Linux 開発アプリケーションゼミ

ゼミ担当者 ： 荒久田 博士，折戸 俊彦，市川 親司
指導院生 ： 片浦 哲平，斉藤 宏樹，谷口 義樹
開催日 ： 2003 年 5 月 25 日

1 はじめに

三木研では，CVS というシステムによりファイルのバージョン管理を行っている．CVS が利用されている理由は，次のとおりである．また，CVS によるファイルのバージョン管理を図で示したものが Fig. 1 である．

- バックアップ
  複数のマシンにファイルのバックアップを取っておくことにより，予期せぬハードディスククラッシュで手元のワークスペースのファイルが無くなったとしても，リポジトリさえ無事であれば，簡単にファイルが復活できる．例えば，duke のリポジトリに自分のファイルを置いておくことにより，自分のマシンが故障によりファイルが失われてしまった際，duke に置いてあるファイルがバックアップとして残る．リポジトリとは，ファイルの変更履歴を保管している保管庫のようなものである．手元で作業するファイルは，この保管庫のコピーとなっている．

  ファイルを forte や cambria でチェックアウトしておくことで，それらのマシンにもバックアップされることになる．duke にあるファイルは定期的に museion いよりバックアップが取られている．そのため安全性が向上する．

- 複数人による開発と変更衝突
  複数人で共有ファイルの変更作業をしていると，変更衝突が起こる可能性がある．
  変更衝突（conflict）とは，ファイルの変更が複数人によって同時に行われ，それにより変更箇所が重なってしまう状態のことをいう．CVS では変更衝突が起こった場合，勝手にファイルを上書きしてしまうことなく，衝突が起こったことを作業者に知らせる（エラーメッセージを出す）．

- リポジトリとワークスペース
  CVS で主流になっている考え方は,「コピー/変更/マージ」という並列開発の形態である．
  変更衝突が起こった場合には，あとから登録作業をした方の責任において衝突回避作業，つまりマージを行う．これは要するに，リポジトリへ「より速く登録作業を行った方がより優先される」というルールだといえる．
  この手法では，ファイル変更履歴の保管場所のリポジトリと，作業者個人個人のワークスペースとに分けるという所が優れている．作業者は，自分専用のワークスペースにリポジトリから編集したいファイルのコピーを所得する．編集作業は，手元のコピーに対して行い，そのあとで，編集作業結果をリポジトリに登録する手続きを取る．

2 CVS とは

2.1 CVS

CVS（Concurrent Versioning System）とは，分散環境でのプログラム開発においてソースコードのバージョンを管理するツールである．
バージョン管理とはファイルや各種のデータなどの内容修正や変更が行われたとき，その修正および変更履歴を管理することである．計算機システムを利用してバージョン管理を行うシステムをバージョン管理システムと呼ぶ．

2.2 CVS の特徴

CVS は次のような特徴を持っている．

- バグを見つけ出すときに役立つ．
  ソフトウェアのバージョンが新しくなり，その際にある機能が動かなくなった場合，新旧でどのような変更が加えられたかがわかれば，実際にプログラミングを動作させなくてもバグを見つけ出せる可能性がある．

