Введение в CVS

Конспект первого дня двухдневного курса по CVS

Jim Blandy
Перевод на русский язык: Алексей Махоткин
alexm@hsys.msk.ru, http://alexm.here.ru

1 Для чего нужен CVS?

CVS поддерживает историю дерева каталогов с исходным кодом, работая с последовательностью изменений. CVS маркирует каждое изменение моментом времени, когда оно было сделано, и именем пользователя, совершившим изменение. Обычно человек, совершивший изменение, также предоставляет текстовое описание причины, по которой произошло изменение. Вооружившись всей этой информацией, CVS может отвечать на такие вопросы, как

- Кто совершил данное изменение?
- Когда они его совершили?
- Зачем они это сделали?
- Какие еще изменения произошли в то же самое время?

2 Как использовать CVS — первый набросок

Перед обсуждением множества разнобразных терминов и идей, давайте взглянем на основные команды CVS.

2.1 Настройка вашего репозитория

CVS хранит все изменения в данном проекте в дереве каталогов, называемом *репозиторием* (repository). Перед тем, как начать использовать CVS, вам необходимо настроить переменную среды `CVSROOT` так, чтобы она указывала на каталог репозитория. Тот, кто ответственен за управление конфигурацией вашего проекта, вероятно, знает, что именно должна содержать в себе эта переменная; возможно даже, что переменная `CVSROOT` уже установлена глобально.

В любом случае, на нашей системе репозиторий находится в '`/usr/src/master`'. В этом случае вам следует ввести команды

‘ `setenv CVSROOT /usr/src/master` ’

если ваш командный интерпретатор — `csh` или порожден от него, или

‘ `CVSROOT=/usr/src/master export CVSROOT` ’

если это Bash или какой-либой другой вариант Bourne shell.

Если вы забудете сделать это, CVS пожалуется, если вы попытаетесь запустить его:

```
$ cvs checkout httpc
cvs checkout: No CVSROOT specified! Please use the ‘-d’ option
cvs [checkout aborted]: or set the CVSROOT environment variable.
$
```

2.2 Извлечение рабочего каталога

CVS не может работать в обычном дереве каталогов; наоборот, вы должны работать в каталоге, который CVS создаст для вас. Точно так же, как вы выписываете книгу из библиотеки перед тем, как забрать ее с собой, вам следует использовать команду ‘`cvs checkout`’, чтобы получить от CVS рабочее дерево каталогов. Предположим, например, что вы работаете над проектом, называемым ‘`httpc`’, тривиальным HTTP клиентом:

```
$ cd
$ cvs checkout httpc
U httpc/.cvsignore
U httpc/Makefile
U httpc/httpc.c
U httpc/poll-server
$
```

Команда ‘`cvs checkout httpc`’ означает “Извлечь дерево исходных текстов с именем ‘`httpc`’ из репозитория, указанного в переменной окружения ‘`CVSROOT`”’.

CVS помещает дерево в подкаталог ‘`httpc`’:

```
$ cd httpc
$ ls -l
total 8
drwxr-xr-x      2 jimb          512 Oct 31  11:04 CVS
-rw-r-r-      1 jimb           89 Oct 31  10:42 Makefile
-rw-r-r-      1 jimb         4432 Oct 31  10:45 httpc.c
-rwxr-xr-x      1 jimb          460 Oct 30  10:21 poll-server
```

Большинство этих файлов — рабочие копии исходных текстов ‘`httpc`’. Однако, подкаталог с именем ‘`CVS`’ (самый первый) имеет другое назначение. CVS использует его для хранения дополнительной информации о каждом файле в этом каталоге, чтобы определять, какие изменения вы внесли в них с тех пор, как извлекли их из репозитория.

2.3 Редактирование файлов

После того, как CVS создал рабочее дерево каталогов, вы можете обычным образом редактировать, компилировать и проверять находящиеся в нем файлы — это просто файлы.

