A.6 Команда admin: администрирование

- Требует: репозиторий, рабочего каталога.
- Изменяет: репозиторий.
- Синоним: `rcs`

Эта команда — интерфейс к разнообразным административным возможностям CVS. Некоторые из них имеют сомнительную ценность для CVS и существуют по историческим причинам. Некоторые из таких возможностей, скорее всего, исчезнут когда-либо. Эта команда работает рекурсивно, поэтому нужно соблюдать крайнюю осторожность.

Если на машине под UNIX существует группа `cvsadmin`, то команду `cvs admin` могут выполнять только члены этой группы. Эта группа должна существовать на сервере или на любой машине, на которой используется не-клиент-серверная версия CVS. Чтобы запретить всем пользователям выполнение команды `cvs admin`, создайте соответствующую группу и никого в неё не помещайте. Под NT группы `cvsadmin` не поддерживается, поэтому все пользователи могут выполнять `cvs admin`.

### A.6.1 Ключи команды admin

Некоторые ключи имеют сомнительную полезность для CVS, но существуют по историческим причинам. Некоторые даже приводят к невозможности использования CVS, пока вы не отмените их действие!

`-A`*старый_файл*
Может не работать совместно с CVS. Добавляет список доступа *старого_файла* к списку доступа RCS-файла.

`-a`*имена* Может не работать совместно с CVS. *имена* перечисляются через запятую. Добавляет *имена* к списку доступа RCS-файла.

`-b`[*ревизия*]
: Устанавливает ветку по умолчанию& В CVS вы обычно не манипулируете ветками по умолчанию, вместо этого используются липкие метки (см. Раздел 4.9 [Липкие метки], с. 38). Есть одна причина использовать `cvs admin -b`: вернуть обратно версию от поставщика при использовании веток поставщика (см. Раздел 13.3 [Отмена локальных изменений], с. 82). Между '`-b`' и аргументом не должно быть пробела.

`-c`*строка*
: Делает *строку* префиксом комментария. Этот префикс не используется ни в текущей версии CVS, ни в RCS 5.7, таким образом, о нём можно не беспокоиться. См. Глава 12 [Подстановка ключевых слов], с. 77.

`-e`[*имена*]
: Может не работать совместно с CVS. *имена* перечисляются через запятую. Удаляет *имена* из списка доступа RCS-файла. Если *имена* не заданы, очищает весь список доступа.

`-I`
: Выполняется интерактивно, даже если стандартный ввод не является терминалом. Этот ключ не работает с сетевой версией CVS и, скорее всего, исчезнет в будущих версиях CVS.

`-i`
: Бесполезный ключ. Создает новый RCS-файл, не создавая новой ревизии. Файлы можно добавлять в репозиторий с помощью команды `cvs add` (см. Раздел 7.1 [Добавление файлов], с. 51).

`-k`*subst*
: Устанавливает режим подстановки ключевых слов по умолчанию. См. Глава 12 [Подстановка ключевых слов], с. 77. Явное указание режима подстановки при использовании команд `cvs update`, `cvs export` и `cvs checkout` переопределяет этот режим по умолчанию.

`-l`[*rev*]
: Блокировать ревизию с номером *rev*. Если указан номер ветки, заблокировать самую последнюю ревизию на ветке. Если *rev* опущено, заблокировать последнюю ревизию на ветке по умолчанию. Между '`-l`' и аргументом не может быть пробела.

  Этот ключ можно использовать в сочетании со скриптом '`rcslock.pl`', находящимся в каталоге '`contrib/`' в дистрибутиве CVS, для того, чтобы пользоваться блокированными извлечениями (когда только один пользователь в каждый момент времени может редактировать данный файл). Смотрите комментарии в этом файле за дальнейшей информацией (а также файл '`README`' в каталоге '`contrib/`', где содержится уведомление об отсутствующей поддержке для содержимого этого каталога.) В соответствии с вышеупомянутыми комментариями следует установить жесткий режим блокировки (по умолчанию это именно так).

`-L`
: Установить жесткий режим блокировки. Это означает, что владелец RCS-файла наравне со всеми прочими должен блокировать файл перед внесением в него изменений. Для работы с CVS жесткий режим блокировки должен быть установлен; смотри обсуждение этого вопроса в описании ключа '`-l`'.

`-m`*rev*`:`*msg*
: Заменить журнальное сообщение ревизии *rev* на *msg*.

`-N`*имя*`[:[`*rev*`]]`

Действует так же, как '`-n`', переопределяя уже существующее *имя*. Об использовании с волшебными ветками смотри Раздел 5.5 [Волшебные номера веток], с. 44.

`-n`*имя*`[:[`*rev*`]]`

Связывает алфавитное *имя* с веткой или ревизией *rev*. Обычно вместо этого ключа лучше использовать '`cvs tag`' и '`cvs rtag`'. Если двоеточие и *rev* опущены, удаляет *имя*; в противном случае сообщает об ошибке, если *имя* уже связано с каким-либо номером. Если *rev* является алфавитным именем, то оно перед связыванием заменяется на соответствующий номер. Если *rev* состоит из номера ветви, за которым следует точка, то это означает самую свежую ревизию на ветке. Двоеточие с отсутствующим *rev* означает самую свежую ревизию на ветке по умолчанию, или на стволе. Например, '`cvs admin -n`*name*' связывает *name* с последней ревизией всех RCS-файлов; подобно этому '`cvs admin -n`*name*`:$`' связывает *name* с номерами ревизий, извлеченных из содержимого ключевых слов в соответствующих рабочих файлах.

`-o`*диапазон*

Удаляет (*делает устаревшими*) ревизии, заданные *диапазоном*.

Заметьте, что эта команда может быть весьма опасна, если только вы не знаете *точно*, что именно вы делаете (например, смотрите предупреждения о возможной путанице в синтаксисе *rev1*:*rev2*).

Если вам не хватает дискового пространства, то эта команда может вам помочь. Подумайте дважды перед её использованием: отменить действие этой команды нельзя никак, кроме восстановления с резервных копий. Вероятно, неплохо будет сначала поэкспериментировать на копии репозитория.

*диапазон* задаётся одним из нескольких способов:

*rev1*`::`*rev2*

Уничтожить ревизии между *rev1* и *rev2*, так что CVS будет хранить только изменения между *rev1* и *rev2*, но не промежуточные изменения. Например, после выполнения команды '`-o 1.3::1.5`' можно извлечь ревизию 1.3, ревизию 1.5, разницу между 1.3 и 1.5, но не ревизию 1.4 или разницу между 1.3 и 1.4. Другой пример: '`-o 1.3::1.4`' и '`-o 1.3::1.3`' не совершат никакого действия, потому что удалять в данных случаях нечего.

`::`*rev*

Удаляет ревизии между началом ветки, содержащей *rev*, и самой *rev*. Точка начала ветки и *rev* остаются нетронутыми. Например, '`-o ::1.3.2.6`' удаляет ревизию 1.3.2.1, ревизию 1.3.2.5 и все ревизии в промежутке между ними, но не трогает 1.3 и 1.3.2.6.

*rev*`::`

Удаляет ревизии между *rev* и концом ветки, содержащей *rev*. Ревизия *rev* остается нетронутой, но головная ревизия (`HEAD`) удаляется.

*rev* Удаляет ревизию *rev*. Например, ‘`-o 1.3`’ эквивалентно ‘`-o 1.2::1.4`’.

*rev1*`:`*rev2* Удаляет ревизии от *rev1*, включительно, до *rev2*, включительно, находящиеся на одной ветке. После выполнения команды нельзя будет извлечь ревизии *rev1*, *rev2*, а также все ревизии в промежутке между ними. Например, команда ‘`cvs admin -oR_1_01:R_1_02 .`’ редко бывает полезна. Она означает "удалить ревизии вплоть до метки ‘`R_1_02`’, включительно". Осторожно! Если есть файл, которые изменились между ‘`R_1_02`’ и ‘`R_1_03`’, то в файле соответствующим меткам будут присвоены *одинаковые* номера ревизии. Из-за этого не только нельзя будет извлечь ‘`R_1_02`’, но и ‘`R_1_03`’ потребуется восстанавливать с резервных копий. В большинстве случаев следует использовать вариант ‘*rev1*`::`*rev2*’.

`:`*rev* Удалить ревизии с начала ветки, содержащей *rev*, вплоть до самой *rev*, включительно.

*rev*`:` Удалить ревизии от *rev*, включительно, до конца ветки, содержащей *rev*.

Ревизии не могут быть удалены, если они заблокированы или с них начинаются ветви.

Если ревизии имеют алфавитные имена и вы используете эти имена в сочетании с синтаксисом ‘`::`’, то CVS выдаст сообщение об ошибке и не будет удалять такие ревизии. Если вы действительно хотите удалить алфавитные имена и ревизии, то сначала удалите имена с помощью `cvs tag -d`, затем выполните `cvs admin -o`. Если вы не используете синтаксис ‘`::`’, то CVS удалит ревизии, но оставит алфавитные имена, которые ссылаются на несуществующие ревизии. Такое поведение оставлено для совместимости с предыдущими версиями CVS, но так как оно не очень полезно, то в будущем может измениться, чтобы совпадать со случаем ‘`::`’.

Из-за способа, которым CVS обрабатывает ветви, *rev* нельзя задавать в виде алфавитного имени, если эта ревизия находится на ветке. См. Раздел 5.5 [Волшебные номера веток], с. 44, где объясняется, почему это так.

Убедитесь, что никто не извлёк копию ревизии, которую вы делаете устаревшей. Могут произойти странные вещи, если кто-то редактирует эту ревизию и пытается зафиксировать её. Из-за этого ключ ‘`-o`’ не следует использовать для отмены ошибочного фиксирования, вместо этого фиксируйте ещё одну ревизию, исправляющую ошибочное изменение (см. Раздел 5.8 [Слияние двух ревизий], с. 46).

`-q` Работать тихо, не выдавать сопроводительных сообщений.

`-s`*state*`[:`*rev*`]`

Полезно использовать вместе с CVS. Устанавливает атрибут состояния ревизии *rev* в *state*. Если *rev* — это номер ветви, то использовать последнюю ревизию на этой ветви. Если *rev* опущена, использовать последнюю ревизию на ветви по умолчанию. В качестве *state* можно использовать любой

идентификатор. Полезным набором является ‘`Exp`’ (экспериментальный), ‘`Stab`’ (стабильный) и ‘`Rel`’ (вошедший в конечный продукт). По умолчанию состояние новой ревизии при создании устанавливается в ‘`Exp`’. Состояние сообщается командой ‘`cvs log`’ (см. Раздел A.13 [Команда log], с. 112) и в ключевых словах ‘`$Log$`’ и ‘`$State$`’ (см. Глава 12 [Подстановка ключевых слов], с. 77). Заметьте, что CVS использует состояние `dead` для своих внутренних нужд; для того, чтобы поместить файл в состояние `dead` или восстановить его из этого состояния, используйте команды `cvs remove` и `cvs add`, а не `cvs admin -s`.

`-t`[*файл*] Полезно при использовании вместе с CVS. Берёт из *файла* описание указанного RCS-файла, удаляя его старое описание. Имя *файла* не должно начинаться с минуса. Описание файла можно увидеть в выдаче команды ‘`cvs log`’ (см. Раздел A.13 [Команда log], с. 112). Между ‘`-t`’ и его аргументом не должно быть пробела.

Если *файл* опущен, то описание берётся со стандартного ввода, завершённое символом конца файла или строчкой, состоящей из единственного символа "точка". При работе с терминала текст описания запрашивается у пользователя, смотри ‘`-I`’. Чтение со стандартного ввода не работает в клиент-серверной CVS и может измениться в будущей версии CVS.

`-t-`*строка*

Похоже на ‘`-t`*файл*’. Текст описания берётся из *строки*, заменяя уже существующее описание. Между ‘`-t`’ и его аргументом не должно быть пробелов.

`-U` Устанавливает мягкий режим блокировок. Это означает, что владелец файла не обязан блокировать ревизию перед фиксированием. Для использования в CVS должен быть установлен жёсткий режим блокировок; смотри выше обсуждение ключа ‘`-l`’.

`-u`[*rev*] Смотри выше описание ключа ‘`-l`’, где обсуждается использование этого ключа в CVS. Разблокировать ревизию *rev*. Если дан номер ветки, разблокировать последнюю ревизию на этой ветке. Если *rev* опущена, удалить последнюю блокировку, сделанную пользователем в этом файле. Обычно только тот, кто сделал блокировку, может снять её. Если это делает кто-то другой, то это называется взломом блокировки. В этом случае владельцу блокировки отсылается уведомление по электронной почте. Почтовое сообщение может содержать комментарий, предоставленный тем, кто взломал блокировку. Комментарий завершается концом файла или строкой, состоящей из единственной точки. Между ключом ‘`-u`’ и его аргументом не должно быть пробела.

`-V`*n* В предыдущей версии CVS этот ключ означал, что RCS-файл нужно создавать в формате, понимаемой программой RCS версии *n*. В настоящий момент этот ключ устарел и его использование приводит к сообщению об ошибке.

`-x`*suffixes* В предыдущих версиях CVS этот ключ можно было использовать, чтобы задать имена RCS-файлов. Однако, CVS требует, чтобы имена RCS-файлов оканчивались на ‘`,v`’, поэтому этот ключ никогда не был полезен.

## A.7 Команда checkout: извлечение исходных текстов для редактирования

- Краткое описание: ‘`checkout [ключи]` *модули*...’
- Требует: репозиторий
- Изменяет: рабочий каталог
- Синонимы: `co`, `get`

Создаёт или обновляет рабочий каталог, содержащий копии файлов с исходными текстами, заданных с помощью параметра *модули*. Команду `checkout` обычно следует использовать перед использованием всех прочих команд CVS, так как большинство их них требует наличия рабочего каталога.

*модули* — это либо алфавитные имена коллекции каталогов и файлов с исходными текстами, или пути к каталогам и файлам в репозитории. Алфавитные имена описываются в файле ‘`modules`’. См. Раздел C.1 [Файл modules], с. 133.

В зависимости от модуля, который вы задали, команда `checkout` может рекурсивно создавать каталоги и заполнять их соответствующими файлами. Теперь вы можете редактировать эти файлы когда угодно, независимо от того, что кто-то ещё редактирует копии тех же самых файлов); затем обновите их, чтобы получить изменения, помещённые другими в репозиторий; зафиксируйте результаты вашей работы в репозиторий.

Заметьте, что `checkout` сам создаёт каталоги. В текущем каталоге при выполнении команды `checkout` образуется каталог верхнего уровня, чьё имя обычно совпадает с именем указанного модуля. В случае псевдонима модуля созданный подкаталог может иметь другое имя, но можно быть уверенным, что это будет именно подкаталог, и что `checkout` покажет относительный путь, ведущий к каждому файлу, который извлекается в ваш рабочий каталог (если вы не укажете глобальный ключ ‘`-Q`’).

Команда `checkout` создаёт файлы с правами на чтение и запись, если не задан глобальный ключ ‘`-r`’ (см. Раздел A.4 [Глобальные ключи], с. 90), не установлена переменная окружения `CVSREAD`, и за этим файлом не установлено слежение (см. Глава 13 [Слежение за исходными текстами], с. 81).

Заметьте, что допускается также выполнение `checkout` в каталоге, который был создан другой командой `checkout`. Это похоже на выполнение команды `update` с ключом ‘`-d`’, в том смысле, что в вашем рабочем каталоге появятся новые каталоги, которые были созданы в репозитории. Однако же, команда `checkout` требует имени модуля, тогда как команда `update` — имени каталога. Для использования `checkout` таким способом её нужно выполнять из каталога верхнего уровня, поэтому прежде чем использовать `checkout`, чтобы обновить существующий каталог, не забудьте перейти в каталог верхнего уровня.

Сообщения, которые выдаются командой `checkout`, описаны в Раздел A.16.2 [Сообщения команды update], с. 119.

### A.7.1 Ключи команды checkout

Команда `checkout` поддерживает стандартные ключи, описанные в (См. Раздел A.5 [Стандартные ключи], с. 92.):

`-D` *date* Использовать самую свежую ревизию, созданную не позднее *date*. Этот ключ является липким, и подразумевает использование ключа '`-P`'. См. Раздел 4.9 [Липкие метки], с. 38, где находится дополнительная информация о липких метках и датах.

`-f` Полезен только при использовании совместно с ключами '`-D` *date*' или '`-r` *tag*'. Если не найдено подходящей ревизии, извлечь самую свежую ревизию, а не игнорировать файл.

`-k` *kflag* Обрабатывать ключевые слова в соответствии с *kflag*. См. Глава 12 [Подстановка ключевых слов], с. 77. Этот ключ является липким: дальнейшие обновления этого рабочего каталога будут использовать тот же самый *kflag*. Для того, чтобы увидеть липкие ключи, используйте команду `status`. См. Приложение B [Вызов CVS], с. 121, где находится дополнительная информация о команде `status`.

`-l` Не обходить дерево каталогов, работать только в текущем рабочем каталоге.

`-n` Не выполнять программ при извлечении (тех, что указаны в файле '`modules`' с помощью ключа '`-o`'). см. Раздел C.1 [Файл modules], с. 133).

`-P` Удалять пустые каталоги. См. Раздел 7.5 [Перемещение каталогов], с. 55.

`-p` Выдавать содержимое файлов на стандартный вывод.

`-R` Извлекать каталоги рекурсивно. По умолчанию это именно так. См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *tag* Использовать ревизию *tag*. Этот ключ является липким и подразумевает использование '`-P`'. См. Раздел 4.9 [Липкие метки], с. 38, где можно найти дополнительную информацию о липких метках и датах.

Вдобавок к этому, можно использовать следующие ключи команды `checkout`:

`-A` Очистить все прилипшие метки, даты и ключи. См. Раздел 4.9 [Липкие метки], с. 38, а также Глава 12 [Подстановка ключевых слов], с. 77.

`-c` Копировать отсортированное содержимое файла '`modules`' на стандартный вывод, вместо того, чтобы создавать или изменять файлы или каталоги в рабочем каталоге.

`-d` *dir* Создать каталог с рабочими файлами, который называется *dir*, а не использовать имя модуля. Вообще использование этого флага эквивалентно использованию '`mkdir` *dir*`; cd` *dir*', за которым следует команда извлечения без ключа '`-d`'.