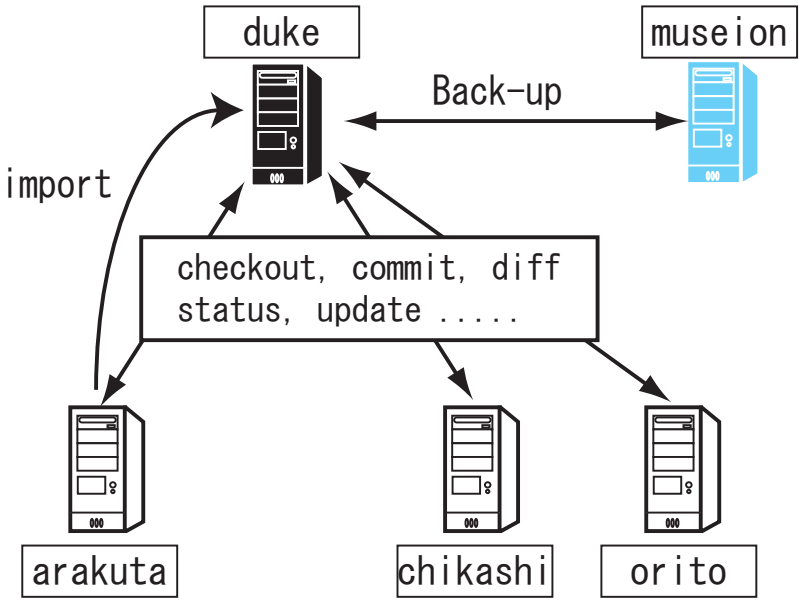

Fig. 1　CVS 管理のファイルの伝播

- バッチを簡単に作成する仕組みを提供する.
  ソフトウェアのリリース版を出した後，少しの変更で済むマイナーなバグを発見した時には,「パッチ」という形で差分（修正部分）だけを送れば，新たに修正したリリース版をダウンロードするような必要がない.
- 以前のバージョンを簡単に戻すことを可能とする.
  プログラムを速くするために加えたはずの変更が，反対に遅くなる要因になってしまった場合は，簡単にロールバックできる.
- 変更の統合を簡単にする.
  ある優れたユーザが新機能を追加し，その差分を送ってきた時に簡単に統合することができる.
- グループ作業を簡単にする.
  複数の人が同時並行的に開発を進めていく上で，同じソースコードを常に持っていれば，差分を統合することは簡単である.

3　CVS のインストールと準備

3.1　CVS のインストール (Debian)

CVS のインストールは次のコマンドを実行すればよい.

```
# apt-get install cvs
```

インストールの際，まずリポジトリの場所を聞かれる (デフォルトでは，/var/lib/cvs となっている)．しかし今回は，リポジトリの場所は duke 内であり，自分のマシン内に作成する必要はないので，次の項目で ignore を選択すればよい．その後の設定は，デフォルトのままでよい.

3.2　CVS のインストール (cygwin)

Windows で CVS を利用するには，cygwin 版の CVS パッケージをインストールすればよい.

```
1. cygwin の setup.exe を起動する.
2. パッケージの選択画面から，cvs を選択し，インストールする.
```

3.3 CVS_RSH の設定

CVS では，pserver という独自のプロトコルを用いてリポジトリを共有できるしくみがある．これは，ネットワーク的に離れたメンバー間のリポジトリ共有のために用いられる．CVS を使うにあたって，三木研では，duke にリポジトリを作成する．そこで duke への接続が必要となるが，デフォルトでは rsh で接続しようとし，接続ができないので，SSH 経由で接続を行うように設定する必要がある．

```
$ export CVS_RSH=ssh
```

4 CVS の基本的な使い方

4.1 import

import とは，ファイルを CVS 管理におくことである．ここでは，/home/arakuta にある semi デリレクトリ中のファイルを duke 内にあるリポジトリに登録する．なお，バイナリファイル (実行ファイル) は CVS 管理に置かない方がよいので，import の前には make clean を実行しておくことが望ましい．
:ext とは，外部にリポジトリがあるとき (三木研では duke)，そのリポジトリの場所を示す場合に必要となるメソッドである．

```
$ cd semi
$ cvs -d:ext:arakuta@duke.doshisha.ac.jp:/home/cvs import personal/arakuta/semi arakuta first
```

このコマンドの記述において，arakuta の部分はベンダータグ，最後の first の部分はリリースタグと呼ばれるものである．三木研では使わないので気にしなくても良いが，import の際に省略することはできないので，なんらかの文字を入れる必要がある．

なお，.bashrc ファイルへ以下の記述をしておくことでコマンドの簡略化を行うことができる．

```
export CVS_RSH=ssh
export :ext:arakuta@duke.doshisha.ac.jp:/home/cvs
```

簡略化が行われる前後のコマンドを次に示す．

```
(.bashrc への記述前)
cvs -d:ext:arakuta@duke.doshisha.ac.jp:/home/cvs import personal/arakuta/semi arakuta first

(.bashrc への記述後)
cvs import personal/arakuta/semi arakuta first
```

Windows 上で cygwin により CVS を使うときも，同じコマンドで利用することが出来る．以下の記述を.bash_profile へ記述しておくことで Linux における CVS の利用と同様，コマンドの簡略化を行うことが出来る．

```
eval ‘ssh-agent ‘
ssh-add ~/.ssh/id_dsa
export CVS_RSH=ssh
export :ext:arakuta@duke.doshisha.ac.jp:/home/cvs
```