Например, предположим, что мы хотим скомпилировать проект, который мы только что извлекли:

```
$ make
gcc -g -Wall -lnsl -lsocket httpc.c  -o httpc
httpc.c: In function ‘tcp_connection’:
httpc.c:48: warning: passing arg 2 of ‘connect’ from incompatible
pointer type
$
```

Кажется, ‘`httpc.c`’ еще не был перенесен на эту операционную систему. Нам нужно сделать приведение типов для одного из аргументов функции `connect`. Чтобы сделать это, надо изменить строку 48, заменив

```
if (connect (sock, &name, sizeof (name)) >=  0)
```

на

```
if (connect (sock, (struct sockaddr *) &name, sizeof (name)) >= 0)
```

Теперь компиляция должна пройти успешно:

```
$ make
gcc -g -Wall -lnsl -lsocket httpc.c  -o httpc
$ httpc GET http://www.cyclic.com
...здесь находится текст домашней страницы Cyclic Software...
$
```

2.4 Объединение изменений

Так как каждый разработчик использует свой собственный рабочий каталог, изменения, которые вы делаете в своем каталоге, не становятся автоматически видимыми всем остальным в вашей команде. CVS не публикует изменений, пока они не закончены. Когда вы протестируете изменения, вы должны *зафиксировать* (commit) их в репозитории и сделать их доступными остальным. Мы опишем команду `cvs commit` далее.

Однако, что если другой разработчик изменил тот же файл, что и вы, и, может быть, даже изменил те же самые строки? Чьи изменения будет использованы? Обычно ответить на этот вопрос автоматически невозможно, и CVS совершенно точно некомпетентен, чтобы принимать такие решения.

Поэтому перед тем, как фиксировать ваши изменения, CVS требует, чтобы исходные тексты были синхронизированы со всеми изменениями, которые сделали остальные члены группы. Команда `cvs update` позаботится об этом:

```
$ cvs update
cvs update: Updating .
U Makefile
RCS file: /u/src/master/httpc/httpc.c,v
retrieving revision 1.6
retrieving revision 1.7
Merging differences between 1.6 and 1.7 into httpc.c
M httpc.c
$
```

Рассмотрим пример строка за строкой:

‘`U Makefile`’
: Строка вида ‘U *файл*’ означает, что файл просто был обновлен; кто-то еще внес в этот файл изменения, и CVS скопировал измененный файл в ваш рабочий каталог.

‘`RCS file:...`
`retrieving revision 1.6`
`retrieving revision 1.7`
`Merging differences between 1.6 and 1.7 into httpc.c`’
: Это сообщение означает, что кто-то еще изменил ‘`httpc.c`’; CVS объединила их изменения с вашими и не обнаружила текстуальных конфликтов.

Цифры ‘1.6’ и ‘1.7’ — это номера редакции (revision numbers), используемые для обозначения определенных точек истории файла.

Заметьте, что CVS объединяет изменения только в вашей рабочей копии; репозиторий и рабочие каталоги других разработчиков остаются нетронутыми. От вас требуется протестировать объединенный текст и убедиться, что он верен.

‘M httpc.c’

Строка вида ‘M *файл*’ означает, что файл был модифицирован вами и содержит изменения, которые еще не стали видны другим разработчикам. Это – изменения, которые вам следует зафиксировать.

Так как CVS объединил чьи-то еще изменения с вашими исходными текстами, следует убедиться, что они все еще работают:

```
$ make
gcc -g -Wall -lnsl -lsocket httpc.c  -o httpc
$ httpc GET http://www.cyclic.com
... здесь находится текст домашней страницы Cyclic Software. . .
$
```

2.5 Фиксирование изменений

Теперь, когда вы синхронизировали свои исходники с коллегами и протестировали их, вы готовы поместить свои изменения в репозиторий и сделать их видимыми остальным разработчикам. Единственный файл, который вы изменили — это ‘httpc.c’, но в любом случае можно без опаски запустить `cvs update`, чтобы получить от CVS список модифицированных файлов:

```
$ cvs update
cvs update: Updating .
M httpc.c
$
```