Однако же, существует важное исключение из этого правила. При извлечении одиночного файла очень удобно, чтобы файл создавались в каталоге, не содержащем пустых промежуточных каталогов. В этом, и только в этом случае CVS пытается "укоротить" пути к файлам, чтобы избежать подобных пустых каталогов.

Например, если имеется модуль '`foo`', содержащий файл '`bar.c`', то команда '`cvs co -d dir foo`' создаст каталог '`dir/`' и поместит внутрь файл

‘bar.c’. Аналогично, если есть модуль ‘bar’, в котором есть подкаталог ‘baz/’, в котором есть файл ‘quux.c’, то команда ‘cvs -d dir co bar/baz’ создаст каталог ‘dir/’ и поместит туда файл ‘quux.c’.

Использование флага ‘-N’ запретит такое поведение. В случае вышеописанной структуры модулей ‘cvs co -N -d dir foo’ создаст каталоги ‘dir/foo’ и поместит туда файл ‘bar.c’, а команда ‘cvs co -N -d dir bar/baz’ создаст каталоги ‘dir/bar/baz/’ и поместит туда файл ‘quux.c’.

`-j` *tag* Если используются два ключа ‘-j’, то изменения, сделанные в ревизии, указанной в первом ключе, будут объединены с ревизией, указанной во втором ключе, и помещены в рабочий каталог.

With one ‘-j’ option, merge changes from the ancestor revision to the revision specified with the ‘-j’ option, into the working directory. The ancestor revision is the common ancestor of the revision which the working directory is based on, and the revision specified in the ‘-j’ option.

Вдобавок каждый ключ ‘-j’ может задавать дату, которая, если используется вместе с ветвями, может ограничить выбор ревизий только подпадающими под эту дату. Дата задаётся с помощью двоеточия: ‘-j*Алфавитная_Метка*:*Указание_Даты*’.

См. Глава 5 [Создание ветвей и слияние], с. 41.

`-N` Полезен только с ‘-d *dir*’. При задании этого ключа CVS не будет “укорачивать” пути к модулям в вашем рабочем каталоге, когда вы извлекаете одиночный модуль. См. описание флага ‘-d’, где обсуждается этот вопрос и даются примеры.

`-s` Похоже на ‘-c’, но выдает также статус модулей и сортирует их в соответствии с этим статусом. См. Раздел C.1 [Файл modules], с. 133, за информацией о том, как ключ ‘-s’ используется для задания в файле ‘modules’ статуса модуля.

### A.7.2 Пример использования команды ‘checkout’

Получить копию модуля ‘tc’:

```
$ cvs checkout tc
```

Получить копию модуля ‘tc’ в том виде, в котором он находился вчера:

```
$ cvs checkout -D yesterday tc
```

## A.8 Команды commit: поместить файлы в репозиторий

- Краткое описание: `commit [-lnRf] [-m '`*журнальное_сообщение*`' | -F` *файл*`] [-r` *ревизия*`] [`*файлы*`...]`
- Требует: рабочий каталог, репозиторий.
- Изменяет: репозиторий.
- Синоним: ‘ci’

Используйте `commit`, если вы хотите поместить в репозиторий изменения, сделанные в вашей рабочей копии.

Если вы не укажете, какие файлы следует зафиксировать, то команда проверит все файлы в рабочем каталоге. `commit` тщательно следит за тем, чтобы помещать в репозиторий только те файл, которые действительно изменились. По умолчанию (или если явно задать ключ '`-R`') файлы в подкаталогах также обрабатываются и фиксируются, если они были изменены; можно использовать ключ '`-l`', чтобы ограничить команду `commit` только текущим каталогом.

`commit` проверяет, что указанные файлы свежее, чем текущие ревизии в репозитории; если это не так, то команда выдаст предупреждение о необходимости выполнить команду `update` (см. Раздел A.16 [Команда update], с. 117) и завершится, ничего более не делая. `commit` не станет выполнять за вас команду `update`, предоставляя вам сделать это в удобное время.

Если все нормально, будет вызван текстовый редактор, в котором можно будет ввести журнальное сообщение, которое будет передано программам журналирования (см. Раздел C.1 [Файл modules], с. 133 и см. Раздел C.7 [Файл loginfo], с. 141), а также помещено в RCS-файл в репозитории. Это журнальное сообщение можно извлечь с помощью команды `log`, см. Раздел A.13 [Команда log], с. 112. Можно задать журнальное сообщение в командной строке с помощью ключа '`-m` *журнальное_сообщение*', при этом редактор не будет вызван, или использовать ключ '`-F` *файл*', чтобы задать файл, содержащий журнальное сообщение.

### A.8.1 Ключи команды commit

Следующие стандартные ключи (см. Раздел A.5 [Стандартные ключи], с. 92, где можно найти их полное описание) поддерживаются командой `commit`:

`-l`

`-l` Не обходить дерево каталогов, работать только в текущем рабочем каталоге.

`-n` Не выполнять программ, относящихся к модулю.

`-R` Фиксировать каталоги рекурсивно. По умолчанию это именно так. См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *рев* Фиксировать с номером ревизии *рев*: это либо номер ветки, либо номер ревизии на основном стволе, больший, чем все существующие номера ревизии (см. Раздел 4.3 [Назначение номеров ревизий], с. 33). Нельзя помещать изменения в ревизию, находящуюся на ветке.

Команда `commit` также поддерживает следующие ключи:

`-F` *file* Прочесть журнальное сообщение из файла *файл*, не запускать редактор для ввода этого сообщения.

`-f` Заметьте, что поведение ключа '`-f`' в этой команде отличается от стандартного, которое описано в Раздел A.5 [Стандартные ключи], с. 92.

Заставляет CVS зафиксировать новую ревизию, даже если вы не сделали никаких изменений в файле. Если текущая ревизия *файла* имеет номер 1.7, то следующие две команды эквивалентны:

```
$ cvs commit -f file
$ cvs commit -r 1.8 file
```

Ключ ‘-f’ запрещает рекурсию (то есть подразумевает использование ‘-l’). Для того, чтобы заставить CVS зафиксировать новую ревизию для всех файлов во всех подкаталогах, используйте ‘-f -R’.

`-m` *сообщение*

Использовать *сообщение* в качестве журнального сообщения, не вызывать редактор.

### A.8.2 Пример использования команды commit

#### A.8.2.1 Помещение изменений на ветку

Вы можете зафиксировать изменения в ревизию, находящуюся на ветке (в её номере четное количество точек) с помощью ключа ‘-r’. Для того, чтобы создать ревизию на ветке, используйте ключ ‘-b’ команд `rtag` и `tag` (см. Глава 5 [Создание ветвей и слияние], с. 41). Затем используйте `checkout` или `update`, чтобы ваши исходные тексты стали основаны на этой свежесозданной ветке. Начиная с этого момента все изменения в этом рабочем каталоге автоматически добавляются в ревизию на ветке, никак не мешая разработке в основном стволе. Например, если вам потребовалось создать исправление к версии 1.2 вашего продукта, несмотря на то, что уже разрабатывается версия 2.0, вы можете:

```
$ cvs rtag -b -r FCS1_2 FCS1_2_Patch product_module
$ cvs checkout -r FCS1_2_Patch product_module
$ cd product_module
[[ hack away ]]
$ cvs commit
```

Все это будет работать автоматически, потому что ключ ‘-r’ является липким.

#### A.8.2.2 Создание ветки после редактирования

Предположим, вы работали над каким-то крайне экспериментальным продуктом, основанным на какой-то ревизии, извлеченной из репозитория неделю назад. Если кто-либо еще в вашей группе захочет вместе с вами работать над этим продуктом, не мешая при этом основному направлению разработки, то вы можете зафиксировать изменения в новую ветку. Другие смогут извлечь результаты вашего эксперимента и воспользоваться автоматическим исправлением конфликтов с помощью CVS. Сценарий таков:

```
[[ hacked sources are present ]]
$ cvs tag -b EXPR1
$ cvs update -r EXPR1
$ cvs commit
```

После команды `update` ключ ‘`-r EXPR1`’ прилипнет ко всем файлам. Заметьте, что ваши изменения в файлах никогда не будут удалены командой `update`. Команда `commit` автоматически поместит изменения на правильную ветку, потому что ключ ‘-r’ является липким. Вы также можете сделать так:

```
[[ hacked sources are present ]]
$ cvs tag -b EXPR1
$ cvs commit -r EXPR1
```

но в этом случае только те файлы, которые вы изменили, будут иметь прилепленный флаг '`-r EXPR1`'. Если вы поредактируете еще какие-либо файлы и зафиксируете их без указания флага '`-r EXPR1`', то эти файлы могут случайно оказаться в главном стволе.

Для того, чтобы работать вместе с вами над экспериментальной версией, другие могут просто сказать:

```
$ cvs checkout -r EXPR1 whatever_module
```

## A.9 Команда diff: показать различия между ревизиями

- Краткая сводка: `diff [-lR] [format_options] [[-r rev1 | -D date1] [-r rev2 | -D date2]] [files...]`
- Требует: рабочий каталог, репозиторий.
- Ничего не изменяет.

Команда `diff` используется для сравнения различных ревизий файлов. Действие по умолчанию – сравнить ваши рабочие копии файлов с ревизиями, на которых эти файлы основаны, и сообщить о всех обнаруженных различиях.

Если заданы какие-либо файлы, то сравниваются только они. Если заданы имена каталогов, то сравниваются файлы в этих каталогах.

Смысл кода завершения для команды `diff` отличается от всех прочих команд; детали описаны в Раздел A.2 [Код выхода], с. 89.

### A.9.1 Ключи команды diff

Команда `checkout` поддерживает стандартные ключи, описанные в См. Раздел A.5 [Стандартные ключи], с. 92.

`-D` *дата* Использовать самую свежую ревизию, созданную не позже чем *дата*. См. описание '`-r`', где описывается, как это влияет на результаты сравнения.

`-k` *kflag* Обрабатывать ключевые слова в соответствии с *kflag*. См. Глава 12 [Подстановка ключевых слов], с. 77.

`-l` Не обходить дерево каталогов, работать только в текущем рабочем каталоге.

`-R` Обрабатывать каталоги рекурсивно. По умолчанию это именно так. См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *tag* Сравнивать с ревизией *метка*. Можно задать от нуля до двух ключей '`-r`'. Без ключа '`-r`' рабочий файл будет сравниваться с ревизией, на которой он основан. С одним ключом '`-r`' указанная ревизия будет сравниваться с вашим рабочим файлом. С двумя ключами '`-r`' указанные две ревизии будут сравниваться друг с другом (а содержимое рабочих файлов не будет влиять на выдачу команды).

Один или оба ключа '`-r`' можно заменить на ключ '`-D` *дата*', описанный выше.

Нижеследующие ключи задают формат выдачи. Они имеют то же значение, что и в программе GNU diff.

`-0 -1 -2 -3 -4 -5 -6 -7 -8 -9`
`-binary`
`-brief`
`-changed-group-format=`*arg*
`-c`
`  -C` *nlines*
`  -context[=`*lines*`]`
`-e -ed`
`-t -expand-tabs`
`-f -forward-ed`
`-horizon-lines=`*arg*
`-ifdef=`*arg*
`-w -ignore-all-space`
`-B -ignore-blank-lines`
`-i -ignore-case`
`-I` *regexp*
`   -ignore-matching-lines=`*regexp*
`-h`
`-b -ignore-space-change`
`-T -initial-tab`
`-L` *label*
`  -label=`*label*
`-left-column`
`-d -minimal`
`-N -new-file`
`-new-line-format=`*arg*
`-old-line-format=`*arg*
`-paginate`
`-n -rcs`
`-s -report-identical-files`
`-p`
`-show-c-function`
`-y -side-by-side`
`-F` *regexp*
`-show-function-line=`*regexp*
`-H -speed-large-files`
`-suppress-common-lines`
`-a -text`
`-unchanged-group-format=`*arg*
`-u`
`  -U` *nlines*
`  -unified[=`*lines*`]`
`-V` *arg*
`-W` *columns*
`  -width=`*columns*

### A.9.2 Примеры использования команды diff

Нижеследующая строка выдает унифицированную (с ключом '`-u`') разницу между ревизиями 1.14 и 1.19 файла '`backend.c`'. Из-за наличия флага '`-kk`' ключевые слова не подставляются, поэтому различия внутри ключевых слов игнорируются.

```
$ cvs diff -kk -u -r 1.14 -r 1.19 backend.c
```

Предположим, что экспериментальная ветка '`EXPR1`' была основана на наборе файлов, помеченных как '`RELEASE_1_0`'. Чтобы увидеть, что произошло на этой ветке, используйте команду

```
$ cvs diff -r RELEASE_1_0 -r EXPR1
```

Такая команда может использоваться, чтобы показать контекстную разницу между двумя версиями программного продукта:

```
$ cvs diff -c -r RELEASE_1_0 -r RELEASE_1_1 > diffs
```

Если вы пользуетесь файлами '`ChangeLog`', то команда типа нижеуказанной поможет придумать подходящий текст для журнальной записи. Все изменения, которые вы сделали, будут продемонстрированы вам в удобном виде.

```
$ cvs diff -u | less
```

## A.10 Команда export: экспортировать исходные тексты

- Краткая сводка: export [-flNnR] [-r rev|-D дата] [-k subst] [-d dir] модуль...
- Требует: репозиторий.
- Изменяет: текущий каталог.

Эта команда похожа на команду `checkout`; её нужно использовать, если вы хотите получить копию исходных текстов модуля без служебных каталогов CVS. Например, команду `export` можно использовать, чтобы подготовить исходные тексты для передачи вовне. Эта команда требует указания даты или метки (с помощью ключей '`-D`' или '`-r`'), чтобы вы могли воспроизвести те же самые файлы, которые вы отдаете.

Часто при использовании `cvs export` приходится указывать флаг '`-kv`'. При этом ключевые слова будут развернуты так, чтобы при импорте в другом месте не потерялась информация о ревизиях. Помните, что в этом случае бинарные файлы не будут корректно обрабатываться. Также помните, что после использования флага '`-kv`' больше нельзя будет использовать команду `ident` (являющуюся частью пакета RCS), которая ищет строки с ключевыми словами (см. `ident(1)`). Если вы хотите использовать `ident`, то не используйте '`-kv`'.

### A.10.1 Ключи команды export

Команда `export` поддерживает стандартные ключи, описанные в См. Раздел A.5 [Стандартные ключи], с. 92.

`-D` *date* Использовать саму свежую ревизию не позже чем *date*.

`-f` Если не найдено совпадающей ревизии, извлечь самую свежую ревизию (а не игнорировать файл).

`-l` Не обходить дерево каталогов, работать только в текущем рабочем каталоге.

`-n` Не выполнять программ при извлечении.

`-R` каталоги рекурсивно. По умолчанию это именно так. См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *метка* Использовать ревизию *метка*.

Вдобавок поддерживаются следующие ключи (общие для `checkout` и `export`):

`-d` *dir* Создать для рабочих файлов каталог *dir*, а не использовать имя модуля. См. Раздел A.7.1 [Ключи команды checkout], с. 100, где полностью описаны детали обработки этого флага.

`-k` *subst* Установить режим подстановки ключевых слов (см. Раздел 12.4 [Режимы подстановки], с. 79).

`-N` Полезно только при использовании вместе с '`-d` *dir*'. См. Раздел A.7.1 [Ключи команды checkout], с. 100, где полностью описаны детали обработки этого флага.

## A.11 Команда history: показать состояние файлов и пользователей

- Краткая сводка: history [-report] [-flags] [-options args] [files. . .]
- Требует: файл '`$CVSROOT/CVSROOT/history`', иногда репозиторий.
- Ничего не изменяет.

CVS может вести файл истории, отслеживающий каждое использование команд `checkout`, `commit`, `rtag`, `update` и `release`. Для того, чтобы отображать эту информацию в разных форматах, используется команда `history`.

Журналирование должно быть включено путем создания файла '`$CVSROOT/CVSROOT/history`'.

**Предупреждение:** `history` использует ключи '`-f`', '`-l`', '`-n`' и '`-p`' не так, как они обычно используются с командами CVS (см. Раздел A.5 [Стандартные ключи], с. 92).

### A.11.1 Ключи команды history

Несколько флагов, показанных выше в качестве параметра '`-report`', задают вид генерируемого отчета:

`-c` Сообщить о всех командах `commit` (то есть о каждом изменении репозитория).

`-e` Сообщить обо всем (все виды записей). Эквивалентно заданию '`-x`' со всеми типами записей. Конечно, '`-e`' будет также включать все типы записей, которые будут добавлены в будущих выпусках CVS; если вы пишете скрипт, которые может обрабатывать только определенные типы записей, используйте '`-x`'.

`-m` *module*
: Сообщать об изменениях конкретного модуля (Можно использовать '`-m`' несколько раз в одной командной строке.)

`-o`
: Сообщать об извлеченных модулях.

`-T`
: Сообщать обо всех метках.

`-x` *type*
: Извлекать указанный набор типов записей *type* из истории CVS. Типы задаются одиночными буквами, которые можно использовать в комбинациях.

  Некоторые команды имеют единственный тип записи:

  `F` release

  `O` checkout

  `E` export

  `T` rtag

  После выполнения команды `update` могут появиться одна из четырех типов записей:

  `C` A merge was necessary but collisions were detected (requiring manual merging).

  `G` A merge was necessary and it succeeded.

  `U` A working file was copied from the repository.

  `W` The working copy of a file was deleted during update (because it was gone from the repository).

  После выполнения команды `commit` могут возникнуть одна из трех типов записей:

  `A` A file was added for the first time.

  `M` A file was modified.

  `R` A file was removed.

Ключи, показанные в виде параметра '`-flags`', задают дополнительные ограничения или, наоборот, добавляют дополнительную информацию к отчету, не требуя дополнительных параметров:

`-a`
: Показать данные обо всех пользователях (по умолчанию выдается только информация о пользователе, выполняющем команду `history`).

`-l`
: Показать только последнее изменение.

`-w`
: Показать только записи об изменениях, выполненных из того же рабочего каталога, где выполняется команда `history`.