コマンド実行後，自動的に vi[1]が起動され，コメントを求められる．import するファイルに関するなんらかのコメントを書き，エディタを終了させる．

```
wrote by arakuta
CVS: ----------------------------------------------------------------------
CVS: Enter Log.  Lines beginning with ‘CVS:’ are removed automatically
CVS:
CVS: ----------------------------------------------------------------------
```

[1]環境変数 CVSEDITOR によって指定可能

なおインポートの際に，-m オプションを利用することで，コメントを予め指定すれば vi が起動されてコメントを求められることがない．正常にインポートが完了すれば，以下のようなメッセージが出力される．出力メッセージの先頭の文字 N は，このファイルが新規ファイルであることを意味している．

```
N personal/arakuta/semi/cvs.tex
N personal/arakuta/semi/cvs.eps
N personal/arakuta/semi/cvs_day.eps

No conflicts created by this import
```

4.2 checkout

checkout とは，CVS 管理されたデータを取り出すことである．checkout を行うと，CVS の管理情報が出力される．

```
$ cvs checkout -d codir personal/arakuta/semi
```

実行すると，以下のようなメッセージが出力される．先頭の U の文字は，リポジトリと一致するよう，ファイルが更新されたことを意味している．

```
U semi/cvs.tex
U semi/cvs.eps
U semi/cvs_day.eps
```

4.3 diff

diff コマンドはファイルの更新前と更新後の比較を行うさいに用いる．diff コマンドの実行方法は，以下のとおりである．

```
$ cvs diff cvs.tex
```

以下に具体的な実行例を示す．cvs.tex に，以下のような記述がされているものとする．

```
          :
          :
例えば，mikilab のリポジトリに自分のファイルを置いておくことにより，自分のマシンが故障によりファイル
が失われてしまった際，mikilab に置いてあるファイルがバックアップとして残る．
          :
          :
```

cvs.tex 中の mikilab という記述を以下のように duke へ書き換えたとする．

```
          :
          :
例えば，duke のリポジトリに自分のファイルを置いておくことにより，自分のマシンが故障によりファイルが失
われてしまった際，duke に置いてあるファイルがバックアップとして残る．
          :
          :
```

このときファイルの比較機能 diff を使い，変更箇所を確認することができる．次のようにコマンドを実行すれば，結果が出力される．

```
arakuta@arakuta:~/cvs/linux_semi$ cvs diff cvs.tex
Index: cvs.tex
===================================================================
RCS file: /home/cvs/personal/arakuta/semi/cvs.tex,v
retrieving revision 1.4
diff -r1.4 cvs.tex
56c56
< 例えば，mikilab のリポジトリに自分のファイルを置いておくことにより，自分のマシ ンが故障によりファイルが失われてしまった際，mikilab に置いてあるファイルがバック アップとして残る．
---
> 例えば，duke のリポジトリに自分のファイルを置いておくことにより，自分のマシンが故障によりファイルが失われてしまった際，duke に置いてあるファイルがバックアップとして残る．
```

この出力を例に，diff の出力の読み方を説明する．”56c56”は，元のファイルの 56 行目が，新しいファイルの 56 行目に変更されたことを表している．その下の < で始まる行が変更前，> で始まる行が変更後のファイルの内容である．

この形式は読みにくいので，”context diff” と呼ばれる形式が好まれる．この形式は，変更された行とその前後を含めて表示するものである．cvs diff に-u オプションを付ければ実行できる．実行結果を以下に示す．

```
arakuta@arakuta:~/cvs/linux_semi$ cvs diff -u cvs.tex
Index: cvs.tex
===================================================================
RCS file: /home/cvs/personal/arakuta/semi/cvs.tex,v
retrieving revision 1.4
diff -u -r1.4 cvs.tex
--- cvs.tex     22 May 2003 09:06:07 -0000      1.4
+++ cvs.tex     22 May 2003 09:07:58 -0000
@@ -53,7 +53,7 @@
\verb|\end{center}|
\verb|\end{figure}|

-例えば，mikilab のリポジトリに自分のファイルを置いておくことにより，自分のマシンが故障によりファイルが失われてしまった際，mikilab に置いてあるファイルがバックア ップとして残る．
+例えば，duke のリポジトリに自分のファイルを置いておくことにより，自分のマシンが 故障によりファイルが失われてしまった際，duke に置いてあるファイルがバックアップとして残る．

\verb|\item 複数人による開発と変更衝突|
```