Как и ожидалось, единственный файл, который упоминает CVS — это ‘httpc.c’; CVS говорит, что этот файл содержит изменения, которые еще не были зафиксированы. Вы можете зафиксировать их так:

```
$cvs commit httpc.c
```

В этом месте CVS запустит ваш любимый текстовый редактор и попросит вас ввести описание изменений. После того, как вы выйдете из редактора, CVS зафиксирует ваши изменения:

```
Checking in httpc.c;
/u/src/master/httpc/httpc.c,v  <- httpc.c
new revision: 1.8; previous revision: 1.7
$
```

Заметьте, что теперь вы зафиксировали ваши изменения и они видны всем остальным членам группы. Когда другой разработчик исполняет `cvs update`, CVS внесет ваши изменения в файлы в его рабочем каталоге.

2.6 Отслеживание изменений

Теперь вы, возможно, захотите узнать, какие именно изменения внес другой разработчик в файл ‘`httpc.c`’. Чтобы увидеть журнальные записи для данного файла, можно использовать команду `cvs log`:

```
$ cvs log httpc.c

RCS file: /usr/src/master/httpc/httpc.c,v
Working file: httpc.c
head: 1.8
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 8; selected revisions: 8
description:
The one and only source file for trivial HTTP client
----------------------
revision 1.8
date: 1996/10/31 20:11:14;  author: jimb;  state: Exp;  lines: +1 -1
(tcp_connection): Cast address stucture when calling connect.
----------------------
revision 1.7
date: 1996/10/31 19:18:45;  author: fred;  state: Exp;  lines: +6 -2
(match_header): Make this test case-insensitive.
----------------------
revision 1.6
date: 1996/10/31 19:15:23;  author: jimb;  state: Exp;  lines: +2 -6
...
$
```

Большую часть текста здесь вы можете игнорировать; следует только обратить внимание на серию журнальных записей после первой строки черточек. Журнальные записи выводятся на экран в обратном хронологическом порядке, исходя из предположения, что недавние изменения обычно более интересны. Каждая запись описывает одно изменение в файле, и может быть разобрано на составные части так:

‘`revision 1.8`’
: Каждая версия файла имеет уникальный *номер редакции*. Номера редакции выглядят как ‘`1.1`’, ‘`1.2`’, ‘`1.3.2.2`’ или даже ‘`1.3.2.2.4.5`’. По умолчанию номер 1.1 — это первая редакция файла. Каждое следующее редактирование увеличивает последнюю цифру на единицу.

‘`date: 1996/10/31 20:11:14; author: jimb; ...`’
: В этой строке находится дата изменения и имя пользователя, зафиксировавшего это изменение; остаток строки не очень интересен.

‘`(tcp_connection`: Cast...’
: Это, очевидно, описание изменения.

Команда `cvs log` может выбирать журнальные записи по дате или по номеру редакции; за описанием деталей обращайтесь к руководству.

Если вы хотите взглянуть на соответствующее изменение, то можете использовать команду `cvs diff`. Например, если вы хотите увидеть, какие изменения Фред зафиксировал в качестве редакции 1.7, используйте такую команду:

```
$ cvs diff -c -r 1.6 -r 1.7 httpc.c
```

Перед рассмотрением того, что нам выдала эта команда, опишем, что означает каждая ее часть.

`-c` Задает использование удобочитаемого формата выдачи изменений. (Интересно, почему это не так по умолчанию)[1].

`-r 1.6 -r 1.7`
Указывает CVS, что необходимо выдать изменения, необходимые, чтобы превратить редакцию 1.6 в редакцию 1.7. Вы можете запросить более широкий диапазон изменений; например, `-r 1.6 -r 1.8` отобразит изменение, сделанные Фредом, и изменения, сделанные вами чуть позже. (Вы также можете заказать выдачу изменений в обратном порядке — как будто бы они были отменены — указав номера редакций в обратном порядке: `-r 1.7 -r 1.6`. Это звучит странно, но иногда полезно.)

`httpc.c` Имя файла для обработки. Если вы не укажете его, CVS выдаст отчет обо всем каталоге.