Ключи, показанные в виде параметра '`-options` *args*', задают дополнительные ограничения, используя аргументы:

`-b` *str*
: Показывать данные от конца вплоть до последней строки *str*, встреченной в имени модуля, имени файла или пути к репозиторию.

`-D` *дата* Показывать данные с *даты*. Это немного отличается от обычного использования '`-D` *дата*', при котором извлекаются самые свежие ревизии, старше чем *дата*.

`-f` *file* Показывать данные для конкретного файла (можно использовать несколько ключей '`-f`' на одной командной строке). Это эквивалентно указанию файла в командной строке.

`-n` *module*
Показывать данные для конкретного модуля (можно использовать несколько ключей '`-n`' на одной командной строке).

`-p` *repository*
Показывать данные, относящиеся к конкретному репозиторию (вы можете задать в командной строке больше одного ключа '`-p`'.)

`-r` *rev* Показывать записи, относящиеся к ревизиям, появившимся после появления ревизии или метки *rev*. Соответствующие ревизии или метки ищутся в каждом RCS-файле.

`-t` *tag* Показывать записи, сделанные с тех пор, как в файле истории появилась запись о метке *tag*. Это отличается от ключа '`-r`' тем, что читается только файл истории, а не RCS-файлы, что гораздо быстрее.

`-u` *name* Показывать записи о пользователе *name*.

## A.12 Команда import: импортировать исходные тексты

- Краткая сводка: import [-options] repository vendortag releasetag. . .
- Требует: репозиторий, каталог с исходными текстами.
- Изменяет: репозиторий.

Используйте `import` для помещения в ваш репозиторий целого дерева каталогов, полученного из внешнего источника (например, поставщика исходных текстов). Эту команду можно использовать как для начального импорта, так и для глобального обновления модуля из внешнего источника. См. Глава 13 [Слежение за исходными текстами], с. 81, где обсуждается этот вопрос.

Параметр *repository* задает имя каталога (или путь к каталогу) в корневом каталоге CVS-репозитория; если этот каталог не существует, то CVS создаст его.

Когда вы импортируете с целью обновления дерева каталогов, которое было изменено в вашем репозитории с момента последнего импорта, вас уведомят обо всех файлах, конфликтующих в двух ветвях разработки; как советует `import`, используйте '`checkout -j`', чтобы согласовать изменения.

Если CVS решает, что файл нужно игнорировать (см. Раздел C.9 [Файл cvsignore], с. 143), то она не импортирует этот файл и печатает '`I` *имя-файла*' (см. Раздел A.12.2 [Сообщения команды import], с. 111, где полностью описаны сообщения).

Если существует файл '`$CVSROOT/CVSROOT/cvswrappers`', то файлы, чьи имена совпадают со спецификациями в этом файле, будут считаться packages и над ними перед импортом будут произведены указанные действия. См. Раздел C.2 [Обертки], с. 136.

Чужие исходные тексты хранятся на ветке первого уровня, по умолчанию имеющей номер 1.1.1. Обновления являются листьями на этой ветке; например, файлы из первой импортированной коллекции исходников будут иметь ревизию 1.1.1.1, файлы из первого импортированного обновления этой коллекции будут иметь ревизию 1.1.1.2 и т. д.

Команде требуется по крайней мере три параметра. *repository* требуется, чтобы задать коллекцию исходников. *vendortag* — это метка целой ветви (например, 1.1.1). Для того, чтобы идентифицировать файлы, находящиеся на листьях, образующихся при каждом импорте, требуется указать *releasetag*.

Заметьте, что `import` *не* изменяет каталог, в котором вы выполнили эту команду. В частности, этот каталог не становится рабочим каталогом CVS; если вы хотите работать с исходными текстами – сначала импортируйте их, а затем извлеките в другой каталог (см. Раздел 1.3.1 [Получение исходного кода], с. 4).

### A.12.1 Ключи команды import

Команда `import` поддерживает стандартный ключ, описанный в см. Раздел A.5 [Стандартные ключи], с. 92:

`-m` *сообщение*
: Использовать *сообщение* в качестве журнальной записи, а не вызывать редактор.

Есть также нижеследующие специальные ключи:

`-b` *branch*
: См. Раздел 13.6 [Несколько веток поставщика], с. 83.

`-k` *subst*
: Задает желаемый режим подстановки ключевых слов. Этот параметр будет влиять на все файлы, созданные во время импорта, но не на те, что уже существовали в репозитории. См. Раздел 12.4 [Режимы подстановки], с. 79, где описываются разрешенные параметры ключа '`-k`'.

`-I` *name*
: Задает имена файлов, которые следует игнорировать при импорте. Этот ключ можно использовать несколько раз. Чтобы не игнорировать ни один файл, задайте '`-I !`'.

  *name* может быть шаблоном имен файлов, типа тех, что можно задать в файле '`.cvsignore`'. См. Раздел C.9 [Файл cvsignore], с. 143.

`-W` *spec*
: Задает имена файлов, которые следует профильтровать перед импортом. Этот ключ можно использовать несколько раз.

  *spec* может быть шаблоном имен файлов, типа тех, что можно задать в файле '`.cvswrappers`'. См. Раздел C.2 [Обертки], с. 136.

### A.12.2 Сообщения команды output

Команда `import` сообщает вам о своей деятельности, печатая строку на каждый файл, в начале которой находится один символ, сообщающий о статусе файла:

`U` *file*
: Файл уже существует в репозитории и не был локально изменен; будет создана новая ревизия, если нужно.

`N` *file* Это новый файл и теперь он добавлен в репозиторий.

`C` *file* Файл уже существует в репозитории, но был изменен локально, поэтому вам потребуется объединить изменения.

`I` *file* Файл игнорируется (см. Раздел C.9 [Файл cvsignore], с. 143).

`L` *file* Файл является символической ссылкой; `cvs import` игнорирует символические ссылки. Периодически предлагается изменить это поведение, но нет четкого соглашения, как именно. (Различные ключ в файле '`modules`' могут использоваться для воссоздания символических ссылок при извлечении, обновлении и т. п.; см. Раздел C.1 [Файл modules], с. 133).

### A.12.3 Примеры использования команды import

См. Глава 13 [Слежение за исходными текстами], с. 81, а также Раздел 3.1.1 [Из файлов], с. 29.

## A.13 Команда log: напечатать информацию о файлах

- Краткая сводка: log [options] [files. . .]
- Требует: репозиторий, рабочий каталог.
- Ничего не изменяет.

Отображает журнальную информацию о файлах. `log` используется, чтобы вызвать утилиту `rlog` из комплекта RCS. Although this is no longer true in the current sources, this history determines the format of the output and the options, which are not quite in the style of the other CVS commands.

Команда сообщает о местонахождении RCS-файла, головной ревизии (`HEAD`) (последней ревизии на стволе, обо всех алфавитных именах меток, а также некоторую другую информацию. Для каждой ревизии печатается её номер, автор, количество удаленных и добавленных строк и журнальное сообщение. Все метки времени отображаются по Гринвичу (в UTC). (Другие части CVS печатают время в местной временно́й зоне).

**Предупреждение:** `log` использует ключ '`-R`' не так, как это обычно делается в CVS (см. Раздел A.5 [Стандартные ключи], с. 92).

### A.13.1 Ключи команды log

По умолчанию команда `log` сообщает всю доступную информацию. Ключи предназначены для ограничения выдачи.

`-b` Печатает информацию о ревизиях на ветке по умолчанию, обычно это самая верхняя ветка на стволе.

`-d` *dates* Печатает информацию о ревизиях, помещенных в репозиторий в промежутке времени, заданном списком дат через точку с запятой. Форматы дат те же, что используются вместе с ключом '`-D`' (см. Раздел A.5 [Стандартные ключи], с. 92). Даты можно комбинировать следующим образом:

*d1*<*d2*
*d2*>*d1* Выбрать ревизии, которые были помещены в репозиторий между датой *d1* и датой *d2*.

<*d*
*d*> Выбрать все ревизии, датированные *d* или раньше.

*d*<
>*d* Выбрать все ревизии, датированные *d* или позже.

*d* Выбрать последнюю ревизию, датированную *d* или раньше.

Вместо символов '>' и '<' можно использовать '<=' и '>=', чтобы задать диапазон включительно, а не исключительно.

Между ключом '-d' и его аргументом не должно быть пробела. Заметьте также, что разделителем является точка с запятой (';').

-h Печатает только имя RCS-файла, имя файла в рабочем каталоге, головную ревизию, ревизию по умолчанию, список прав доступа, блокировки, алфавитные имена и суффикс.

-l Не обходить дерево каталогов, работать только в текущем рабочем каталоге.

-N Не выдает список меток для этого файла. Этот ключ полезен, если вы используете много меток, чтобы избежать просматривания трех экранов информации о метках.

-R Печатает только имя RCS-файла.

-r*revisions*

Печатает информацию о ревизиях, перечисленных в списке ревизий и их диапазонов, разделенных запятыми. В этой таблице приведены все доступные форматы диапазонов:

*rev1*:*rev2* Ревизии от *rev1* до *rev2* (должны находиться на одной ветке).

:*rev* Ревизии с начала ветки вплоть до *rev* включительно.

*rev*: Ревизии, начиная с *rev* до конца ветки, содержащей *rev*.

*branch* Аргумент, являющийся веткой означает все ревизии на этой ветке.

*branch1*:*branch2*
Диапазон ветвей означает все ревизии на ветках в этом диапазоне.

*branch*. Последнюю ревизию на ветке *branch*.

Пустой ключ '-r', без списка ревизий, означает использование последней ревизии на ветке по умолчанию, обычно на стволе. Между ключом '-r' и его параметром не должно быть пробела.

-s *states* Печатает информацию о ревизиях, чей атрибут состояния совпадает с состоянием из разделенного запятыми списка *states*.

`-t` Печатает то же самое, что и '`-h`', плюс текст описания.

`-w`*logins* Печатает информацию о ревизиях, созданных пользователями, перечисленными через запятую в списке *logins*. Если список опущен, используется имя текущего пользователя. Между ключом '`-w`' и его аргументом не должно быть пробела.

`log` печатает информацию о ревизиях удовлетворяющих ключам '`-d`', '`-s`', '`-w`' и совокупности ключей '`-b`' и '`-r`'.

### A.13.2 Примеры использования команды log

Примеры будут с благодарностью приняты.

## A.14 Команда rdiff: выдать изменения между версиями в формате patch

- Краткая сводка: rdiff [-flags] [-V vn] [-r t|-D d [-r t2|-D d2]] modules...
- Требует: репозиторий.
- Ничего не изменяет.
- Синоним: `patch`.

Создает файл изменений между двумя версиями продукта в формате программы `patch(1)`, написанной Ларри Воллом. Этот файл можно скормить программе `patch`, чтобы обновить старую версию до новой. (Это одна из немногих команд CVS, которые работают напрямую с репозиторием и не требуют предварительного извлечения исходных текстов.) Результат выдается на стандартный вывод.

Вы можете задать (используя стандартные ключи '`-r`' и '`-D`') любую комбинацию двух ревизий или дат. Если указана только одна ревизия или дата, то результат содержит изменения в промежутке между этой ревизией или датой и текущей головной ревизией в RCS-файле.

Заметьте, что если соответствующая версия продукта находится в нескольких каталогах, то может потребоваться указать команде `patch` при обновлении старых исходников ключ '`-p`', чтобы `patch` смогла найти файлы, находящиеся в других каталогах.

### A.14.1 Ключи команды rdiff

Команда `rdiff` поддерживает стандартные ключи, описанные в см. Раздел A.5 [Стандартные ключи], с. 92:

`-D` *date* Использовать саму свежую ревизию, сделанную не позже *date*.

`-f` Если не найдено совпадающей ревизии, извлечь самую свежую ревизию (а не игнорировать файл).

`-l` Не обходить дерево каталогов, работать только в текущем рабочем каталоге.

`-R` каталоги рекурсивно. По умолчанию это именно так. См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *tag* Использовать ревизию *tag*.

Вдобавок доступны следующие ключи:

`-c` Использовать контекстный формат. По умолчанию это именно так.

`-s` Создать вместо файла с исправлениями краткий отчет об изменениях. Отчет содержит информацию о файлах, которые были изменены или добавлены в промежутке между версиями. Этот отчет выдается на стандартный вывод. Это полезно, например, для выяснения, какие именно файлы изменились между двумя датами или ревизиями.

`-t` Изменения между двумя последними ревизиями выдаются на стандартный вывод. Это особенно полезно для выяснения, в чем заключалось последнее изменение файла.

`-u` Использовать унифицированный формат файла изменений. Учтите, что старые версии программы `patch` не могли обрабатывать этот формат, поэтому если вы планируете опубликовать изменения в сети, то вам, скорее всего, не следует использовать ключ '`-u`'.

`-V` *vn* Раскрывать ключевые слова в соответствии с правилами, принятыми в RCS версии *vn* (формат подстановки изменился в RCS версии 5). Заметьте, что этот ключ больше не обрабатывается. CVS всегда будет раскрывать ключевые слова так, как этот делает RCS версии 5.

### A.14.2 Примеры использования команды rdiff

Предположим, вы получаете письмо от `foo@example.net`, который просит вас прислать обновление с версии 1.2 до версии 1.4 компилятора `tc`. У вас нету под рукой такого обновления, но с помощью CVS вы можете легко сделать так:

```
$ cvs rdiff -c -r FOO1_2 -r FOO1_4 tc | \
$$ Mail -s 'Исправления, которые Вы запрашивали' foo@example.net
```

Предположим, что вы сделали версию 1.3 и ветку '`R_1_3fix`' для исправлений этой версии. '`R_1_3_1`' соответствует версии 1.3.1, которая была сделана некоторое время назад. Теперь вы хотите узнать, что именно было сделано на этой ветке. Можно использовать такую команду:

```
$ cvs patch -s -r R_1_3_1 -r R_1_3fix module-name
cvs rdiff: Diffing module-name
File ChangeLog,v changed from revision 1.52.2.5 to 1.52.2.6
File foo.c,v changed from revision 1.52.2.3 to 1.52.2.4
File bar.h,v changed from revision 1.29.2.1 to 1.2
```

## A.15 Команда release: сообщить, что модуль более не используется

- Краткая сводка: release [-d] directories...
- Требует: рабочий каталог.
- Изменяет: рабочий каталог, журнал истории.

Эту команду можно использовать, чтобы безопасным образом отменить ‘`cvs checkout`’. Так как CVS не блокирует файлы, то использовать эту команду необязательно. Вы всегда можете просто удалить рабочий каталог, если желаете; правда, в этом случае вы рискуете потерять изменения, о которых забыли, а в файле истории (см. Раздел C.10 [Файл history], с. 144) не остается никаких следов того, что вы отбросили извлеченные исходники.

Команда ‘`cvs release`’ позволяет избежать этой проблемы. Она проверяет, что в рабочем каталоге нет незафиксированных изменений; что вы выполняете эту команду из каталога, в котором находится рабочий каталог; что репозиторий, из которого был извлечен рабочий каталог, совпадает с репозиторием, описанным в базе данных модулей.

Если все эти условия выполняются, ‘`cvs release`’ оставляет запись о своем выполнении в журнал истории (удостоверяя, что вы сознательно отложили извлеченные тексты).

### A.15.1 Ключи команды release

Команда `release` поддерживает единственный ключ:

`-d` Удалить рабочую копию файлов, если все нормально. Если этот флаг не указан, то ваши файлы останутся в вашем рабочем каталоге.

**Предупреждение:** Команда `release` рекурсивно удаляет все каталоги и файлы. Это имеет очень важный побочный эффект: все каталоги, которые вы создали в извлеченном дереве исходников, но не добавили в репозиторий (используя команду `add`; см. Раздел 7.1 [Добавление файлов], с. 51) будут бесшумно удалены — даже если эти каталоги непусты!

### A.15.2 Сообщения команды release

Перед тем, как `release` освободит ваши исходные тексты, эта команда печатает однострочное сообщение для каждого файла, который не соответствует содержимому репозитория.

**Предупреждение:** Все каталоги, которые вы создали, но не добавили в репозиторий с помощью команды `add` (см. Раздел 7.1 [Добавление файлов], с. 51), будут бесшумно проигнорированы (и удалены, если был указан флаг ‘`-d`’), даже если эти каталоги содержат файлы.

`U` *file*
`P` *file* В репозитории существует новая ревизия этого файла, а вы изменили его рабочую копию (‘`U`’ и ‘`P`’ означают одно и то же).

`A` *file* Файл был добавлен в ваш рабочий каталог, но еще не был помещен в репозиторий. Если вы удаляете вашу копию исходных текстов, то этот файл будет потерян.

`R` *file* Файл был удален из вашего рабочего каталога, но еще не был удален из репозитория, потому что вы еще не зафиксировали удаление. См. Раздел A.8 [Команда commit], с. 102.

`M` *file* Файл изменен в рабочем каталоге. В репозитории также может быть более новая ревизия.

`?` *file* *file* находится в рабочем каталоге, но не соответствует ни одному файлу в репозитории, и не находится в списке файлов, которые нужно игнорировать (см. описание ключа '`-I`' и см. Раздел C.9 [Файл cvsignore], с. 143). Если вы удалите рабочий каталог, изменения будут потеряны.

### A.15.3 Примеры использования команды release

Освободить каталог '`tc`' и удалить рабочие копии файлов:

```
$ cd ..             # Вам нужно находиться в каталоге, содержащем
                    # ваш каталог с исходными текстами, перед тем,
                    # как вы выполните команду 'cvs release'.
$ cvs release -d tc
You have [0] altered files in this repository.
Are you sure you want to release (and delete) directory 'tc': y
$
```

## A.16 Команда update: обновить рабочий каталог из репозитория

- Краткая сводка: update [-AdflPpR] [-d] [-r tag|-D date] files...
- Требует: репозиторий, рабочий каталог.
- Изменяет: рабочий каталог.

После того, как вы извлечете из общего репозитория рабочую копию исходных текстов, другие разработчики продолжат вносить изменения в этот репозиторий. Время от времени, в удобное для вас время можно использовать команду `update` в вашем рабочем каталоге, чтобы увязать вашу работу с ревизиями, помещенными в репозиторий после того, как вы извлекли или последний раз обновляли ваш рабочий каталог.