4.4 commit

commit とは，変更されたファイルを新しいリビジョンとしてリポジトリに登録する作業のことである．注意点は，ファイルを登録する前に，そのファイルが登録する価値があるものか，中途半端な状態のファイルではないかということを確認することである．commit の実行の仕方は次のとおりである．commit のさいには，cvs import と同様にログメッセージの入力が必要である．

```
$ cvs commit -m "wrote by arakuta" cvs.tex
Checking in cvs.tex;
/home/arakuta/semi/cvs.tex,v <-- cvs.tex
new revision: 1.4; previous revision: 1.3
done
```

この実行結果では,「リビジョン 1.3 が 1.4 に変更された」というメッセージが表示されている．commit すると作業ファイルは自動的に新しいリビジョンになる．作業ディレクトリの中の複数のファイルの変更をすべて commit する場合は，ファイル名の指定を省略して以下のように実行する．

```
$ cvs commit
```

4.5 status

status は，現在の作業ディレクトリにあるファイルとリポジトリにあるファイルとの関係を表示するコマンドである．status の実行は次のとおりである．ファイル名を指定することで，特定のファイルの状態だけを表示することも可能である．

```
$ cvs status
```

このコマンドを実行すると，次のような出力がされる．

```
===================================================================
File: cvs.tex           Status: Needs Patch

   Working revision:    1.4     Fri May 20 09:44:06 2003
   Repository revision: 1.5     /home/cvs/personal/arakuta/semi/cvs.tex,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

この出力は，cvs.tex はリポジトリにあるファイルの方が作業ディレクトリにあるファイルよりも新しくなっていることを示している．そのため，update を実行して最新のファイルをとってくる必要がある．

4.6 update

update は，リポジトリ上にある最新のリビジョンを自分の作業ディレクトリに持ってくるためのコマンドである．次のようにコマンドを実行することで update は実行される．status 同様，特定のファイルだけを更新することも可能である．

```
$ cvs update
```

その結果，次のような出力が得られる．この中で，`U cvs.tex` というのは，リポジトリにある最新リビジョンのものが作業ディレクトリ上のものに反映されたということを示している．他の出力を，Table 1 に示す．

```
cvs update: Updating .
U cvs.tex
```

Table 1　update 出力の意味

記号	意味
U	リポジトリの最新リビジョンがコピーされたファイル
A	リポジトリに新しく追加されるファイル（commit が必要）
R	リポジトリから削除されるファイル（commit が必要）
M	リポジトリにあるファイルと作業ディレクトリで加えた変更がマージされたファイル
C	リポジトリにあるファイルとのマージ中にコンフリクトが生じたファイル

リポジトリにあるファイルから変更が加えられ，前リビジョンのままではないファイルでも，CVS は最新リビジョンと作業ディレクトリにあるファイルをマージすることによって，適切な状態にしてくれる．行われた変更が正しいものであるかを確認した後，リポジトリに commit するのは，利用者の責任となる．

ただし，場合によっては CVS が自動的に更新状況を反映できないこともある．そのような場合は，ファイル名の前に C がつき，コンフリクトが生じたことが報告される．コンフリクトが生じると，次のようなファイルが生成される．

```
This is code of basic semi's resume.
<<<<<<< cvs.tex
This source is linux semi's.
=======
This file is source of linux semi.
>>>>>>> 1.3
```

<<<<<<<から=======までの部分が作業ディレクトリ上での変更点であり，=======から>>>>>>>の部分がリポジトリから来た変更点である．このような場合，コンフリクトが生じている部分を取り除き，ファイルを適切な状態にし，commit を行わなければならない．

4.7 annotate

annotate は各リビジョンで誰がどのような変更を行ったかを表示することができるコマンドである．annotate の実行には次のとおりである．ファイル名の指定により，特定ファイルの更新履歴のみ表示することもできる．

```
$ cvs annotate
```

この結果，次のような出力が得られる．これを見れば，誰がいつどのような変更を行ったかが把握できる．

```
Annotations for readme
***************
1.2          (arakuta   03-May-19): This is tex file.
1.3          (arakuta   03-May-20): This source need to compile.
1.5          (arakuta   03-May-20): This source is basic semi's resume.
```

4.8 add

リポジトリに新たにファイルを追加したいという場合は，add を使用する．add コマンドの使い方は次のとおりである．