Вот что выдаст эта команда:

```
Index: httpc.c
===================================================================
RCS file: /u/src/master/httpc/httcp.c,v
retrieving revision 1.6
retrieving revision 1.7
diff -c -r1.6 -r1.7
*** httpc.c     1996/10/31 19:15:23     1.6
-- httpc.c     1996/10/31 19:18:45     1.7
***************
*** 62,68 ****
  }


! /* Return non-zero iff HEADER is a prefix of TEXT.  HEADER should be
     null-terminated; LEN is the length of TEXT.  */
  static int
  match_header (char *header, char *text, size_t len)
-- 62,69 ---
  }


! /* Return non-zero iff HEADER is a prefix of TEXT, ignoring
!    differences in case.  HEADER should be lower-case, and
```

[1] Интересно, почему это вообще не `-u`? *Прим. перев.*

```
     null-terminated; LEN is the length of TEXT.  */
  static int
  match_header (char *header, char *text, size_t len)
***************
*** 76,81 ****
-- 77,84 ---
    for (i = 0; i < header_len; i++)
      {
        char t = text[i];
+       if ('A' <= t && t <= 'Z')
+         t += 'a' - 'A';
        if (header[i] != t)
          return 0;
      }
$
```

Требуются некоторые усилия, чтобы привыкнуть к такой подаче информации, но это определенно стоит того[2].

Интересная часть информации начинается с первых двух строк, начинающихся с `***` и `---`; они описывают старый и новый файлы, подлежащие сравнению. Остальное состоит из двух *ломтей* (hunk), каждый из которых начинается со строки из звездочек. Вот первый "ломоть":

```
***************
*** 62,68 ****
  }


! /* Return non-zero iff HEADER is a prefix of TEXT.  HEADER should be
     null-terminated; LEN is the length of TEXT.  */
  static int
  match_header (char *header, char *text, size_t len)
-- 62,69 ---
  }


! /* Return non-zero iff HEADER is a prefix of TEXT, ignoring
!    differences in case.  HEADER should be lower-case, and
     null-terminated; LEN is the length of TEXT.  */
  static int
  match_header (char *header, char *text, size_t len)
```

Текст из более старой редакции находится после строки `*** 62,68 ***`; текст новой редакции находится после строки `--- 62,69 ---`. Пара цифр означает показанный промежуток строк. CVS показывает контекст вокруг изменений и отмечает измененные строки символами '`!`'. Таким образом, вы видите, что одна строка из верхней половины была заменена на две строки из нижней.

Вот второй "ломоть":

[2] Все же лучше использовать `-u`. *Прим. перев.*

```
***************
*** 76,81 ****
-- 77,84 ---
    for (i = 0; i < header_len; i++)
      {
        char t = text[i];
+       if ('A' <= t && t <= 'Z')
+         t += 'a' - 'A';
        if (header[i] != t)
          return 0;
      }
```

Здесь описывается добавление двух строк, что обозначается символами '+'. CVS не выводит старый текст — это было бы избыточно. Для описания удаленных строк используется подобный формат.

Как и вывод команды `diff`, вывод команды `cvs diff` обычно называется *заплатой* (patch), потому что разработчики традиционно использовали этот формат для распространения исправлений и небольших новый возможностей. Заплата достаточно читабельна и содержит достаточно информации, чтобы применить изменения, которые она содержит, к текстовому файлу. В действительности, команда `patch` в среде UNIX делает с заплатами именно это.

2.7 Добавление и удаление файлов

CVS обращается с добавлением и удалением файлов так же, как и с прочими изменениями, записывая такие события в истории файлов. Можно смотреть на это так, как будто CVS сохраняет историю каталогов вместе с историей файлов.

CVS не считает, что созданные файлы должны оказаться под его контролем; это не так во многих случаях. Например, не требуется записывать историю изменений объектных и выполняемых файлов, потому что их содержимое всегда может быть воссоздано из исходных файлов (надо надеяться). Вместо этого, когда вы создадите новый файл, `cvs update` маркирует этот файл флагом '?', пока вы не скажете CVS, что именно вы намереваетесь сделать с этим файлом.