### A.16.1 Ключи команды update

Команда `update` поддерживает стандартные ключи, которые полностью описаны в см. Раздел A.5 [Стандартные ключи], с. 92:

`-D date` Использовать самую свежую ревизию, созданную не позже *date*. Этот ключ является липким, и подразумевает использование ключа '`-P`'. См. Раздел 4.9 [Липкие метки], с. 38, где полностью описаны липкие метки и даты.

`-f` Полезно только при использовании вместе с ключами '`-D` *дата*' или '`-r` *метка*'. Если не найдено совпадающей ревизии, извлечь самую свежую ревизию, а не игнорировать файл.

`-k` *kflag* Обрабатывать ключевые слова в соответствии с *kflag*. См. Глава 12 [Подстановка ключевых слов], с. 77. Этот ключ является липким; дальнейшие обновления этого файла в рабочем каталоге будут использовать тот же самый *kflag*. Команду `status` можно использовать для просмотра липких

ключей. См. Приложение B [Вызов CVS], с. 121, где описана команда `status`.

`-l` Не обходить дерево каталогов, работать только в текущем рабочем каталоге.

`-P` Удалять пустые каталоги. См. Раздел 7.5 [Перемещение каталогов], с. 55.

`-p` Писать файлы в стандартный вывод.

`-R` Обрабатывать каталоги рекурсивно. По умолчанию это именно так. См. Глава 6 [Рекурсивное поведение], с. 49.

`-r rev` Извлечь ревизию/метку *rev*. Этот ключ является липким и подразумевает использование '`-P`'. См. Раздел 4.9 [Липкие метки], с. 38, где полностью описаны липкие метки и даты.

Команду `update` можно также использовать с такими ключами:

`-A` Очистить все прилипшие метки, даты и ключи. См. Раздел 4.9 [Липкие метки], с. 38, а также Глава 12 [Подстановка ключевых слов], с. 77.

`-d` Создавать каталоги, существующие в репозитории, если они отсутствуют в рабочем каталоге. Обычно `update` работает только с файлами и каталогами, которые уже были созданы в рабочем каталоге.

Этот ключ полезен при обновлении каталогов, которые были созданы в репозитории уже после извлечения вашей рабочей копии, но у него есть неприятный побочный эффект: если вы специально избегали определенных каталогов в репозитории, когда создавали рабочий каталог (используя имена модулей или явно перечисляя в командной строке требуемые файлы и каталоги), то обновление с ключом '`-d`' создаст эти нежелаемые каталоги.

`-I` *name* Во время обновления игнорировать файлы в вашем рабочем каталоге, чьи имена совпадают с *name*. Можно использовать этот ключ несколько раз, чтобы задать несколько файлов, которые нужно игнорировать. Используйте '`-I !`', чтобы не игнорировать ни один файл. См. Раздел C.9 [Файл cvsignore], с. 143, где описаны другие способы игнорирования файлов.

`-W`*spec* Задает имена файлов, которые следует фильтровать во время обновления. Этот ключ можно использовать несколько раз.

*spec* — это шаблон имен файлов типа тех, что можно использовать в файле '`.cvswrappers`'. См. Раздел C.2 [Обертки], с. 136.

`-j`*revision* При использовании двух ключей '`-j`' в рабочем каталоге происходит объединение изменений между ревизией, заданной первым ключом, и ревизией, заданной вторым ключом.

При использовании одного ключа '`-j`' в рабочем каталоге происходит слияние изменений между ревизией-предком и ревизией, заданной ключом '`-j`'. Ревизия-предок — это общий предок ревизии, на основе которой создан рабочий каталог, и ревизии, заданной ключом '`-j`'.

Вдобавок, каждый ключ '`-j`' может содержать необязательное указание даты, которая, при использовании вместе с ветвями, может дополнительно

ограничить выбор ревизий. Необязательная дата задается добавлением двоеточия (‘:’) к метке: ‘-j*Symbolic_Tag*:*Date_Specifier*’.

См. Глава 5 [Создание ветвей и слияние], с. 41.

### A.16.2 Сообщения команды update

Команды `update` и `checkout` информируют о своей деятельности, печатая строчку на каждый обработанный файл. Первый символ означает статус этого файла:

`U` *file*
: Файл был обновлен из репозитория. Обновление производится: для файлов, существующих в репозитории, но не в вашем рабочем каталоге; для файлов, которые вы не изменяли, и для которых в репозитории существует более свежая ревизия.

`P` *file*
: Похоже на ‘`U`’, но для этого файла CVS-сервер посылает файл с исправлениями вместо целого файла. Результирующий файл оказывается тем же самым.

`A` *file*
: Файл был добавлен в вашем рабочем каталоге, и будет помещен в репозиторий, когда вы выполните команду `commit`. Это сообщение — напоминание о том, что файл требуется зафиксировать.

`R` *file*
: Файл был удален из вашей личной копии исходников, и будет удален из репозитория, когда вы выполните команду `commit`. Это сообщение — напоминание о том, что файл требуется зафиксировать.

`M` *file*
: Файл был изменен в вашем рабочем каталоге.

  ‘`M`’ может означать одно из двух состояний файла, над которым вы работаете: либо этот файл не менялся в репозитории, и поэтому остался неизменным в результате выполнения команды `update`, либо же файл изменился как в рабочем каталоге, так и в репозитории, но слияние изменений в ваш рабочий файл прошло успешно, без конфликтов.

  CVS выдает некоторые сообщения, когда сливает изменения, и оставляет резервную копию рабочего файла (как он выглядел перед выполнением `update`). Точное имя этого файла печатается во время работы `update`.

`C` *file*
: При попытке объединить изменения из репозитория в вашу рабочую копию файла был обнаружен конфликт. *file* (ваша рабочая копия) теперь является результатом попытки объединить две ревизии; неизмененная копия файла также находится в рабочем каталоге, с именем ‘`.#`*file*`.`*revision*’, где *revision* — это ревизия, на которой был основан измененный вами файл. Разрешение конфликтов описано в Раздел 10.3 [Пример конфликта], с. 65. (Заметьте, что некоторые системы автоматически удаляют файлы, начинающиеся с ‘`.#`’, если к этим файлам не было обращений в течение нескольких дней. Если вы хотите сохранить копию исходного файла, переименуйте его.) Под VMS имя файла начинается с ‘`__`’, а не с ‘`.#`’.

`?` *file*
: *file* находится в вашем рабочем каталоге, но не соответствует ни одному файлу в репозитории, и не находится в списке файлов, которые нужно игнорировать (см. описание ключа ‘`-I`’ и см. Раздел C.9 [Файл cvsignore], с. 143).

# Приложение B Краткий справочник по командам CVS

В этом приложении описано, как вызывать CVS, со ссылками на места в руководстве, где детально описана каждая возможность. Дополнительную информацию можно получить, выполнив `cvs --help` или изучив [Индекс], с. 163.

Команда CVS выглядит так:

`cvs [` *global_options* `] command [` *command_options* `] [` *command_args* `]`

Глобальные ключи:

`-allow-root=`*rootdir*
: Разрешает использование каталога CVSROOT (только для сервера) (не реализовано в CVS 1.9 и ранее). См. Раздел 2.9.3.1 [Сервер парольной аутентификации], с. 21.

`-a`
: Аутентифицировать все взаимодействие (только для клиента) (не реализовано в CVS 1.9 и ранее). См. Раздел A.4 [Глобальные ключи], с. 90.

`-b`
: Задает местонахождение программ RCS (CVS 1.9 и ранее). См. Раздел A.4 [Глобальные ключи], с. 90.

`-d` *root*
: Задает CVSROOT. См. Глава 2 [Репозиторий], с. 7.

`-e` *редактор*
: Редактировать сообщение с помощью *редактора*. См. Раздел 1.3.2 [Фиксирование изменений], с. 4.

`-f`
: Не читать файл '`~/.cvsrc`'. См. Раздел A.4 [Глобальные ключи], с. 90.

`-H`
`-help`
: Выдает справочное сообщение. См. Раздел A.4 [Глобальные ключи], с. 90.

`-l`
: Не журналировать команду в файле '`CVSROOT/history`'. См. Раздел A.4 [Глобальные ключи], с. 90.

`-n`
: Не изменять файлы на диске. См. Раздел A.4 [Глобальные ключи], с. 90.

`-Q`
: Совсем не выдавать сообщений. См. Раздел A.4 [Глобальные ключи], с. 90.

`-q`
: Почти совсем не выдавать сообщений. См. Раздел A.4 [Глобальные ключи], с. 90.

`-r`
: Создавать новые рабочие файлы в режиме "только для чтения". См. Раздел A.4 [Глобальные ключи], с. 90.

`-s` *variable*=*value*
: Установить пользовательскую переменную. См. Раздел C.11 [Переменные], с. 144.

`-T` *tempdir*
: Создавать временные файлы в каталоге *tempdir*. См. Раздел A.4 [Глобальные ключи], с. 90.

`-t`
: Отслеживать ход выполнения CVS. См. Раздел A.4 [Глобальные ключи], с. 90.

`-v`

`-version` Напечатать информацию об версии программы CVS и авторских правах.

`-w` Создавать новые рабочие файлы в режиме для чтения и записи. См. Раздел A.4 [Глобальные ключи], с. 90.

`-x` Шифровать все переговоры с сервером (только для клиента). См. Раздел A.4 [Глобальные ключи], с. 90.

`-z` *gzip-level*
Установить коэффициент сжатия (только для клиента).

Режимы подстановки ключевых слов (см. Раздел 12.4 [Режимы подстановки], с. 79):

```
-kkv  $Id: file1,v 1.1 1993/12/09 03:21:13 joe Exp $
-kkvl $Id: file1,v 1.1 1993/12/09 03:21:13 joe Exp harry $
-kk   $Id$
-kv   file1,v 1.1 1993/12/09 03:21:13 joe Exp
-ko   не подставлять
-kb   не подставлять, файл является двоичным
```

Ключевые слова (см. Раздел 12.1 [Список ключевых слов], с. 77):

```
$Author: joe $
$Date: 1993/12/09 03:21:13 $
$Header: /home/files/file1,v 1.1 1993/12/09 03:21:13 joe Exp harry $
$Id: file1,v 1.1 1993/12/09 03:21:13 joe Exp harry $
$Locker: harry $
$Name: snapshot_1_14 $
$RCSfile: file1,v $
$Revision: 1.1 $
$Source: /home/files/file1,v $
$State: Exp $
$Log: file1,v $
Revision 1.1  1993/12/09 03:30:17  joe
Initial revision
```

Команды, их ключи и параметры:

`add` [*options*] [*files*...]
Добавить новый файл или каталог. См. Раздел 7.1 [Добавление файлов], с. 51.

`-k` *kflag* Задать режим подстановки ключевых слов. См. Глава 12 [Подстановка ключевых слов], с. 77.

`-m` *msg* Задать описание файла.

`admin` [*options*] [*files*...]
Административные функции файлов истории версий в репозитории. См. Раздел A.6 [Команда admin], с. 95.

`-b`[*rev*] Установить ветку по умолчанию. См. Раздел 13.3 [Отмена локальных изменений], с. 82.

`-c`*string* Задать префикс комментария.

`-k` *kflag* Задать режим подстановки ключевых слов. См. Глава 12 [Подстановка ключевых слов], с. 77.

`-l`[*rev*] Блокировать ревизию *rev* или последнюю ревизию.

`-m`*rev*`:`*msg*
Заменить журнальную запись ревизии *rev* сообщением *msg*.

`-o`*range* Удалить ревизии из репозитория. См. Раздел A.6.1 [Ключи команды admin], с. 95.

`-q` Выполнять команды, не выдавая сообщений.

`-s`*state*`[:`*rev*`]`
Установить состояние ревизии.

`-t` Получить описание файла со стандартного ввода.

`-t`*file* Получить описание файла из *file*.

`-t-`*string* Установить описание файла в *string*.

`-u`[*rev*] Разблокировать ревизию *rev* или последнюю ревизию.

`annotate` [*options*] [*files*`...`]
Для каждой строки показать последнюю ревизию, когда она была изменена. См. Раздел 8.4 [annotate], с. 58.

`-D` *date* Аннотировать самую последнюю ревизию, созданную не позже *date*. См. Раздел A.5 [Стандартные ключи], с. 92.

`-f` Использовать головную ревизию (`HEAD`), если не найдено указанной метки или даты. См. Раздел A.5 [Стандартные ключи], с. 92.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *tag* Аннотировать ревизию *tag*. См. Раздел A.5 [Стандартные ключи], с. 92.

`checkout` [*options*] *modules*`...`
Извлечь копию исходных текстов. См. Раздел A.7 [Команда checkout], с. 100.

`-A` Очистить все прилипшие метки, даты и ключи. См. Раздел 4.9 [Липкие метки], с. 38, а также Глава 12 [Подстановка ключевых слов], с. 77.

`-c` Напечатать содержимое базы данных модулей. См. Раздел A.7.1 [Ключи команды checkout], с. 100.

`-D` *date* Извлечь ревизии, датированные *date* (это липкий ключ). См. Раздел A.5 [Стандартные ключи], с. 92.

`-d` *dir* Извлекать в каталог *dir*. См. Раздел A.7.1 [Ключи команды checkout], с. 100.

`-f` Использовать головную ревизию (`HEAD`), если не найдено указанной метки или даты. См. Раздел A.5 [Стандартные ключи], с. 92.

`-j` *rev* Слить изменения. См. Раздел A.7.1 [Ключи команды checkout], с. 100.

`-k` *kflag* Использовать режим подстановки ключевых слов *kflag*. См. Раздел 12.4 [Режимы подстановки], с. 79.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-N` Не обрезать пути к модулям, если задан ключ '`-d`'. См. Раздел A.7.1 [Ключи команды checkout], с. 100.

`-n` Не выполнять никаких программ. См. Раздел A.7.1 [Ключи команды checkout], с. 100.

`-P` Удалять пустые каталоги. См. Раздел 7.5 [Перемещение каталогов], с. 55.

`-p` Извлекая файлы, печатать их в стандартный вывод (избегая липкости). См. Раздел A.7.1 [Ключи команды checkout], с. 100.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *tag* Извлечь ревизию *tag* (ключ липкий). См. Раздел A.5 [Стандартные ключи], с. 92.

`-s` Похоже на '`-c`', но выдает также статус модуля. См. Раздел A.7.1 [Ключи команды checkout], с. 100.

`commit [`*options*`] [`*files*`...]`

Помещает изменения в репозиторий. См. Раздел A.8 [Команда commit], с. 102.

`-F` *файл* Читает журнальное сообщение из *файла*. См. Раздел A.8.1 [Ключи команды commit], с. 103.

`-f` Принудительно фиксирует файл; запрещает рекурсию. См. Раздел A.8.1 [Ключи команды commit], с. 103.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-m` *msg* Использовать *msg* в качестве журнального сообщения. См. Раздел A.8.1 [Ключи команды commit], с. 103.

`-n` Не выполнять программ. См. Раздел A.8.1 [Ключи команды commit], с. 103.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *rev* Фиксировать в ревизию *rev*. См. Раздел A.8.1 [Ключи команды commit], с. 103.

`diff` [*options*] [*files*...]
Показывает изменения между ревизиями. См. Раздел A.9 [Команда diff], с. 105. Вдобавок к нижеуказанным поддерживает множество ключей, управляющих форматом выдачи, например, '`-c`' для создания контекстных файлов изменений.

`-D` *date1* Выдать изменения от ревизии, датированной *date1*, до рабочего файла. См. Раздел A.9.1 [Ключи команды diff], с. 105.

`-D` *date2* Выдать изменения от *rev1* или *date1* до *date2*. См. Раздел A.9.1 [Ключи команды diff], с. 105.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-N` Включает изменения для добавленных и удаленных файлов. См. Раздел A.9.1 [Ключи команды diff], с. 105.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *rev1* Выдать изменения от *rev1* до рабочего файла. См. Раздел A.9.1 [Ключи команды diff], с. 105.

`-r` *rev2* Выдать изменения от *rev1* или *date1* до *rev2*. См. Раздел A.9.1 [Ключи команды diff], с. 105.

`edit` [*options*] [*files*...]
Приготовиться к редактированию файла, за которым ведется наблюдение. См. Раздел 10.6.3 [Редактирование файлов], с. 71.

`-a` *actions* Задает действия, за которыми нужно следить. *actions* может быть `edit`, `unedit`, `commit`, `all` или `none`. См. Раздел 10.6.3 [Редактирование файлов], с. 71.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`editors` [*options*] [*files*...]
Посмотреть, кто редактирует файл, за которым ведется наблюдение. Раздел 10.6.4 [Информация о слежении], с. 72.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`export` [*options*] *modules*...
Экспортировать файлы из CVS. См. Раздел A.10 [Команда export], с. 107.

`-D` *date* Извлечь ревизии, датированные *date*. См. Раздел A.5 [Стандартные ключи], с. 92.

`-d` *dir* Извлекать в каталог *dir*. См. Раздел A.10.1 [Ключи команды export], с. 107.

`-f` Использовать головную ревизию (`HEAD`), если не найдено указанной метки или даты. См. Раздел A.5 [Стандартные ключи], с. 92.

`-k` *kflag* Задать режим подстановки ключевых слов. См. Глава 12 [Подстановка ключевых слов], с. 77.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-N` Не обрезать пути к модулям, если задан ключ '`-d`'. См. Раздел A.7.1 [Ключи команды checkout], с. 100.

`-n` Не выполнять программ перед извлечением. См. Раздел A.10.1 [Ключи команды export], с. 107.

`-P` Удалять пустые каталоги. См. Раздел 7.5 [Перемещение каталогов], с. 55.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *tag* Извлечь ревизию *tag* (липкий ключ). См. Раздел A.5 [Стандартные ключи], с. 92.

`history` [*options*] [*files*...]

Показать историю обращений к репозиторию. См. Раздел A.11 [Команда history], с. 108.

`-a` Показать информацию обо всех пользователях (по умолчанию — только о себе). См. Раздел A.11.1 [Ключи команды history], с. 108.

`-b` *str* Показывать до записи с вхождением строки *str* в имя модуля, файла или репозитория. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-c` Сообщать о зафиксированных (измененных) файлах. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-D` *date* Сообщать о событиях, начиная с *date*. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-e` Сообщать о всех типах записей. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-l` Last modified (committed or modified report). См. Раздел A.11.1 [Ключи команды history], с. 108.