```
$ cvs add newIMG.eps
```

そうすると，次のような出力がされる．

```
cvs add: scheduling file `newIMG.eps' for addition
cvs add: use 'cvs commit' to add this file permanently
```

これにより，リポジトリに新しいファイルが登録される．出力にあるように，実際にファイルがリポジトリに置かれるためには，commit を実行する必要がある．

```
$ cvs commit -m "Add File"
cvs commit: Examining .
RCS file: /home/arakuta/semi/newIMG.eps,v
done
Checking in newIMG.eps;
/home/arakuta/semi/newIMG.eps,v  <--  newIMG.eps
initial revision: 1.1
done
```

4.9 remove

リポジトリからファイルを削除するときは，remove を使用する．remove を実行する場合はまず，作業ディレクトリのファイルを削除してから行うようにしなければいけない．remove の使い方は次のとおりである．

```
$ rm readme
$ cvs remove readme
```

コマンドの実行により，画面には次のように出力される．

```
cvs remove: scheduling ‘readme’ for removal
cvs remove: use ’cvs commit’ to remove this file permanently
```

これで，リポジトリからファイルを削除する準備が整う．実際にファイルを削除するには，add 同様，commit を実行する必要がある．

```
$ cvs commit -m "Delete readme file."
cvs commit: Examining .
Removing readme;
/home/arakuta/semi/readme,v  <--  readme
new revision: delete; previous revision: 1.3
done
```

4.10 tag

ある時点でのリポジトリの状態を記録しておきたい場合，tag コマンドを使い，その状態にタグ名を付けることで記録を行う．tag コマンドの利用は，次のとおりである．タグ名に使用できる文字は，最初が英文字でその後ろは，英数字，ハイフン，アンダースコアとなっている．

```
$ cvs tag tag_20030523
```

この結果，次のような出力がされる．

```
cvs server: Tagging .
T CVS.eps
T cvs.tex
T readme
```

これにより，リポジトリ内の特定リビジョンを参照するためのタグが作成された．このタグを利用するには，各コマンドで，“-r タグ名” というオプションを利用する．たとえば，作成したタグ名を利用して checkout を行いたい場合，次のコマンドを実行する．

```
$ cvs checkout -r tag_20030523 -d oldlog personal/arakuta/semi
```

status コマンドを実行すると，特定タグの状態が checkout されたことが確認できる．以下が status コマンドの実行画面である．

```
cvs server: Examining .
===================================================================
File: CVS.eps           Status: Up-to-date

   Working revision:    1.2
   Repository revision: 1.2     /home/cvs/personal/arakuta/semi/CVS.eps,v
   Sticky Tag:          tag_20030523 (revision: 1.2)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: cvs.tex           Status: Up-to-date

   Working revision:    1.10
   Repository revision: 1.10    /home/cvs/personal/arakuta/semi/cvs.tex,v
   Sticky Tag:          tag_20030523 (revision: 1.10)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: readme            Status: Up-to-date

   Working revision:    1.3
   Repository revision: 1.3     /home/cvs/personal/arakuta/semi/readme,v
   Sticky Tag:          tag_20030523 (revision: 1.3)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

5 CVS の応用

5.1 過去のリビジョンを取り出す方法

以下のようなファイル (hello.c) があったとする.

```
$ cvs annotate
Annotations for hello.c
***************
1.1          (arakuta  19-May-03): #include<stdio.h>
1.1          (arakuta  19-May-03):
1.1          (arakuta  20-May-03): int main()
1.1          (arakuta  20-May-03): {
1.1          (arakuta  20-May-03):  printf("Hello!\n");
1.2          (arakuta  20-May-03):  printf("Hello! revision1.2!\n");
1.3          (arakuta  21-May-03):  printf("Hello! revision1.3!\n");
1.1          (arakuta  21-May-03):
1.1          (arakuta  23-May-03):  return 0;
1.1          (arakuta  23-May-03): }
```

過去のリビジョン 1.2 を取り出したいときは，オプションをつけて update を行う．実行方法は以下のとおりである.

```
$ cvs update -r 1.2 -p hello.c > hello.c.1.2
===================================================================
Checking out hello.c
RCS:  /home/cvs/personal/arakuta/semi/hello.c,v
VERS: 1.2
***************
$ less hello.c.1.2
#include<stdio.h>

int main()
{
        printf("Hello!\n");
        printf("Hello! revision1.2!\n");

        return 0;
}
```