Чтобы добавить файл в проект, сначала вы должны создать его, затем использовать команду `cvs add`, чтобы маркировать его как добавленный. Затем при следующем выполнении команды `cvs commit` CVS добавит этот файл в репозиторий. Например, вот так можно добавить файл README в проект httpc:

```
$ ls
CVS     Makefile        httpc.c         poll-server
$ vi README
...введите описание httpc...
$ ls
CVS     Makefile        README          httpc.c         poll-server
$ cvs update
cvs update: Updating .
? README — CVS еще не знает об этом файле
$ cvs add README
cvs add: scheduling file 'README' for addition
```

```
cvs add: use 'cvs commit' to add this file permanently
$ cvs update — что же теперь думает CVS?
cvs update: Updating .
A README — Файл помечен как добавленный
$ cvs commit README
... CVS просит вас ввести журнальную запись...
RCS file: /u/jimb/cvs-class/rep/httpc/README,v
done
Checking in README;
/u/src/master/httpc/README,v  <- README
initial revision: 1.1
done
$
```

CVS обращается с удаленными файлами почти так же. Если вы удалите файл и выполните '`cvs update`', CVS не считает, что вы намереваетесь удалить файл из проекта. Вместо этого он поступает милосерднее — он восстанавливает последнюю сохраненную в репозитории версию файла и маркирует его флагом `U`, точно так же, как и любой другое обновление. (Это означает, что если вы хотите отменить изменения файла в рабочем каталоге, вы можете просто удалить его и позволить команде '`cvs update`' создать его заново.)

Чтобы удалить файл из проекта, вы должны сначала удалить его, а затем использовать команду '`cvs rm`', чтобы пометить его для удаления. При следующем запуске команда '`cvs commit`' удалит файл из репозитория.

Фиксирование файла, маркированного с помощью '`cvs rm`' не уничтожает историю этого файла — к ней просто добавляется еще одна редакция — "не существует". В репозитории по прежнему хранятся все записи об этом файле, и к ним можно обращаться по желанию — например, с помощью '`cvs diff`' или '`cvs log`'.

Для переименования файла существует несколько стратегий; самая простая — переименовать файл в рабочем каталоге, затем выполнить '`cvs rm`' со старым именем и '`cvs add`' с новым. Недостаток этого подхода в том, что журнальные записи о содержимом старого файла не переносятся в новый файл. Другие стратегии позволяют избежать этого, но зато доставляют другие, более странные проблемы.

Вы можете добавлять каталоги точно так же, как и обычные файлы.

2.8 Написание хороших журнальных записей

Если можно использовать '`cvs diff`', чтобы получить точное содержание любого изменения, то зачем тогда придумывать еще журнальную запись о нем? Очевидно, что журнальные записи короче, чем тексты изменений, и позволяют читателю получить общее понимание изменения без необходимости углубляться в детали.

Однако же, хорошая запись в журнале описывает *причину*, по которой было сделано изменение. Например, плохая журнальная запись для редакции 1.7 может звучать как "Преобразовать '`t`' к нижнему регистру". Это правильно, но бесполезно — '`cvs diff`' предоставляет точно ту же информацию, и гораздо яснее. Гораздо лучшей журнальной записью было бы "Сделать эту проверку независящей от регистра", потому что это гораздо яснее описывает причину любому, кто понимает, что происходит в коде —

клиенты HTTP должны игнорировать регистр букв при анализе заголовков ответа от сервера.

2.9 Обработка конфликтов

Как уже упоминалось, команда ‘`cvs update`’ объединяет изменения, сделанные другими разработчиками, с исходными текстами в вашем рабочем каталоге. Если вы отредактировали файл одновременно с кем-то другим, CVS объединит ваши изменения.

Довольно легко представить себе, как работает объединение, если изменения были совершены в разных участках файла, но что если вы оба изменили одну и ту же строку? CVS называет эту ситуацию *конфликтом* и предоставляет вам самому разобраться с ним.