`-m` *module*

Сообщать о модуле *module* (ключ можно задать несколько раз). См. Раздел A.11.1 [Ключи команды history], с. 108.

`-n` *module*
: Сообщать об изменениях в модуле *module*. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-o`
: Сообщать об извлеченных модулях. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-r` *rev*
: Сообщать об изменениях, начиная с ревизии *rev*. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-T`
: Produce report on all TAGs. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-t` *tag*
: Сообщать об изменениях, сделанных с момента, когда была создана метка *tag*. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-u` *user*
: Сообщать об изменениях, сделанных пользователем *user* (ключ можно задать несколько раз). См. Раздел A.11.1 [Ключи команды history], с. 108.

`-w`
: Рабочие каталоги должны совпадать. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-x` *types*
: Сообщать о типах событий *types*, обозначаемых буквами `TOEFWUCGMAR`. См. Раздел A.11.1 [Ключи команды history], с. 108.

`-z` *zone*
: Использовать временну́ю зону *zone*. См. Раздел A.11.1 [Ключи команды history], с. 108.

`import` [*options*] *repository vendor-tag release-tags*...
: Импортировать файлы в CVS, используя ветки поставщика. См. Раздел A.12 [Команда import], с. 110.

  `-b` *bra*
  : Импортировать на ветку поставщика *bra*. См. Раздел 13.6 [Несколько веток поставщика], с. 83.

  `-d`
  : Использовать время модификации файла в качестве времени импорта. См. Раздел A.12.1 [Ключи команды import], с. 111.

  `-k` *kflag*
  : Задать режим подстановки ключевых слов. См. Глава 12 [Подстановка ключевых слов], с. 77.

  `-k` *kflag*
  : Задать режим подстановки ключевых слов, действующий по умолчанию. См. Раздел A.12.1 [Ключи команды import], с. 111.

  `-m` *msg*
  : Использовать *msg* в качестве журнального сообщения. См. Раздел A.12.1 [Ключи команды import], с. 111.

  `-I` *ign*
  : Список файлов, которые нужно игнорировать (‘`!`’ очищает этот список). См. Раздел A.12.1 [Ключи команды import], с. 111.

  `-W` *spec*
  : Дополнительные обертки. См. Раздел A.12.1 [Ключи команды import], с. 111.

`init` Создать репозиторий CVS, если он еще не существует. См. Раздел 2.6 [Создание репозитория], с. 17.

`log` [*options*] [*files*...]
Напечатать информацию об истории файлов. См. Раздел A.13 [Команда log], с. 112.

- `-b` Выдавать информацию только о ревизиях на ветви по умолчанию. См. Раздел A.13.1 [Ключи команды log], с. 112.
- `-d` *dates* Задает даты (*d1*<*d2* означает диапазон, *d* — не позже). См. Раздел A.13.1 [Ключи команды log], с. 112.
- `-h` Печатать только заголовок. См. Раздел A.13.1 [Ключи команды log], с. 112.
- `-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.
- `-N` Не выдавать имена меток. См. Раздел A.13.1 [Ключи команды log], с. 112.
- `-R` Печатать только имя RCS-файла. См. Раздел A.13.1 [Ключи команды log], с. 112.
- `-r`*revs* Печатать только информацию о ревизиях *revs*. См. Раздел A.13.1 [Ключи команды log], с. 112.
- `-s` *states* Печатать только информацию о ревизиях, находящихся в указанных состояниях. См. Раздел A.13.1 [Ключи команды log], с. 112.
- `-t` Печатать только заголовок и текст описания. См. Раздел A.13.1 [Ключи команды log], с. 112.
- `-w`*logins* Только о ревизиях, созданных указанными пользователями. См. Раздел A.13.1 [Ключи команды log], с. 112.

`login` Ввести пароль для аутентификации на сервере. См. Раздел 2.9.3.2 [Парольная аутентификация клиента], с. 23.

`logout` Удалить сохраненный пароль на сервер. См. Раздел 2.9.3.2 [Парольная аутентификация клиента], с. 23.

`rdiff` [*options*] *modules*...
Показать различия между версиями. См. Раздел A.14 [Команда rdiff], с. 114.

- `-c` Контекстный формат выдачи изменений (по умолчанию). См. Раздел A.14.1 [Ключи команды rdiff], с. 114.
- `-D` *date* Выбрать ревизии, созданные в *date*. См. Раздел A.5 [Стандартные ключи], с. 92.
- `-f` Использовать головную ревизию (`HEAD`), если не найдено указанной метки или даты. См. Раздел A.5 [Стандартные ключи], с. 92.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *rev* Выбрать ревизии *rev*. См. Раздел A.5 [Стандартные ключи], с. 92.

`-s` Короткая заплатка — одна строка на файл. См. Раздел A.14.1 [Ключи команды rdiff], с. 114.

`-t` Последнее изменение, сделанное в файле. См. Раздел A.9.1 [Ключи команды diff], с. 105.

`-u` Унифицированный формат выдачи изменений. См. Раздел A.14.1 [Ключи команды rdiff], с. 114.

`-V` *vers* Использовать RCS версии *vers* для подстановки ключевых слов (устарело). См. Раздел A.14.1 [Ключи команды rdiff], с. 114.

`release` [*options*] *directory*

Указывает, что каталог больше не используется. См. Раздел A.15 [Команда release], с. 115.

`-d` Удалить указанный каталог. См. Раздел A.15.1 [Ключи команды release], с. 116.

`remove` [*options*] [*files*...]

Удаляет файл из репозитория. См. Раздел 7.2 [Удаление файлов], с. 52.

`-f` Удалить файл в рабочем каталоге перед удалением из репозитория. См. Раздел 7.2 [Удаление файлов], с. 52.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`rtag` [*options*] *tag modules*...

Пометить набор ревизий в модуле. См. Глава 4 [Ревизии], с. 33, а также Глава 5 [Создание ветвей и слияние], с. 41.

`-a` Убрать метку с удаленных файлов, которые в противном случае не были бы помечены. См. Раздел 4.8 [Пометка добавлений и удалений], с. 38.

`-b` *tag* Создать ветку *tag*. См. Глава 5 [Создание ветвей и слияние], с. 41.

`-D` *date* Пометить ревизии, датированные *date*. См. Раздел 4.6 [Пометка по дате или метке], с. 37.

`-d` *tag* Удалить метку *tag*. См. Раздел 4.7 [Изменение меток], с. 37.

`-F` Переместить метку *tag*, если она уже существует. См. Раздел 4.7 [Изменение меток], с. 37.

`-f` Использовать головную ревизию (`HEAD`), если не найдена метка или дата. См. Раздел 4.6 [Пометка по дате или метке], с. 37.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-n` Не выполнять программ при создании меток. См. Раздел A.5 [Стандартные ключи], с. 92.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *rev* Пометить существующую метку *rev*. См. Раздел 4.6 [Пометка по дате или метке], с. 37.

`status` [*options*] *files*...
Напечатать информацию о статусе файлов в рабочем каталоге. См. Раздел 10.1 [Статус файла], с. 63.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-v` Сообщить также информацию о метках в файле. См. Раздел 4.4 [Метки], с. 34.

`tag` [*options*] *tag* [*files*...]
Пометить извлеченные версии файлов. См. Глава 4 [Ревизии], с. 33, а также Глава 5 [Создание ветвей и слияние], с. 41.

`-b` Создать ветку *tag*. См. Глава 5 [Создание ветвей и слияние], с. 41.

`-c` Проверить, что рабочие файлы не изменялись. См. Раздел 4.5 [Пометка рабочего каталога], с. 36.

`-D` *date* Пометить ревизии, датированные *date*. См. Раздел 4.6 [Пометка по дате или метке], с. 37.

`-d` Удалить метку *tag*. См. Раздел 4.7 [Изменение меток], с. 37.

`-F` Переместить метку *tag*, если она уже существует. См. Раздел 4.7 [Изменение меток], с. 37.

`-f` Использовать головную ревизию (`HEAD`), если не найдена метка или дата. См. Раздел 4.6 [Пометка по дате или метке], с. 37.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`-r` *rev* Пометить существующую метку *rev*. См. Раздел 4.6 [Пометка по дате или метке], с. 37.

`unedit` [*options*] [*files*...]
: Отменить команду ‘`edit`’. См. Раздел 10.6.3 [Редактирование файлов], с. 71.

  `-a` *actions* Задает действия, за которыми нужно следить. *actions* может быть `edit`, `unedit`, `commit`, `all` или `none`. См. Раздел 10.6.3 [Редактирование файлов], с. 71.

  `-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

  `-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`update` [*options*] [*files*...]
: Обновить рабочее дерево каталогов из репозитория. См. Раздел A.16 [Команда update], с. 117.

  `-A` Очистить все прилипшие метки, даты и ключи. См. Раздел 4.9 [Липкие метки], с. 38, а также Глава 12 [Подстановка ключевых слов], с. 77.

  `-D` *date* Извлекать ревизии, датированные *date* (ключ является липким). См. Раздел A.5 [Стандартные ключи], с. 92.

  `-d` Создавать каталоги. См. Раздел A.16.1 [Ключи команды update], с. 117.

  `-f` Использовать головную ревизию (`HEAD`), если не найдено указанной метки или даты. См. Раздел A.5 [Стандартные ключи], с. 92.

  `-I` *ign* Добавить файлы в список игнорируемых (‘`!`’ очищает этот список). См. Раздел A.12.1 [Ключи команды import], с. 111.

  `-j` *rev* Объединить изменения. См. Раздел A.16.1 [Ключи команды update], с. 117.

  `-k` *kflag* Использовать режим подстановки ключевых слов *kflag*. См. Раздел 12.4 [Режимы подстановки], с. 79.

  `-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

  `-P` Удалять пустые каталоги. См. Раздел 7.5 [Перемещение каталогов], с. 55.

  `-p` Извлекать файлы на стандартный вывод (избежав липкости). См. Раздел A.16.1 [Ключи команды update], с. 117.

  `-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

  `-r` *tag* Извлечь ревизию *tag* (ключ липкий). См. Раздел A.5 [Стандартные ключи], с. 92.

`-W` *spec* Добавить обертку. См. Раздел A.12.1 [Ключи команды import], с. 111.

`watch [on|off|add|remove]` [*options*] [*files*...]

on/off: включить/выключить извлечение файлов только для чтения. См. Раздел 10.6.1 [Включение слежения], с. 69.

add/remove: добавить или удалить уведомление о производимых действиях. См. Раздел 10.6.2 [Получение уведомлений], с. 69.

`-a` *actions* Задает действия, за которыми нужно следить. *actions* может быть `edit`, `unedit`, `commit`, `all` или `none`. См. Раздел 10.6.3 [Редактирование файлов], с. 71.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

`watchers` [*options*] [*files*...]

Вывести список следящих за файлом. См. Раздел 10.6.4 [Информация о слежении], с. 72.

`-l` Работать только в текущем каталоге. См. Глава 6 [Рекурсивное поведение], с. 49.

`-R` Рекурсивно обходить дерево каталогов (по умолчанию это именно так). См. Глава 6 [Рекурсивное поведение], с. 49.

# Приложение C Справочник по административным файлам

В репозитории CVS в каталоге '`$CVSROOT/CVSROOT/`' находится несколько вспомогательных файлов. При работе с CVS эти файлы можно и не использовать, но, будучи правильно настроенными, они способны сильно облегчить вам жизнь. Методика редактирования таких файлов обсуждается в Приложение C [Административные файлы], с. 133.

Самым важным таким файлом является '`modules`', которые описывает модули, находящиеся в репозитории.

## C.1 Файл '`modules`'

В файле '`modules`' находится описание модулей, то есть коллекций исходных текстов. Файл модулей можно редактировать обычным для административных файлов способом.

Файл '`modules`' может содержать пустые строки и комментарии (строки, начинающиеся с '`#`'), а также описания модулей. Длинные описания можно разбивать на несколько строк, используя обратную косую черту ('\') в качестве последнего символа в строке.

Существует три основных типа модулей: модули-синонимы, обычные модули и амперсенд-модули. Разница между ними заключается в способе сопоставления файлов в репозитории файлам в рабочем каталоге. В нижеприведенных примерах в репозитории находится каталог '`first-dir/`', содержащий два файла, '`file1`' и '`file2`', а также каталог '`sdir/`'. '`first-dir/sdir/`' содержит также файл '`sfile`'.

### C.1.1 Модули-синонимы

Модули синонимы — это самый простой вид модулей:

*mname* `-a` *aliases*`...`

> Это — простейший путь описания модуля *mname*. Флаг '`-a`' означает, что новый модуль будет лишь синонимом для указанного списка модулей: CVS будет обращаться с *mname*, указанным в командной строке, точно так же, как если бы вместо него был указан список *aliases*. *aliases* может содержать имена других модулей или имена каталогов. При использовании имен каталогов CVS создает промежуточные каталоги в рабочем каталоге, как если бы путь был задан явно в командной строке.

Например, если файл '`modules`' содержит строку

```
amodule -a first-dir
```

то следующие две команды эквивалентны:

```
$ cvs co amodule
$ cvs co first-dir
```

и обе выдадут одинаковые сообщения:

```
cvs checkout: Updating first-dir
U first-dir/file1
```

```
U first-dir/file2
cvs checkout: Updating first-dir/sdir
U first-dir/sdir/sfile
```

### C.1.2 Обычные модули

*mname* `[ options ]` *dir* `[` *files*`... ]`

> В простейшем случае эта форма описания модуля сокращается до '*mname dir*'. В этом случае файлы в каталоге *dir* становятся модулем *mname*. *dir* — путь к каталогу с исходными текстами, относительно `$CVSROOT`. При извлечении исходных текстов в рабочем каталоге создается каталог *mname*; по умолчанию не используются промежуточные каталоги, даже если *dir* состоит из нескольких уровней каталогов.

Например, если модуль описан как

```
regmodule first-dir
```

то *regmodule* будет содержать файлы из '`first-dir/`':

```
$ cvs co regmodule
cvs checkout: Updating regmodule
U regmodule/file1
U regmodule/file2
cvs checkout: Updating regmodule/sdir
U regmodule/sdir/sfile
$
```

Явно указывая в описании модуля после имени каталога имена файлов, можно извлекать их из каталога по отдельности. Вот пример:

```
regfiles first-dir/sdir sfile
```

При таком описании извлечение модуля *regfiles* создает единственный рабочий каталог '`regfiles`', содержащий указанный файл, который берется из каталога '`first-dir/sdir/`', находящегося в репозитории:

```
$ cvs co regfiles
U regfiles/sfile
$
```

### C.1.3 Амперсенд-модули

Описание модуля может ссылаться на другие модули, используя запись '&*module*'.

*mname* `[ options ]` *&module*`...`

При извлечении такого модуля для каждого амперсенд-модуля создается соответствующий подкаталог. Например, если файл '`modules`' содержит строчку

```
ampermod &first-dir
```

то при извлечении будет создан каталог '`ampermod/`', содержащий каталог, который называется '`first-dir/`', который, в свою очередь, содержит все каталоги и файлы, находящиеся в этом каталоге. Например, команда

```
$ cvs co ampermod
```

создаст следующие файлы:

```
ampermod/first-dir/file1
ampermod/first-dir/file2
ampermod/first-dir/sdir/sfile
```

В реализации CVS есть одна ошибка: сообщения, которые выдает CVS, не содержат упоминания ‘ampermod/’, и поэтому неправильно сообщают о местонахождении извлеченных файлов:

```
$ cvs co ampermod
cvs checkout: Updating first-dir
U first-dir/file1
U first-dir/file2
cvs checkout: Updating first-dir/sdir
U first-dir/sdir/sfile
$
```

Не полагайтесь на такое ошибочное поведение; в будущих версиях CVS оно может быть исправлено.

### C.1.4 Исключение каталогов из списка

Модуль-синоним может исключить определенные каталоги из модулей, используя восклицательный знак (‘!’) перед именем каждого исключенного каталога.

Например, если файл ‘modules’ содержит

```
exmodule -a !first-dir/sdir first-dir
```

то при извлечении модуля ‘exmodule’ будут извлечено все содержимое ‘first-dir/’, кроме файлов из каталога ‘first-dir/sdir’.

### C.1.5 Флаги модулей

Описание обычных и амперсенд-модулей может содержать флаги, предоставляющие дополнительную информацию о модуле.

`-d` *name* Дать рабочему каталогу другое имя, отличающееся от имени модуля.

`-e` *prog* Задает программу *prog*, которая выполняется при экспорте модуля. *prog* выполняется с единственным аргументом, именем модуля.

`-i` *prog* Задает программу *prog*, которая выполняется, когда изменения в файлах модуля помещаются в репозиторий. *prog* выполняется с полным именем соответствующего каталога (??? а не файла?) в репозитории. Файлы ‘commitinfo’, ‘loginfo’ и ‘verifymsg’ обеспечивают дополнительные способы выполнения программ при фиксировании.

`-o` *prog* Задает программу *prog*, которая выполняется, когда файлы модуля извлекаются в рабочий каталог. *prog* выполняется с единственным аргументом, именем модуля.

`-s` *status* Задает статус модуля. Когда файл модулей выдается на экран с помощью ‘cvs checkout -s’, модули в нем отсортированы по статусу, а затем по имени модуля. Этот ключ не имеет какого-либо другого значения. Этот ключ можно использовать для нескольких вещей, помимо статуса модуля: например, перечислить людей, ответственных за модуль.

`-t` *prog* Задает программу *prog*, которая выполняется, когда файлы в модуле помечаются с помощью команды `rtag`. *prog* выполняется с двумя аргументами: именем модуля и именем метки, указанной в команде `rtag`. Эта программа *не* выполняется, когда дается команда `tag`. Обычно лучше использовать файл '`taginfo`' (см. Раздел 8.3 [Настройка журналирования], с. 57).

`-u` *prog* Задает программу, которая выполняется, когда команда '`cvs update`', выполняется в основном каталоге извлеченного модуля. *prog* выполняется с единственным аргументом, полным путем к указанному модулю в репозитории.