5.2 任意のリビジョンを付加する方法

CVS では，commit するたびにファイルのリビジョン番号は 1.1,1.2,1.3,... というふうに増える．また 1.9 の次は，2.0 ではなく 1.10,1.11 となる．

ここでは，リビジョンを 2.xx にする方法を紹介するが，通常用いられることは少ない．1.9999xx とどんどん増やしていくのが一般的である．

```
$ cvs commit -r 2.1 hello.c
```

こうすることで，以後 cvs commit を実行すると，2.2,2.3,.... とリビジョンが上がる．

5.3 ブランチ

CVS では，本流の開発の流れからさらに派生して開発を進めることができる．本流を木のトランク（幹）と呼び，支流をブランチ（枝）と呼ぶ．ここでは，ブランチの使い方について紹介する．

5.3.1 ブランチの作り方

ブランチの作成手順は，次のとおりである．まず tag -b でブランチタグを付加する．

```
$ cvs tag -b BRANCH_linuxsemi
cvs server: Tagging .
T CVS.eps
T cvs.tex
$ cvs status -v
cvs server: Examining .
===================================================================
File: CVS.eps           Status: Up-to-date

   Working revision:    1.2
   Repository revision: 1.2     /home/cvs/personal/arakuta/cvs/CVS.eps,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
        BRANCH_linuxsemi               (branch: 1.2.2)
        tag_20030523                   (revision: 1.2)
        first                          (revision: 1.1.1.1)
        arakuta                        (branch: 1.1.1)

===================================================================
File: cvs.tex           Status: Locally Modified

   Working revision:    1.12
   Repository revision: 1.12    /home/cvs/personal/arakuta/cvs/cvs.tex,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
        BRANCH_linuxsemi               (branch: 1.12.2)
        tag_20030523                   (revision: 1.10)
        first                          (revision: 1.1.1.1)
        arakuta                        (branch: 1.1.1)
```

このようにブランチタグを付加し，update -r と実行することで，自分の作業ファイルがトランクから離れ，ブランチの流れに入る．

```
arakuta@arakuta:~/cvs/linuxsemi$ cvs update -r BRANCH_linuxsemi
cvs server: Updating .
arakuta@arakuta:~/cvs/linuxsemi$ cvs status -v
cvs server: Examining .
===================================================================
File: CVS.eps           Status: Up-to-date

   Working revision:    1.2
   Repository revision: 1.2     /home/cvs/personal/arakuta/semi_cvs/CVS.eps,v
   Sticky Tag:          BRANCH_linuxsemi (branch: 1.2.2)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
        BRANCH_linuxsemi               (branch: 1.2.2)
        tag_20030523                   (revision: 1.2)
        log_20030523                   (revision: 1.2)
        first                          (revision: 1.1.1.1)
        arakuta                        (branch: 1.1.1)

===================================================================
File: cvs.tex           Status: Up-to-date

   Working revision:    1.12
   Repository revision: 1.12    /home/cvs/personal/arakuta/semi_cvs/cvs.tex,v
   Sticky Tag:          BRANCH_linuxsemi (branch: 1.12.2)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
        BRANCH_linuxsemi               (branch: 1.12.2)
        tag_20030523                   (revision: 1.10)
        log_20030523                   (revision: 1.8)
        first                          (revision: 1.1.1.1)
        arakuta                        (branch: 1.1.1)
```

このようにブランチの流れに入ると，なにか編集を加え，commit した場合，ブランチのバージョンは上がるが，トランクには影響を与えない．

```
$ emacs cvs.tex (cvs.tex の編集)
$ cvs commit -m "wrote at branch's cvs.tex"
cvs commit: Examining .
Checking in cvs.tex;
/home/cvs/personal/arakuta/semi/cvs.tex,v  <--  cvs.tex
new revision: 1.12.2.1; previous revision: 1.12
done
```

自分の作業ファイルをブランチからトランクの流れに戻したいときは，次のようにコマンドを実行する．

```
$ cvs update -A
cvs server: Updating .
P cvs.tex
$ cvs status
cvs server: Examining .
===================================================================
File: CVS.eps           Status: Up-to-date

   Working revision:    1.2
   Repository revision: 1.2     /home/cvs/personal/arakuta/semi/CVS.eps,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: cvs.tex           Status: Up-to-date

   Working revision:    1.14
   Repository revision: 1.14    /home/cvs/personal/arakuta/semi/cvs.tex,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

5.3.2 マージの方法

ブランチで作業する場合，いつかは修正内容を集約するためのマージ作業が必要となる．その方法として，まず以下の方法を示す．これにより，トランクのものをブランチにマージすることができる．