Например, предположим, что вы добавили некую проверку на ошибки в код, определяющий адрес сервера. Перед фиксированием изменений вы должны запустить ‘`cvs update`’, чтобы синхронизировать ваш рабочий каталог с репозиторием:

```
$ cvs update
cvs update: Updating .
RCS file: /u/src/master/httpc/httpc.c,v
retrieving revision 1.8
retrieving revision 1.9
Merging differences between 1.8 and 1.9 into httpc.c
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in httpc.c
C httpc.c
$
```

В этом случае другой разработчик изменил тот же участок файла, что и вы, поэтому CVS жалуется на конфликт. Вместо того, чтобы напечатать `M httpc.c`, как это обычно происходит, CVS печатает `C httpc.c`, что означает наличие конфликта в этом файле.

Чтобы справиться с конфликтом, откройте этот файл в редакторе. CVS обозначает конфликтующий текст так:

```
    /* Look up the IP address of the host. */
    host_info = gethostbyname (hostname);
<<<<<<< httpc.c
    if (! host_info)
      {
        fprintf(stderr, "%s: host not found: %s\n", progname, hostname);
        exit(1);
      }
=======
    if (! host_info)
      {
        printf("httpc: no host");
        exit(1);
      }
>>>>>>> 1.9
```

```
sock = socket (PF_INET, SOCK_STREAM, 0);
```

Текст вашего рабочего файла появляется сверху, после символов `<<<`. Внизу находится конфликтующий код другого разработчика. Номер редакции `1.9` указывает, что конфликтующее изменение было внесено в версии 1.9 этого файла, упрощая проверку журнальных записей или просто выяснение изменений с помощью '`cvs diff`'.

Когда вы решите, как именно справиться с конфликтом, уберите маркеры из кода и отредактируйте его. В этом случае, так как ваша обработка ошибок определенно лучше, то просто отбросьте чужой вариант, оставив такой код:

```
/* Look up the IP address of the host. */
host_info = gethostbyname (hostname);
if (! host_info)
  {
    fprintf(stderr, "%s: host not found: %s\n", progname, hostname);
    exit(1);
  }

sock = socket (PF_INET, SOCK_STREAM, 0);
```

Теперь протестируйте изменения и зафиксируйте их:

```
$ make
gcc -g -Wall -Wmissing-prototypes -lnsl -lsocket  httpc.c  -o httpc
$ httpc GET http://www.cyclic.com
HTTP/1.0 200 Document follows
Date: Thu, 31 Oct 1996 23:04:06 GMT
...
$ httpc GET http://www.frobnitz.com
httpc: host not found: www.frobnitz.com
$ cvs commit httpc.c
```

Важно понимать, что именно CVS считает конфликтом. CVS не понимает семантики вашей программы, он обращается с исходным кодом просто как с деревом текстовых файлов. Если один разработчик добавляет новый аргумент в функцию и исправляет все ее вызовы, пока другой разработчик одновременно добавляет новый вызов этой функции, и не передает ей этот новый аргумент, что определенно является конфликтом — два изменения несовместимы — но CVS не сообщит об этом. Его понимание конфликтов строго текстуально.

На практике, однако, конфликты случаются редко. Обычно они происходят потому, что два человека пытаются справиться с одной и той же проблемой, от недостатка взаимодействия между разработчиками, или от разногласий по поводу архитектуры программы. Правильное распределение задач между разработчиками уменьшает вероятность конфликтов.

Многие системы контроля версий позволяют разработчику *блокировать* файл, предотвращая внесение в него изменений до тех пор, пока его собственные изменения не будут зафиксированы. Блокировки уместны в некоторых ситуациях, но их использование не всегда лучше, чем использование CVS — без блокировок. Изменения обычно объединяются без проблем, а разработчики иногда забывают убрать блокировку, в обоих случаях явное блокирование приводит к ненужным задержкам.

Более того, блокировки предотвращают только текстуальные конфликты — они ничего не могут поделать с семантическими конфликтами типа вышеописанного — когда два разработчика редактируют разные файлы.

Оглавление