## C.2 Файл 'cvswrappers'

*Обертки* — это возможность CVS, позволяющая управлять определенными настройками, основываясь на имени обрабатываемого файла. В список таких настроек входят ключи '`-k`' для двоичных файлов и '`-m`' для файлов, которые нельзя автоматически объединять.

Ключ '`-m`' задает метод объединения, который нужно использовать при обновлении не-двоичного файла. '`MERGE`' означает обычное поведение CVS: попробовать объединить файлы. '`COPY`' означает, что `cvs update` откажется объединять файлы, точно так же, как это происходит с двоичными файлами, описанными с помощью ключа '`-kb`' (если файл описан как двоичный, то использовать '`-m 'COPY''`' необязательно). CVS предоставит пользователю две версии файлов, и потребует вручную внести необходимые изменения, пользуясь внешними по отношению к CVS инструментами. **Предупреждение:** не используйте '`COPY`' с CVS версии 1.9 и раньше — они просто перезапишут один файл поверх другого, уничтожая старое содержимое. Ключ '`-m`' влияет только на поведение при обновлении, не затрагивая способ хранения файла. См. Глава 9 [Двоичные файлы], с. 59, где описана работа с ними.

В основном формат файла '`cvswrappers`' таков:

```
маска_файла      [ключ значение][ключ значение]...
```

где ключ — это

`-m` способ обновления ('`MERGE`' или '`COPY`')

`-k` способ подстановки ключевых слов. Подробности в См. Глава 12 [Подстановка ключевых слов], с. 77.

а значение заключено в одиночные кавычки.

Например, нижеследующая команда импортирует каталог, считая файлы, заканчивающиеся на '`.exe`', двоичными:

```
cvs import -I ! -W "*.exe -k 'b'" first-dir vendortag reltag
```

## C.3 Выполнение программ на разных стадиях фиксирования

Флаг '`-i`' в файле '`modules`' может использоваться для выполнения определенной программы, когда соответствующие файлы помещаются в репозиторий (см. Раздел C.1 [Файл modules], с. 133). Файлы, описанные в этой секции, обеспечивают другие, более гибкие способы выполнения программ при фиксировании.

Есть три вида программ, которые можно выполнять при фиксировании. Они указываются в файлах в репозитории, как описано ниже. В этой таблице находится сводка таких файлов и назначение соответствующих программ:

‘`commitinfo`’
: Программа, ответственная за проверку, разрешена ли команда фиксирования изменений. Если эта программа заканчивается с ненулевым кодом завершения, фиксирование будет прервано.

‘`verifymsg`’
: Указанная программа используется для проверки журнального сообщения, чтобы убедиться, что оно содержит все требуемые поля. Полезно использовать этот файл в комбинации с ‘`rcsinfo`’, который может содержать шаблон журнального сообщения (см. Раздел C.8 [Файл rcsinfo], с. 142).

‘`editinfo`’
: Заданная программа используется для редактирования журнального сообщения, и, возможно, проверки, что оно содержит все требуемые поля. Это особенно полезно в комбинации с файлом ‘`rcsinfo`’, который может содержать шаблон журнального сообщения (см. Раздел C.8 [Файл rcsinfo], с. 142). Устарело.

‘`loginfo`’ Указанная программа вызывается, когда завершено фиксирование. Она получает журнальное сообщение и дополнительную информацию, и может сохранить сообщение в файле, отправить его по почте ответственному человеку, поместить в местной группе новостей, или... Пределом является только ваше воображение!

### C.3.1 Обычный синтаксис

Административные файлы, такие как ‘`commitinfo`’, ‘`loginfo`’, ‘`rcsinfo`’, ‘`verifymsg`’, и т. д., все имеют общий формат. Назначение этих файлов описано позднее, а здесь описан их общий синтаксис.

Каждая строка содержит следующее:

- Регулярное выражение: простое регулярное выражение, использующее синтаксис GNU emacs.
- Разделитель – один или больше пробелов или символов табуляции.
- Имя файла или шаблон командной строки.

Пустые строки игнорируются. Строки, которые начинаются с символа ‘`#`’, считаются комментариями. Длинные строки, к сожалению, *не* могут быть разбиты на части.

Используется первое регулярное выражение, которое совпадает с именем текущего каталога в репозитории. Остаток строки используется, соответственно, как имя файла или командная строка.

## C.4 Файл ‘`commitinfo`’

Файл ‘`commitinfo`’ описывает программы, которые выполняются перед тем, как команда ‘`cvs commit`’ выполняет свою работу. Эти программы используются перед

фиксированием изменений для проверки, чтобы измененный, добавленные и удаленные файлы действительно готовы к фиксированию. Это можно использовать, например, чтобы убедиться, что измененные файлы соответствуют стандартам кодирования, принятым в вашей организации.

Как упоминалось ранее, каждая строка в файле '`commitinfo`' состоит из регулярного выражения и шаблона командной строки. Шаблон может включать имя программы и аргументы, которые вы хотите передать этой программе. К шаблону добавляется полный путь к текущему репозиторию, за которым следуют имена файлов, участвующих в фиксировании (добавленные, удаленные и измененные).

Используется первая строка с регулярным выражением, соответствующим каталогу в репозитории. Если команда возвращает ненулевой код выхода, то фиксирование будет прервано.

Если имя репозитория не соответствует ни одному регулярному выражению в этом файле, то используется строка '`DEFAULT`', если она есть.

Помимо совпадающих строк, используются также все строки, начинающиеся с '`ALL`'.

Замечание: когда CVS обращается к сетевому репозиторию, '`commitinfo`' будет выполняться на сервере, а не на клиенте (см. Раздел 2.9 [Сетевые репозитории], с. 19).

## C.5 Проверка журнальных записей

Когда вы ввели журнальное сообщение, вы можете проверить его, чтобы убедиться, что в нем представлена вся необходимая информация, такая, как номера исправленных ошибок и т. п.

Файл '`verifymsg`' полезнее всего использовать вместе с файлом '`rcsinfo`', который используется в качестве шаблона журнального сообщения.

Каждая строка в файле '`verifymsg`' состоит из регулярного сообщения и шаблона команды. В шаблоне должно присутствовать имя программы и, возможно, несколько аргументов. К шаблону добавляется полный имя файла с шаблоном журнального сообщения.

Следует заметить, что ключевое слово '`ALL`' не поддерживается. Если найдено более одной совпадающей строки, используется первая. Это полезно для указания скрипта проверки, используемого по умолчанию, а затем переопределения его в подкаталоге.

Если имя репозитория не совпадает ни с одним регулярным выражением в этом файле, то используется строка '`DEFAULT`', если она есть.

Если проверочный скрипт завершается с ненулевым кодом завершения, то процесс фиксирования завершается.

Заметьте, что скрипт верификации не может изменять журнальное сообщение, но лишь принимать или отвергать его.

Вот простой пример файла '`verifymsg`', использующегося вместе с соответствующим шаблоном журнальной записи в файле '`rcsinfo`' и скриптом проверки этой записи. Сначала — шаблон журнальной записи. Нам нужно, чтобы в первой строке этой записи находился номер исправленной ошибки. Остаток журнальной записи — в свободной форме. Вот такой шаблон находится в файле '`/usr/cvssupport/tc.template`':

```
BugId:
```

Скрипт ‘/usr/cvssupport/bugid.verify’ используется для проверки журнального сообщения.

```
#!/bin/sh
#
#       bugid.verify filename
#
#  Verify that the log message contains a valid bugid
#  on the first line.
#
if head -1 < $1 | grep '^BugId:[ ]*[0-9][0-9]*$' > /dev/null; then
    exit 0
else
    echo "No BugId found."
    exit 1
fi
```

Файл ‘verifymsg’ содержит строку:

```
^tc     /usr/cvssupport/bugid.edit
```

Файл ‘rcsinfo’ содержит такую строку:

```
^tc     /usr/cvssupport/tc.template
```

## C.6 Файл ‘editinfo’

*ЗАМЕЧАНИЕ:* использование ‘editinfo’ устарело. Для задания редактора журнальных записей по умолчанию используйте переменную окружения EDITOR (см. Приложение D [Переменные окружения], с. 147) или глобальный ключ ‘-e’ (см. Раздел A.4 [Глобальные ключи], с. 90) См. Раздел C.5 [Файл verifymsg], с. 138, где описано, как использовать ‘verifymsg’.

Если вы хотите убедиться, что все журнальные сообщения выглядят одинаково, то можете использовать файл ‘editinfo’, чтобы задать программу, используемую для редактирования этих сообщений. Этой программой может быть текстовый редактор, настроенный специальным образом, или небольшой скрипт, который вызывает редактор и проверяет, что введенное сообщение содержит все требуемые поля.

Если в файле ‘editinfo’ не найдено совпадающей строки, используется редактор, заданный переменной окружения $CVSEDITOR. Если эта переменная не установлена, используется переменная окружения $EDITOR. Если и эта переменная не установлена, используется редактор по умолчанию. См. также Раздел 1.3.2 [Фиксирование изменений], с. 4.

Файл ‘editinfo’ наиболее полезно использовать вместе с файлом ‘rcsinfo’, который используется в качестве шаблона журнального сообщения.

Каждая строка в файле ‘editinfo’ состоит из регулярного выражения и шаблона команды, состоящего из имени программы и, возможно, нескольких аргументов. К шаблону программы добавляется полное имя текущего шаблона журнального сообщения.

Следует заметить, что ключевое слово ‘ALL’ не поддерживается. Если совпадает более одной строки, используется первая. Это полезно для задания скрипта редактирования по умолчанию, а затем переопределения его в подкаталоге.

Если имя каталога в репозитории не совпадает ни с одним регулярным выражением в этом файле, то используется строка ‘DEFAULT’, если она есть.

Если скрипт редактирования завершается с ненулевым кодом завершения, то процесс фиксирования аварийно завершается.

Заметьте, что когда CVS обращается к сетевому репозиторию, или когда используются ключи ‘-m’ и ‘-F’ команды cvs commit, то файл ‘editinfo’ не используется. Вместо него можно использовать ‘verifymsg’.

### C.6.1 Пример использования Editinfo

Ниже следует небольшой глупый пример файла ‘editinfo’, вместе с соответствующим шаблоном журнального сообщения в файле ‘rcsinfo’ и скрипт редактора. Мы начнем с шаблона журнального сообщения. Предположим, мы хотим хранить номер исправленной ошибки в первой строке журнального сообщения. Остаток журнального сообщения содержит любой описательный текст. В файле ‘/usr/cvssupport/tc.template’ находится такой шаблон:

```
BugId:
```

Скрипт ‘/usr/cvssupport/bugid.edit’ используется для редактирования журнального сообщения.

```
#!/bin/sh
#
#       bugid.edit filename
#
#  Call $EDITOR on FILENAME, and verify that the
#  resulting file contains a valid bugid on the first
#  line.
if [ "x$EDITOR" = "x" ]; then EDITOR=vi; fi
if [ "x$CVSEDITOR" = "x" ]; then CVSEDITOR=$EDITOR; fi
$CVSEDITOR $1
until head -1|grep '^BugId:[ ]*[0-9][0-9]*$' < $1
do  echo -n  "No BugId found.  Edit again? ([y]/n)"
    read ans
    case ${ans} in
        n*) exit 1;;
    esac
    $CVSEDITOR $1
done
```

Файл ‘editinfo’ содержит такую строчку:

```
^tc     /usr/cvssupport/bugid.edit
```

Файл ‘rcsinfo’ содержит такую строчку:

```
^tc     /usr/cvssupport/tc.template
```

## C.7 Файл loginfo

Файл ‘`loginfo`’ используется для управления тем, куда посылается журнальная информация при выполнении ‘`cvs commit`’. В левой части строки находится регулярное выражение, с которым совпадает имя каталога, в котором производится изменение, относительно `$CVSROOT`. Остаток соответствующей строки – это программа-фильтр, которая получает журнальное сообщение на стандартный ввод.

Если имя в репозитории не совпадает ни с одним регулярным выражением, используется строка ‘`DEFAULT`’, если она есть.

Все строки, начинающиеся с ‘`ALL`’, используются вдобавок к обычным строкам с совпадающим регулярным выражением, и со строкой ‘`DEFAULT`’.

Используется первое совпадающее регулярное выражение.

См. Раздел C.3 [Фиксирование и программы], с. 136, где описан синтаксис файла ‘`loginfo`’.

Пользователь может использовать в имени команды форматные строки. Такие строки состоят из символа ‘`%`’, за которым следует пробел, одиночный форматный символ или набор форматных символов, заключенных в скобки ‘`{`’ и ‘`}`’. Форматные символы таковы:

`s` имя файла

`V` старый номер ревизии (перед фиксированием)

`v` новый номер ревизии (после фиксирования)

Все прочие символы, появляющиеся в форматной строке, превращаются в пустые строки (запятые, разделяющие их, сохраняются).

Например, можно использовать форматные строки ‘`%`’, ‘`%s`’, ‘`%{s}`’ и ‘`%{sVv}`’.

На выходе образуется строка токенов, разделенных пробелами. Для обратной совместимости первый токен — это имя репозитория, остальные — список запрошенных в форматной строке полей, разделенных запятыми. Например, если репозиторий находится в ‘`/u/src/master`’, форматная строка ‘`%{sVv}`’, а были изменены три файла, (‘`ChangeLog`’, ‘`Makefile`’ и ‘`foo.c`’), то на выходе появится

```
/u/src/master ChangeLog,1.1,1.2 Makefile,1.3,1.4 foo.c,1.12,1.13
```

В качестве другого примера: ‘`%{}`’ означает, что на выходе появится только имя репозитория.

Замечание: когда CVS обращается к сетевому репозиторию, то ‘`loginfo`’ будет исполнен на стороне сервера, а не на стороне клиента (см. Раздел 2.9 [Сетевые репозитории], с. 19).

### C.7.1 Пример использования loginfo

Нижеследующий файл ‘`loginfo`’ с помощью крохотного скрипта добавляет журнальные сообщения к файлу ‘`$CVSROOT/CVSROOT/commitlog`’, а также журналирует в ‘`/usr/adm/cvsroot-log`’ фиксирование изменений в административных файлах. Журнальные записи, соответствующие фиксированию изменений в каталоге ‘`prog1/`’ отсылаются по почте пользователю ‘`ceder`’.

```
ALL             /usr/local/bin/cvs-log $CVSROOT/CVSROOT/commitlog $USER
^CVSROOT        /usr/local/bin/cvs-log /usr/adm/cvsroot-log
^prog1          Mail -s %s ceder
```

Скрипт '`/usr/local/bin/cvs-log`' выглядит так:

```
#!/bin/sh
(echo "-------------------------------------";
 echo -n $2"  ";
 date;
 echo;
 cat) >> $1
```

### C.7.2 Обновление извлеченной копии

Часто бывает полезно хранить дерево каталогов, содержащее файлы, соответствующие последней версии из репозитория. Например, другие разработчики могут захотеть обращаться к последним версиям исходных текстов без необходимости извлекать их, или же вы можете обслуживать с помощью CVS web-сайт и хотели бы, чтобы каждое зафиксированное изменение приводило бы к обновлению файлов, которые показываются web-сервером.

Можно настроить такое поведение с помощью '`loginfo`', который будет вызывать `cvs update`. Если сделать это напрямую, то возникнет проблема с блокировками, поэтому `cvs update` должен выполняться на фоне. Вот пример для операционной системы UNIX (всё это должно помещаться на одной строке):

```
^cyclic-pages (date; cat; (sleep 2; cd /u/www/local-docs;
 cvs -q update -d) &) >> $CVSROOT/CVSROOT/updatelog 2>&1
```

При такой конфигурации фиксирование изменений в каталогах репозитория, которые начинаются с '`cyclic-pages`' приведет к обновлению извлеченного дерева, находящегося в '`/u/www/local-docs`'.

## C.8 Файл rcsinfo

Файл '`rcsinfo`' может использоваться, чтобы задать форму, которая редактируется при заполнении журнала фиксирований. Файл '`rcsinfo`' имеет синтаксис, подобный синтаксису файлов '`verifymsg`', '`commitinfo`' и '`loginfo`'. См. Раздел C.3.1 [Синтаксис], с. 137. В отличие от остальных файлов, правая часть строки является не шаблоном команды, но полным именем файла, содержащего шаблон журнального сообщения.

Если имя в репозитории не соответствует ни одному регулярному выражению в этом файле, используется строка '`DEFAULT`', если она есть.

Все строки, начинающиеся с '`ALL`', используются вдобавок к первому совпадающему регулярному выражению или строке '`DEFAULT`'.

Шаблон журнального сообщения будет использоваться по умолчанию. Если вы зададите журнальное сообщение с помощью '`cvs commit -m` *message*' или '`cvs commit -f` *file*', то вместо шаблона будет использоваться именно оно.

См. Раздел C.5 [Файл verifymsg], с. 138, где приведен пример файла '`rcsinfo`'.

Когда CVS обращается к сетевому репозиторию, используется то содержимое файла '`rcsinfo`', которое было, когда каталог был извлечен в последний раз. Если вы редактируете '`rcsinfo`' или шаблоны, которые используются в нем, вам потребуется заново извлечь рабочий каталог.

## C.9 Игнорирование файлов с помощью cvsignore

Есть определенные имена файлов, которые постоянно находятся в вашем рабочем каталоге, но которые вы не хотите помещать под контроль версий. Примерами являются объектные файлы, получающиеся после компиляции. Обычно когда вы выполняете команду '`cvs update`', она выдает по строке на каждый файл, о котором не знает (см. Раздел A.16.2 [Сообщения команды update], с. 119).

CVS использует список файлов (или шаблонов файлов в стиле sh(1)), которые следует игнорировать при выполнении `update`, `import` и `release`. This list is constructed in the following way.