```
$ cvs update -j TAGNAME
```

次に，タグを用いる方法を以下に示す．まず，マージ元にタグをつける．

```
$ cvs tag TAG-BRANCH
cvs server: Tagging .
T cvs.tex
```

次に，マージ先にタグをつける．

```
$ cvs tag TAG-TRANK
cvs server: Tagging .
T cvs.tex
```

そして，マージを以下のように行う．

```
$ cvs update -j TAG-BRANCH
cvs server: Updating .
RCS file: /home/cvs/personal/arakuta/semi/cvs.tex,v
retrieving revision 1.1.1.1
retrieving revision 1.1.1.1.2.1
Merging differences between 1.1.1.1 and 1.1.1.1.2.1 into cvs.tex
```

6 Emacs からの CVS の利用

6.1 VC モード

Emacs でファイルを編集しているさい，C-x v v を押すと，扱っているファイルの状態によって，処理が行われる．それぞれの処理は次の Table 2 のとおりである．

Table 2　vc モード

状況	処理
新規のファイルである	CVS に登録する
CVS リポジトリの内容が変更されている	CVS リポジトリの内容の変更部分を自分のソースにマージする
CVS リポジトリの内容から変更がある場合	変更点を commit する

6.2　pcl-cvs モード

Emacs には pcl-cvs モードという作業ディレクトリを一まとめとして管理するためのモードがある．M-x cvs-update で CVS アップデートバッファになる．そこで Table 3 のコマンドが利用できる．

Table 3　pcl-cvs モード

キー	意味
g	再度 cvs update を行う
m	ファイルにマークをつける
c	“ M ”がついているファイルの変更を CVS レポジトリに反映（commit）する
i	cvsignore というファイルに追加して，cvs の管理から外す
a	“ ? ”がついているファイルを CVS の管理に登録

7　CVS チュートリアル

実際に順を追って CVS を使用していくことで，CVS の利用方法を簡単に説明する．

1. まず，リポジトリに登録するためのファイルを作成する．

```
arakuta@arakuta:~$ mkdir cvstest
arakuta@arakuta:~$ cd cvstest
arakuta@arakuta:~/cvstest$ vi sample.tex
arakuta@arakuta:~/cvstest$ vi readme.txt
```

2. 次に，作成したファイルをリポジトリに登録する．import の際，import を実行したディレクトリにあるファイル全てがリポジトリに登録されることになる．そのため不必要なファイルが登録されないように注意しなければならない．

```
arakuta@arakuta:~/cvstest$ cvs import personal/arakuta/test arakuta first
```

3. 今登録したファイルをリポジトリから取得する．

```
arakuta@arakuta:~/cvstest$ cd ..
arakuta@arakuta:~$ cvs checkout -d cvsdir personal/arakuta/test
```

4. ファイルを修正し，リポジトリに反映させる．

```
arakuta@arakuta:~$ cd cvsdir
arakuta@arakuta:~/cvsdir$ vi sample.tex
arakuta@arakuta:~/cvsdir$ cvs commit
```

5. duke のリポジトリからファイルを取得し，編集作業を行う．

```
arakuta@arakuta:~/cvsdir$ cvs checkout -d workdir personal/arakuta/test
arakuta@arakuta:~/cvsdir$ cd workdir
arakuta@arakuta:~/cvsdir/workdir$ vi sample.tex
```

6. 以降，自分のコンピュータでソースファイルの修正を行ってリポジトリに反映させ，duke ではリポジトリからの取得・編集を行う．

```
arakuta@arakuta:~/cvsdir/workdir$ vi sample.tex
arakuta@arakuta:~/cvsdir/workdir$ cvs diff -u sample.tex
arakuta@arakuta:~/cvsdir/workdir$ cvs commit -m sample.tex
arakuta@arakuta:~/cvsdir/workdir$ cvs status
arakuta@arakuta:~/cvsdir/workdir$ cvs update
```

参考文献

1) The CVS Book 日本語訳，http://kahori.com/j-cvsbook/

2) バージョン管理システム (CVS) の導入と活用，鯉江英隆 他，2000.