- Список инициализируется определенными шаблонами имен файлов: имена, служащие для служебных целей CVS и других распространенных систем контроля версий; обычные имена файлов с заплатами, объектных и архивных файлов, а также резервных копий файлов, создаваемых текстовыми редакторами. Остальные имена – побочные продукты деятельности разнообразных утилит. В настоящее время стандартный список шаблонов игнорируемых файлов таков:

```
RCS     SCCS    CVS     CVS.adm
RCSLOG  cvslog.*
tags    TAGS
.make.state     .nse_depinfo
*~      #*      .#*     ,*      _$*     *$
*.old   *.bak   *.BAK   *.orig  *.rej   .del-*
*.a     *.olb   *.o     *.obj   *.so    *.exe
*.Z     *.elc   *.ln
core
```

- Список игнорируемых файлов для каждого репозитория находится в '`$CVSROOT/CVSROOT/cvsignore`' и добавляется к общему списку, если этот файл существует.
- Список игнорируемых файлов для каждого пользователя находится в домашнем каталоге пользователя в файле '`~/.cvsignore`' и добавляется к общему списку, если этот файл существует.
- Содержимое переменной окружения `$CVSIGNORE` добавляется к списку.
- Параметры ключей '`-I`' добавляются к списку.
- Когда CVS обходит дерево каталогов, к списку добавляется содержимое файлов '`.cvsignore`'. Шаблоны, находящиеся в файле '`.cvsignore`', используются только в соответствующем каталоге, но не в его подкаталогах.

Во всех перечисленных местах использование восклицательного знака ('`!`') очищает список. Это можно использовать для хранения файлов, которые обычно игнорируются CVS.

Задание команде `cvs import` ключа '`-I !`' приведет к импорту всего, и обычно вы именно этого и хотите, когда импортируете дистрибутив исходных текстов, не содержащий ничего лишнего. Однако, глядя на вышеприведенные правила, можно заметить ложку дегтя в бочке меда: если в дистрибутиве находятся файлы '`.cvsignore`', то они будут обработаны, даже если в командной строке был указан '`-I !`'. Для того, чтобы импортировать абсолютно все файлы, единственным обходным маневром будет удалить файлы '`.cvsignore`'. Это уродливо, поэтому в будущем '`-I !`' может перестать обрабатывать файлы '`.cvsignore`'.

Заметьте, что синтаксис файла со списком игнорируемых файлов состоит из набора строк, каждая из которых содержит список файлов, разделенных пробелами. Таким образом, нет простого способа задать имена файлов, содержащие пробелы, но можно использовать шаблон '`foo?bar`', чтобы игнорировать файл '`foo bar`' (в этом случае, правда, будет также проигнорирован файл '`fooxbar`' и т. п._). Заметьте, также, что сейчас не существует способа поместить в этот файл комментарии.

## C.10 Файл history

Файл '`$CVSROOT/CVSROOT/history`' используется для журналирования информации для команды `history` (см. Раздел A.11 [Команда history], с. 108). Для того, чтобы включить журналирование, этот файл следует создать. Это происходит автоматически при выполнении команды `cvs init`, которая используется для инициализации репозитория (см. Раздел 2.6 [Создание репозитория], с. 17).

Формат файла '`history`' документирован только в комментариях в исходном тексте CVS, но, в любом случае, обычно программы должны использовать команду `cvs history`, на тот случай, если формат изменится в следующих версиях CVS.

## C.11 Подстановки в административных файлах

Иногда, при написании административного файлы вы хотели бы, чтобы в этом файле можно было бы использовать информацию о среде, в которой выполняется CVS. Есть несколько механизмов, с помощью которых можно этого добиться.

Для того, чтобы узнать домашний каталог пользователя, который запустил CVS (эта информация хранится в переменной окружения `HOME`), используйте '`~`', за которым следует '`/`' или конец строки. Точно так же, для получения домашнего каталога пользователя используйте '`~`*user*'. Подстановка этих переменных происходит на серверной машине, и поэтому такая подстановка не работает, если используется `pserver` (см. Раздел 2.9.3 [Парольная аутентификация], с. 21). Для того, чтобы изменить поведение для каждого пользователя, лучше использовать пользовательские переменные (см. ниже).

Иногда требуется узнать различную информацию, используемую CVS. Внутренняя переменная CVS имеет такой синтаксис: `${`*переменная*`}`, где *переменная* начинается с буквы и состоит из алфавитно-цифровых символов и символа подчерка ('`_`'). Если символ, который следует за *variable*, не является буквой, цифрой или знаком подчерка, то фигурные скобки можно опустить. Внутренние переменные CVS таковы:

`CVSROOT` Здесь хранится корневой каталог используемого репозитория. См. Глава 2 [Репозиторий], с. 7, где описаны различные способы задания корневого каталога.

`RCSBIN` В CVS 1.9.18 и раньше в этой переменной находился каталог, в котором находились программы RCS. Так как теперь CVS более не запускает RCS, использование этой внутренней переменной запрещено.

`CVSEDITOR`
`VISUAL`
`EDITOR` Эти три переменных содержат одно и то же значение – используемый текстовый редактор. См. Раздел A.4 [Глобальные ключи], с. 90, где описано, как задать этот редактор.

`USER` Имя пользователя, запустившего CVS (на серверной машине).

Если вы хотите, чтобы пользователь мог задать какое-то значение, передающееся в административный файл, используйте пользовательскую переменную. Для подстановки такой переменной в административном файле написано `${=`*variable*`}`. Для того, чтобы установить пользовательскую переменную, задайте CVS глобальный флаг '`-s`' с аргументом *переменная=значение*. Особенно полезно будет задать такой флаг в файле '`~/.cvsrc`' (см. Раздел A.3 [~/.cvsrc], с. 90).

Например, если вы хотите, чтобы административный файл ссылался на тестовый каталог, вы можете создать пользовательскую переменную `TESTDIR`. Затем, если запустить CVS как

```
cvs -s TESTDIR=/work/local/tests
```

и при административном файле, содержащем `sh ${=TESTDIR}/runtests`, то эта строка преобразуется в `sh /work/local/tests/runtests`.

Все другие строки, содержащие '`$`', зарезервированы; нет способа экранировать символ '`$`', чтобы он обозначал сам себя.

## C.12 Файл конфигурации CVSROOT/config

Административный файл '`config`' содержит различные настройки, влияющие на поведение CVS. Синтаксис этого файла слегка отличается от синтаксиса прочих файлов. Переменные не подставляются. Строки, начинающиеся с '`#`', считаются комментариями.

Все прочие строки состоят из ключевого слова, символа '`=`' и значения. Заметьте, что этот синтаксис очень строг. Дополнительные пробелы и символы табуляции не допускаются.

В настоящий момент определены следующие ключевые слова:

`RCSBIN=`*bindir*
Для CVS версий от 1.9.12 до 1.9.18, это ключевое слово указывало, что следует искать программы RCS в каталоге *bindir*. Современные версии CVS не требуют программ RCS; для совместимости эта установка допускается, но ничего не делает.

`SystemAuth=`*value*

Если *value* равно ‘`yes`’, то pserver должен искать пользователя в системной базе данных пользователей, если он не найден в ‘`CVSROOT/passwd`’. Если же значение равно ‘`no`’, то все пользователи сервера с парольной аутентификацией должны существовать в ‘`CVSROOT/passwd`’. По умолчанию значение равно ‘`yes`’. Дополнительная информация о pserver находится в Раздел 2.9.3 [Парольная аутентификация], с. 21.

`PreservePermissions=`*value*

Включить поддержку для хранения в репозитории специальных файлов устройств, символических ссылок, прав доступа к файлами и информации об их владельцах. Значение по умолчанию: ‘`no`’. См. Глава 15 [Специальные файлы], с. 87, где описаны подробности использования этого ключевого слова.

`TopLevelAdmin=`*value*

Изменить поведение команды ‘`checkout`’ так, чтобы она создавала каталог ‘`CVS/`’ на уровень выше вашего рабочего каталога, вдобавок к каталогам ‘`CVS/`’, которые создаются внутри извлеченных каталогов. Значение по умолчанию – ‘`no`’.

Эта опция полезна, если вы обнаружите, что выполняете многие команды в каталоге на уровень выше вашего рабочий каталога, а не в одном из извлеченных подкаталогов. Каталог ‘`CVS/`’, созданный таким образом, позволяет не указывать ‘`CVSROOT`’ при каждой команде. Обеспечивается также место для файла ‘`CVS/Template`’ (см. Раздел 2.3 [Хранение файлов в рабочем каталоге], с. 13).

`LockDir=`*directory*

Создавать файлы блокировок CVS в каталоге *directory*, а не в репозитории. Это полезно, если вы хотите разрешить пользователям читать из репозитория, предоставив им доступ на запись только в *directory*, а не в репозиторий. Вам нужно создать *directory*, а CVS сама создаст там требуемые подкаталоги. Информация о блокировках CVS находится в главе Раздел 10.5 [Совместный доступ], с. 68.

Перед включением опции ‘`LockDir`’ убедитесь, что вы не используете ни одной копии CVS версий 1.9 или раньше, которые не поддерживают ‘`LockDir`’, и не дадут об этом никакого предупреждения. Если позволить такому случиться, то некоторые пользователи CVS будут делать блокировки в одном каталоге, а другие — в другом, и репозиторий может быть испорчен. CVS 1.10 не поддерживает ‘`LockDir`’, но выдаст предупреждение, если использовать его на репозитории с включенным ‘`LockDir`’.

# Приложение D Все переменные окружения, используемые в CVS

Вот полный список переменных окружения, влияющих на работу CVS.

`$CVSIGNORE`
: Список шаблонов файлов, разделенный пробелами. CVS будет игнорировать эти файлы. См. Раздел C.9 [Файл cvsignore], с. 143.

`$CVSWRAPPERS`
: Список имен файлов, разделенный пробелами. CVS будет считать такие файлы обертками. См. Раздел C.2 [Обертки], с. 136.

`$CVSREAD`
: Если эта переменная установлена, команды `checkout` и `update` будут стараться создавать файлы в вашем рабочем каталоге в режиме только для чтения. Если эта переменная не установлена, то поведением по умолчанию будет разрешить изменение ваших рабочих файлов.

`$CVSUMASK`
: Управляет правами доступа к файлам в репозитории. См. Раздел 2.2.2 [Права доступа к файлам], с. 10.

`$CVSROOT`
: Эта переменная должна содержать полный путь к репозиторию исходных текстов CVS (там, где хранятся RCS-файлы). Эта информация должна наличествовать для выполнения большинства команды CVS; если `$CVSROOT` не установлена, или же вы хотите один раз использовать другой репозиторий, вы можете использовать такую командную строку: '`cvs -d cvsroot cvs_command...`'. После того, как вы извлекли рабочий каталог, CVS сохраняет путь к репозиторию в файле '`CVS/Root`', поэтому обычно вам нужно беспокоиться об этом только при первом извлечении.

`$EDITOR`
`$CVSEDITOR`
`$VISUAL`
: Задает программу, использующуюся для написания журнальных сообщений во время фиксирования. `$CVSEDITOR` переопределяет `$EDITOR`. См. Раздел 1.3.2 [Фиксирование изменений], с. 4.

`$PATH`
: Если переменная `$RCSBIN` не установлена, и путь поиска программ не задан на этапе компиляции, то CVS будет использовать `$PATH`, чтобы найти все используемые программы.

`$HOME`

`$HOMEPATH`

`$HOMEDRIVE`
: Используется для установки каталога, в котором хранится файл '`.cvsrc`' и некоторые другие файлы. Под UNIX CVS проверяет только `$HOME`. Под Windows NT система устанавливает переменные `$HOMEDRIVE`, например, '`d:`' и `$HOMEPATH`, например, '`\joe`'. Под Windows 95 вам, скорее всего, потребуется самому установить `$HOMEDRIVE` и `$HOMEPATH`.

`$CVS_RSH`
: Задает внешнюю программу, с помощью которой CVS устанавливает соединение, когда используется метод доступа `:ext:`. см. Раздел 2.9.2 [Соединение с помощью rsh], с. 20.

`$CVS_SERVER`
: Используется в режиме клиент-сервер при обращении к сетевому репозиторию с помощью `rsh`. В этой переменной задается имя программы, которую нужно запустить на сервере при доступе к сетевому репозиторию с помощью `rsh`. Значение по умолчанию — `cvs`. см. Раздел 2.9.2 [Соединение с помощью rsh], с. 20.

`$CVS_PASSFILE`
: Используется в режиме клиент-сервер при обращении к `cvs login server`. Значение по умолчанию — '`$HOME/.cvspass`'. см. Раздел 2.9.3.2 [Парольная аутентификация клиента], с. 23.

`$CVS_CLIENT_PORT`
: Используется в режиме клиент-сервер при доступе к серверу с помощью Kerberos. см. Раздел 2.9.5 [Аутентификация с помощью Kerberos], с. 26.

`$CVS_RCMD_PORT`
: Используется в режиме клиент-сервер. Если установлено, задает номер порта, который используется при обращении к демону RCMD на сервере. (В настоящий момент не используется для клиентов Unix).

`$CVS_CLIENT_LOG`
: Используется для отладки в режиме клиент-сервер. Если установлено, то все, что посылается на сервер, журналируется в файле '`$CVS_CLIENT_LOG.in`', а все, что принимается от сервера, журналируется в '`$CVS_CLIENT_LOG.out`'.

`$CVS_SERVER_SLEEP`
: Используется для отладки на стороне сервера в режиме клиент-сервер. Если установлена, задерживает запуск нового процесса-сервера на указанный период времени, чтобы вы могли присоединить к процессу отладчик.

`$CVS_IGNORE_REMOTE_ROOT`
: Для CVS 1.10 и старше установка этой переменной не позволяет CVS перезаписывать файл '`CVS/Root`', когда задан глобальный ключ '`-d`'. Поздние версии CVS не перезаписывают этот файл, поэтому `$CVS_IGNORE_REMOTE_ROOT` игнорируется.

`$COMSPEC`
: Используется только под OS/2. Задает имя командного интерпретатора, по умолчанию '`cmd.exe`'.

`$TMPDIR`
`$TMP`
`$TEMP`
: Каталог, в котором расположены временные файлы. CVS-сервер использует `TMPDIR`. См. Раздел A.4 [Глобальные ключи], с. 90, где описано, как задать этот параметр. Некоторые части CVS всегда используют '`/tmp`' (с помощью функции `tmpnam()`, которая обеспечивается системой).

  Под Windows NT используется `$TMP` (с помощью функции `_tempnam()`, которая обеспечивается системой).

  Программа `patch`, которая используется клиентом CVS, использует `TMPDIR`, а если она не установлена, то '`/tmp`' (по крайней мере, это так для GNU

patch 2.1). Заметьте, что если ваши сервер и клиент оба используют CVS 1.9.10 или позже, то CVS не вызывает внешнюю программу `patch`.

# Приложение E Совместимость между версиями CVS

Формат репозитория совместим со старыми версиями вплоть до CVS 1.3. Обратитесь к Раздел 10.6.5 [Совместимость слежений], с. 72, если у вас есть копии CVS 1.6 или раньше, и вы хотите использовать новые возможности для общения разработчиков.

Формат рабочего каталога совместим с версиями вплоть до CVS 1.5. Этот формат изменился между версиями CVS 1.3 и CVS 1.5. Если вы используете CVS 1.5 или новее в рабочем каталоге, извлеченном с помощью CVS 1.3, то произойдет автоматическая конвертация, но для того, чтобы вернуться обратно к CVS 1.3, нужно извлечь новый рабочий каталог с помощью CVS 1.3.

Протокол общения с сетевым сервером понимается версиями вплоть до CVS 1.5, но не далее (1.5 была первой официальной версией, поддерживающей сетевой протокол, но некоторые старые версии всё еще могут использоваться где-нибудь). Во многих случаях для того, чтобы воспользоваться новыми возможностями и исправлениями, требуется обновить как клиента, так и сервер.

# Приложение F Исправление ошибок

Если у вас есть какие-то проблемы при использовании CVS, то вам поможет это приложение. Если вы видите конкретное сообщение об ошибке, то можете найти его здесь по алфавиту. В противном случае обратитесь к главе о прочих проблемах и попробуйте найти там свою.

## F.1 Частичный список сообщений CVS

Вот частичный список сообщений об ошибке, которые может выдавать CVS. Это неполный список — CVS может выдавать множество разнообразных сообщений об ошибке, часть которых выдается операционной системой. Назначение этого раздела — перечислить обычные и/или потенциально непонятные сообщения.

Сообщения перечислены в алфавитном порядке, но вводный текст, такой как ‘`cvs update:` ’ не учитывается.

В некоторых случаях в этом списке находятся сообщения, которые выдаются старыми версиями CVS (частично из-за того, что пользователи могут быть не уверены, какую версию CVS они используют в настоящий момент).

`cvs` *command*`: authorization failed: server` *host* `rejected access`
: Это — неспецифическое сообщение при попытке соединиться с сервером парольной аутентификации, который отказывает в авторизации без указания конкретной причины. Проверьте, что указанные имя пользователя и пароль верны, и что заданный `$CVSROOT` разрешен с помощью ключа ‘`-allow-root`’ в ‘`/etc/inetd.conf`’. См. Раздел 2.9.3 [Парольная аутентификация], с. 21.

*file*`:`*line*`: Assertion ’`*text*`’ failed`
: Точный формат этого сообщения может варьироваться в зависимости от вашей системы. Это сообщение указывает на ошибку реализации CVS, что делать с которой, написано в Приложение H [Ошибки в CVS], с. 161.

`cvs` *command*`: conflict: removed` *file* `was modified by second party`
: Это сообщение указывает, что вы удалили файл, а кто-то еще изменил его в то же самое время. Для того, чтобы справиться с этим конфликтом, сначала выполните ‘`cvs add` *file*’. Если требуется, взгляните на внесенные изменения и выясните, хотите ли вы все еще удалять его. Если нет, то больше ничего делать не надо. Если да, то еще раз скажите ‘`cvs remove` *file*’ и зафиксируйте изменения.

`cannot change permissions on temporary directory`
`Operation not permitted`
: Это сообщение иногда появлялось при выполнении набора тестов под RedHat Linux 3.0.3 и 4.1, причем повторить его, а также выяснить его причину, мы не смогли. Неизвестно, проявляется ли эта ошибка только под Linux (или вообще только на этой конкретной машине!). Если проблема не случается на других UNIXах, то, скорее всего, сообщением об ошибке будет ‘`Not owner`’ или другое сообщение, возникающее при ошибке `EPERM`,

вместо ‘`Operation not permitted`’. Если вы можете что-то добавить, сообщите нам, как описано в Приложение H [Ошибки в CVS], с. 161. Если вы сталкиваетесь с этой ошибкой при использовании CVS, следует повторить операцию, и она пройдет успешно.

`cannot open CVS/Entries for reading: No such file or directory`
: Обычно это означает внутреннюю ошибку CVS, с которой можно справиться так же, как и с другими (см. Приложение H [Ошибки в CVS], с. 161). Обычно эту ошибку можно обойти. Надеемся, что в конкретной ситуации будет ясно, как это сделать.

`cvs [init aborted]: cannot open CVS/Root: No such file or directory`
: Это сообщение совершенно безвредно. Если оно не сопровождается другими сообщениями об ошибке, значит, операция завершится успешно. Это сообщение не должно появляться в свежих версиях CVS, но документировано здесь для удобства пользователей версий CVS 1.9 и раньше.

`cvs [checkout aborted]: cannot rename file` *file* `to CVS/`■*file*`: Invalid argument`
: Это сообщение, по отзывам, появляется от случая к случаю при работе с CVS 1.9 под Solaris 2.5. Причина неизвестна; если вы можете что-нибудь сказать по этому поводу, сообщите нам, как описано в Приложение H [Ошибки в CVS], с. 161.

`cvs [`*command* `aborted]: cannot start server via rcmd`
: Это, к сожалению, довольно неспецифическое сообщение об ошибке, которое CVS версии 1.9 выдает, если вы выполняете клиента CVS и при соединении с сервером появляются проблемы. Текущие версии CVS должны выдавать более конкретное сообщение. Если вы получили это сообщение, совершенно не имея в виду запускать клиента CVS, значит, вероятно, вы забыли указать `:local:`, как описано в Глава 2 [Репозиторий], с. 7.

`ci:` *file*`,v: bad diff output line: Binary files - and /tmp/T2a22651 differ`
: CVS 1.9 и старше выдают это сообщение при попытке зафиксировать измененный двоичный файл, если система RCS установлена неправильно. Перечитайте инструкции из дистрибутива RCS и файл ‘`INSTALL`’ в дистрибутиве CVS. Еще можно обновить версию CVS, которая сама будет заниматься фиксированием, без необходимости использовать RCS.

`cvs checkout: could not check out` *file*
: При работе с CVS 1.9 это сообщение может означать, что программа `co` (из комплекта RCS) завершилась с ошибкой. Перед этим сообщением должно быть другое, с объяснением причины, однако, наблюдались и ситуации с отсутствием оного, которые так и не объяснены. При работе с текущей версией CVS, которая не использует `co`, появление этого сообщения без сопровождающего объяснения определенно означает ошибку в CVS (см. Приложение H [Ошибки в CVS], с. 161).

`cvs [login aborted]: could not find out home directory`
: Это означает, что вам требуется установить переменные окружения, которые CVS использует, чтобы найти ваш домашний каталог. См. обсуждение

‘$HOME’, ‘$HOMEDRIVE’ и ‘$HOMEPATH’ в Приложение D [Переменные окружения], с. 147.

`cvs update: could not merge revision` *rev* `of` *file*`: No such file or directory`
: CVS 1.9 и раньше выдают это сообщение, если не смогли найти программу `rcsmerge`. Убедитесь, что она находится в вашем `PATH`, или поставьте свежую версию CVS, которая не требует внешней программы `rcsmerge`.

`cvs [update aborted]: could not patch` *file*`: No such file or directory`
: Это означает, что не обнаружена программа `patch`. Убедитесь, что она находится в вашем `PATH`. Заметьте, что, несмотря на формулировку сообщения, оно *не* относится к файлу ‘`file`’. Если клиент и сервер оба используют текущую версию CVS, то внешняя программа `patch` не требуется и вы не должны получать такое сообщение. Если же клиент либо сервер используют CVS 1.9, то программа `patch` требуется.

`cvs update: could not patch` *file*`; will refetch`
: Это означает, что по какой-то причине клиент не смог применить файл изменений, посланный ему сервером. Можно не беспокоиться, получив это сообщение — работа CVS просто несколько замедлится, но никак не повлияет на конечный результат.

`dying gasps from` *server* `unexpected`
: В сервере версий CVS 1.9.18 и раньше имеется известная ошибка, приводящая к такому сообщению. У меня лично получалось вызывать её, используя глобальный ключ ‘`-t`’. Эта ошибка в файле ‘`src/filesubr.c`’ была исправлена Энди Пайпером (Andy Piper) 14 ноября 1997 года, если кому-то интересно. Если вы видите это сообщение, просто повторите команду, или, если вы обнаружили её причину, дайте нам знать, как описано в Приложение H [Ошибки в CVS], с. 161.

`end of file from server (consult above messages if any)`
: Самая распространенная причина этого сообщения – вы используете внешнюю программу `rsh` и она завершилась с кодом ошибки. В этом случае она должна была напечатать сообщение, которое появится перед обсуждаемым сообщением. Дальнейшая информация о настройке клиента и сервера CVS находится в Раздел 2.9 [Сетевые репозитории], с. 19.

`cvs commit: Executing 'mkmodules'`
: Это означает, что ваш репозиторий настроен для версии CVS раньше 1.8. При использовании CVS 1.8 и позже перед обсуждаемым сообщением появится еще одно:

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

: Если вы видите оба сообщения, то база данных перестраивается дважды, что необязательно, но нестрашно. Если вы хотите избежать ненужной работы, и не используете версию CVS 1.7 или раньше, удалить `-i mkmodules` везде, где эта строка появляется в файле ‘`modules`’. Дальнейшая информация об этом файле находится в Раздел C.1 [Файл modules], с. 133.

`missing author`
: Обычно это происходит, если вы создали RCS-файл с пустым именем пользователя. CVS может по ошибке создать такой файл. Решение — убедиться, что имя пользователя установлено в непустое значение и пересоздать RCS-файл.

`*PANIC* administration files missing`
: Это обычно означает, что существует каталог ‘`CVS/`’, но в нем нет административных файлов, которые обычно помещает туда CVS. Если проблема в том, что вы создали каталог ‘`CVS/`’ как-то по-другому, не с помощью CVS, то ответ прост — используйте другое имя. В противном случае это сообщение означает ошибку в CVS (см. Приложение H [Ошибки в CVS], с. 161).

`rcs error: Unknown option: -x,v/`
: Вслед за этим сообщением будет информация о правильном использовании RCS. Это означает, что у вас старая версия RCS (вероятно, входящая в комплект операционной системы). CVS работает только с RCS версии 5 и старше.

`cvs [server aborted]: received broken pipe signal`
: Это сообщение вызывается какой-то очень сложной ошибкой в CVS или системах, под которыми CVS работает (мы не знаем, какой именно, потому что еще не отследили эту ошибку!). Кажется, она возникает только после завершения команды CVS, и вам, скорее всего, нужно лишь игнорировать это сообщение. Однако, если вы обнаружили причину этого сообщения, дайте нам знать, как описано в Приложение H [Ошибки в CVS], с. 161.

`Too many arguments!`
: Обычно это сообщение выдается скриптом ‘`log.pl`’, находящимся в каталоге ‘`contrib/`’ в дистрибутиве CVS. В некоторых версиях CVS этот скрипт устанавливался по умолчанию. Он вызывается из административного файла ‘`loginfo`’. Проверьте, что аргументы, которые передаются этому скрипту из файла ‘`loginfo`’, совпадают с теми, которые ожидаются. В частности, ‘`log.pl`’ из CVS 1.3 и раньше ожидает имя журнального файла в качестве аргумента, тогда как ‘`log.pl`’ версии 1.5 и новее получает имя журнального файла с помощью ключа ‘`-f`’. Конечно, если вам не нужен ‘`log.pl`’, то просто закомментируйте его в ‘`loginfo`’.

`cvs [login aborted]: unrecognized auth response from` *server*
: Это сообщение обычно означает, что сервер не был настроен должным образом. Например, строка в файле ‘`/etc/inetd.conf`’ задает имя несуществующего исполняемого файла. Для исправления ошибок найдите журнальный файл, используемый ‘`inetd`’ом, например, ‘`/var/log/messages`’. Детали обсуждаются в Раздел F.2 [Соединение], с. 157, а также Раздел 2.9.3.1 [Сервер парольной аутентификации], с. 21.

`cvs commit: Up-to-date check failed for` ‘*file*’
: Это означает, что кто-либо еще зафиксировал изменения в файл с тех пор, как вы последний раз делали `cvs update`. Сделайте `cvs update` и повто-

рите `cvs commit`. CVS объединит изменения, которые сделали вы, с изменениями, сделанными остальными. Если не случится конфликтов, то CVS выдаст сообщение '`M cacErrCodes.h`', и можно сразу выполнять `cvs commit`. Если обнаружены конфликты, то CVS сообщит об этом, сказав, что '`C cacErrCodes.h`', и вам потребуется вручную устранить конфликт. Дальнейшие детали этого процесса обсуждаются в Раздел 10.3 [Пример конфликта], с. 65.

`Usage: diff3 [-exEX3 [-i | -m] [-L label1 -L label3]] file1 file2 file3`
`Only one of [exEX3] allowed`

Это указывает на проблему с установленными программами `diff3` и `rcsmerge`. Точнее говоря, `rcsmerge` скомпилирован так, что должен использовать GNU-версию `diff3`, а вместо этого находит UNIX-версию. Точный текст сообщения разный на разных системах. Самым простым решением будет обновить версию CVS, которая не использует внешних программ `rcsmerge` и `diff3`.

`warning: unrecognized response '`*text*`' from cvs server`

Если *text* содержит разрешенный текст ответа (например, '`ok`'), за которым следует дополнительный символ возврата каретки (на многих системах это приведет к тому, что вторая часть сообщения перезапишет первую часть), то это, вероятно, означает, что вы используете метод доступа '`:ext:`' с такой версией `rsh`, которая, как большинство не-UNIX версий, не обеспечивает прозрачного потока данных. В этом случае попробуйте '`:server:`' вместо '`:ext:`'. Если в *text* содержится что-то ещё, это может означать проблемы с вашим CVS-сервером. Ещё раз проверьте, как вы установили CVS-сервер.

`cvs commit: [`*time*`] waiting for `*user*`'s lock in `*directory*

Это нормальное сообщение, а не ошибка. Смотри Раздел 10.5 [Совместный доступ], с. 68, где описаны детали.

`cvs commit: warning: editor session failed`

Это означает, что редактор, используемый CVS, возвращает ненулевой код завершения. Некоторые версии `vi` делают это даже в том случае, если при редактировании файла не было ни одной ошибки. Если это так, то пусть ваша переменная окружения '`CVSEDITOR`' указывает на маленький скрипт, например

```
#!/bin/sh
vi $*
exit 0
```

## F.2 Ошибки при установке соединения с CVS-сервером

В этой главе обсуждается, что делать, если у вас проблемы с установкой соединения с CVS-сервером. Если вы использует клиент командной строки CVS под Windows, сначала обновите его до версии 1.9.12 или более поздней. Сообщения об ошибках в старой версии предоставляли значительно меньше информации о местонахождении

проблемы. Если клиент работает под другой операционной системой, то CVS 1.9 вполне достаточно.

Если сообщений об ошибках недостаточно, чтобы отследить проблему, то следующие шаги сильно зависят от используемого метода доступа.

`:ext:` Попробуйте запустить программу `rsh` из командной строки. Например,

```
$ rsh servername cvs -v
```

должно выдать информацию о версии CVS. Если это не срабатывает, то ваш сервер нужно исправить, прежде чем беспокоиться о проблемах с CVS.

`:server:` Для того, чтобы использовать этот метод доступа, программа `rsh` не требуется, но она может быть полезна в качестве средства отладки. Следуйте инструкциям, приведенным для метода '`:ext:`'.

`:pserver:`

Хорошим средством отладки является

```
$ telnet servername 2401
```

После соединения напечатайте любой текст, например, '`foo`', нажмите ⟨RET⟩. Если CVS работает, то ответом будет

```
cvs [pserver aborted]: bad auth protocol start: foo
```

В противном случае убедитесь, что `inetd` работает правильно. Замените вызов CVS в файле '`/etc/inetd.conf`' на программу '`echo`'. Например:

```
2401  stream  tcp  nowait  root /bin/echo echo hello
```

Теперь сделайте так, чтобы '`inetd`' перечитал свой файл конфигурации, попробуйте '`telnet`' ещё раз, и вы должны увидеть слово '`hello`', а затем сервер должен закрыть соединение. Если это не так, то нужно исправить ваш '`inetd`', перед тем, как беспокоиться о CVS.

На системах под AIX зачастую порт 2401 уже используется системой. Это проблема AIX в том смысле, что порт 2401 зарегистрирован для CVS. Я слышал, что есть исправление этой проблемы под AIX.

## F.3 Другие распространенные проблемы

Вот список проблем, не попадающих ни в одну из вышеперечисленных категорий.

- Если вы используете CVS 1.9.18 или раньше, и `cvs update` обнаруживает конфликт и пытается слить изменения, как описано в Раздел 10.3 [Пример конфликта], с. 65, но не сообщает вам, где именно присутствуют конфликты, значит, вы используете старую версию RCS. Самым простым решением, вероятно, будет обновить версию CVS до той, которая не нуждается во внешних программах RCS.

# Приложение G Титры

Roland Pesch (roland@wrs.com), когда-то работавший в Cygnus Support, написал страницы руководства, которые распространялись с CVS версии 1.3. Большая часть их текста была скопирована в это руководство. Она также читал ранние черновики этого руководства и внес множество идей и исправлений.

Список рассылки `info-cvs` иногда бывает информативным. В это руководство включена информация их писем David G. Grubbs (dgg@think.com).

Часть текста извлечена из страниц руководства по RCS.

В часто задаваемых вопросах (FAQ) по CVS, созданных тем же Дэвидом Г. Грубб-сом, нашлось множество полезного материала. Этот FAQ, однако, больше не поддерживается, а это руководство является его ближайшим наследником (по крайней мере, с точки зрения использования CVS).

Вдобавок, эти люди помогли мне, указав на совершенные ошибки:

Roxanne Brunskill <rbrunski@datap.ca>,
Kathy Dyer <dyer@phoenix.ocf.llnl.gov>,
Karl Pingle <pingle@acuson.com>,
Thomas A Peterson <tap@src.honeywell.com>,
Inge Wallin <ingwa@signum.se>,
Dirk Koschuetzki <koschuet@fmi.uni-passau.de>
and Michael Brown <brown@wi.extrel.com>.

Полный список участников создания руководства можно найти в файле ‘`doc/ChangeLog`’ в дистрибутиве исходных текстов CVS.

# Приложение H Что делать с ошибками в CVS и этом руководстве?

Ни CVS, ни это руководство не совершенны, и, вероятно, никогда не будут таковыми. Если у вас проблемы с использованием CVS, или если вы считаете, что обнаружили ошибку, есть несколько вещей, которые можно предпринять. Заметьте, что если в руководстве есть нечеткие места, то это можно посчитать ошибкой, поэтому стоило бы что-нибудь предпринять, точно так же, как если бы это было ошибкой в CVS.

- Если вы хотите, чтобы кто-нибудь помог вам и исправил найденные ошибки, есть компании, которые сделают это за определенную плату. Вот две такие компании:

  ```
  Signum Support AB
  Box 2044
  S-580 02  Linkoping
  Sweden
  Email: info@signum.se
  Phone: +46 (0)13 - 21 46 00
  Fax:   +46 (0)13 - 21 47 00
  http://www.signum.se/

  Cyclic Software
  United States of America
  http://www.cyclic.com/
  info@cyclic.com
  ```

- Если вы получили CVS от распространителя, например, производителя операционной системы или поставщика компакт-дисков со свободным программным обеспечением, вы можете выяснить, предоставляет ли этот распространитель поддержку. Обычно они этого не делают, или же предоставляют предельно минимальную поддержку, но у разных распространителей по разному.
- Если ваши способности и время позволяют, вы можете сами исправить ошибку. Если вы хотите, чтобы ваше исправление вошло в очередную версию CVS, смотрите файл ‘`HACKING`’ в дистрибутиве исходных текстов CVS. Там содержится гораздо больше информации о процессе внесения исправлений.
- В сети могут быть ресурсы, которые могут помочь. Два хороших места для начала таковы:

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

  Если вы вдохновитесь чем-нибудь, то мы будем очень признательны за увеличение количества информации, доступной в сети. Например, до того, как стандартный дистрибутив CVS заработал под Windows 95, была создана web-страница с объяснениями и заплатами, а различные участники списка рассылки помогали, упоминая эту страницу при возникновении вопросов.
- Можно также сообщить об ошибке в `bug-cvs`. Заметьте, что необязательно ваше сообщение об ошибке будет учтено. Что делать, если вам требуется решение — описано выше. В основном там хотят слышать об особенно опасных или легких ошибках. Вы можете увеличить шансы успешного исхода, максимально четко описав ошибку и добавив всю необходимую дополнительную информацию. Сообщить об ошибке можно, отправив письмо по адресу `bug-cvs@gnu.org`. Заметьте,

что исправления, оказавшиеся в `bug-cvs`, могут распространяться на условиях Публичной Лицензии GNU. Если вам это не нравится – не присылайте исправление. Обычно не следует слать письма напрямую одному из разработчиков CVS, потому что те из них, кто заинтересован в получении сообщений об ошибках, читают `bug-cvs`. Заметьте также, что отправка сообщений об ошибках в другие списки рассылки или группы новостей *не* заменяет писем в `bug-cvs`. Можно обсуждать ошибки в CVS в любом форуме, но если вы хотите, чтобы ваше сообщений прочитал один из разработчиков, используйте `bug-cvs`.

Часто задают вопрос, имеется ли список известных ошибок, и известна ли уже конкретная ошибка. Файл '`BUGS`' в дистрибутиве исходных текстов CVS является одним из таких списков известных ошибок, но он необязательно полон. Возможно, полного списка никогда не будет.

# Индекс

## default



## A



## B



## C

## D



## E



## F



## G



## H



## I



## K



## L



## M



## N

## P



## R



## S



## T



## U



## V



## W



## Z



## A

## Б



## В



## Г



## Д



## Ж



## З



## И

# K

## Л



## М



## Н



## О



## П

## P



## C

## Т



## У



## Ф



## Х



## Ч



## Ш



## Э

# Краткое содержание

# Оглавление