Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2585

INTRODUÇÃO AO USO DE CVS

(*Concurrent Version System*)

DÉBORA MARIA BARROSO PAIVA
DANIEL CARNIO JUNQUEIRA
RENATA PONTIN DE MATTOS FORTES

Nº 55

NOTAS DIDÁTICAS DO ICMC

São Carlos
SETEMBRO/2002

Índice

1- Introdução

O gerenciamento das versões dos artefatos obtidas durante, o desenvolvimento e a evolução de um produto de software, é uma atividade de suma importância quando envolve equipes e projetos de maior porte. Do ponto de vista de qualidade de software, de fato, é muito comum que sejam efetuadas mudanças para melhoria do *software* ou para correção de erros detectados durante e após a etapa de implementação. Diante da constante evolução dos sistemas atualmente é, portanto, imprescindível manter cópia de tudo o que é desenvolvido e modificado, incluindo código fonte, documentação, arquivos executáveis, etc.

O controle de versões pode ser definido como o *processo de organização, coordenação e gerenciamento de objetos em evolução* [Hicks *et al*. 1998]. Em muitas áreas de aplicação, o processo de evolução do objeto pode ser caracterizado como uma série de refinamentos incrementais. Por exemplo, desenvolvedores de *software* freqüentemente fazem mudanças em módulos de *software* quando erros são detectados, e os clientes raramente ficam satisfeitos com a primeira versão, o que implica na produção de várias revisões antes da obtenção de uma versão final. Em cada um desses casos, objetos em desenvolvimento são alterados e atualizados de forma a produzir o próximo refinamento no processo evolucionário.

Uma versão de um sistema (artefato) é uma instância do mesmo que difere, de algum modo, de outras instâncias [Sommerville, 1995; Vitali e Durand, 1999]. Cada versão de um arquivo tem um número de *revisão*, por exemplo, ‘1.1’, ‘1.2’ e ‘1.3.2.2’. Existem várias ferramentas automatizadas para o gerenciamento de versões de arquivos fontes, por exemplo, o SCCS (*Source Code Control System*) [Rochkind, 1975] e o RCS (*Revision Control System*) [Tichy, 1985] para desenvolvimento de software. Os objetivos principais desses sistemas são reduzir o espaço utilizado no armazenamento de múltiplas versões de software e controlar o trabalho cooperativo entre os desenvolvedores.

O *Concurrent Version System* (CVS) [CVS, 2002] é um sistema de gerenciamento de versões originalmente desenvolvido por Dick Grune em 1986. Inicialmente consistia de um conjunto de scripts shell do Unix. Em 1989 foi projetado e codificado na linguagem de programação *C* por Brian Berlinger e, mais tarde, Jeft Polk o ajudou com o projeto de módulos e suporte à geração de *branches* (ramificações) no CVS.

Dentre as funcionalidades básicas do CVS, pode-se citar [CVS, 2002]:

- Mantém um histórico de todas as alterações feitas nos diretórios que são gerenciados. Com base nesse histórico, o CVS pode mostrar a um desenvolvedor quando, porque e por quem uma alteração foi feita;
- Armazena versões em um repositório central que mantém as cópias mestres de todos os arquivos que estão sob o gerenciamento de versões;
- Recupera versões anteriores de arquivos eficientemente;
- Permite que grupos de desenvolvedores controlem arquivos através da rede de forma transparente;

- Suporta desenvolvimento paralelo, permitindo que mais de uma pessoa trabalhe em um mesmo arquivo ao mesmo tempo;
- Fornece acesso confiável aos diretórios a partir de máquinas hospedeiras (*hosts*) remotas usando protocolos Internet;
- Permite adicionar, remover e alterar arquivos e diretórios do repositório;
- Permite agrupar uma coleção de arquivos relacionados em módulos e, então, o módulo passa a ser gerenciado;
- Pode ser executado em várias plataformas, como, UNIX, Linux e demais sistemas POSIX, Windows 95, Windows NT, Macintosh e VMS.

O CVS armazena todas as informações sobre o controle de versões em arquivos contidos em uma hierarquia de diretórios, chamada *repositório*.

Este material tem como objetivo apresentar os conceitos importantes relacionados ao CVS e seus principais comandos, de maneira introdutória. A Seção seguinte descreve a estrutura do repositório CVS. As Seções 3 e 4 indicam como a instalação e a configuração do CVS devem ser feitas e a Seção 5 apresenta os principais comandos do sistema.

2- O Repositório CVS

Um conceito inerente ao controle de versões se refere ao *repositório*. Nele, o CVS armazena uma cópia completa de todos os arquivos e diretórios que estão sob controle de versões. Normalmente, o usuário nunca acessa os arquivos no repositório diretamente. Ele deve usar comandos do CVS para obter uma cópia dos arquivos em um diretório de trabalho (*checkout*) e, então, pode trabalhar nessa cópia. Ao terminar de fazer as mudanças, ele deve enviar novamente o arquivo para o repositório (*commit*). Dessa forma, o repositório contém as mudanças que foram feitas, registrando exatamente o que foi mudado, quando a mudança ocorreu e outras informações relacionadas a cada *commit* realizado. O repositório não é um subdiretório de um diretório de trabalho ou vice-versa; repositório e diretório de trabalho estão em locais diferentes.

Os arquivos administrativos do CVS ficam armazenados em *$CVSROOT/CVSROOT*[1]. Os outros diretórios que estão em *$CVSROOT* contêm os módulos definidos pelo usuário. Um exemplo de uma estrutura do repositório CVS é apresentado na Figura 2.1.

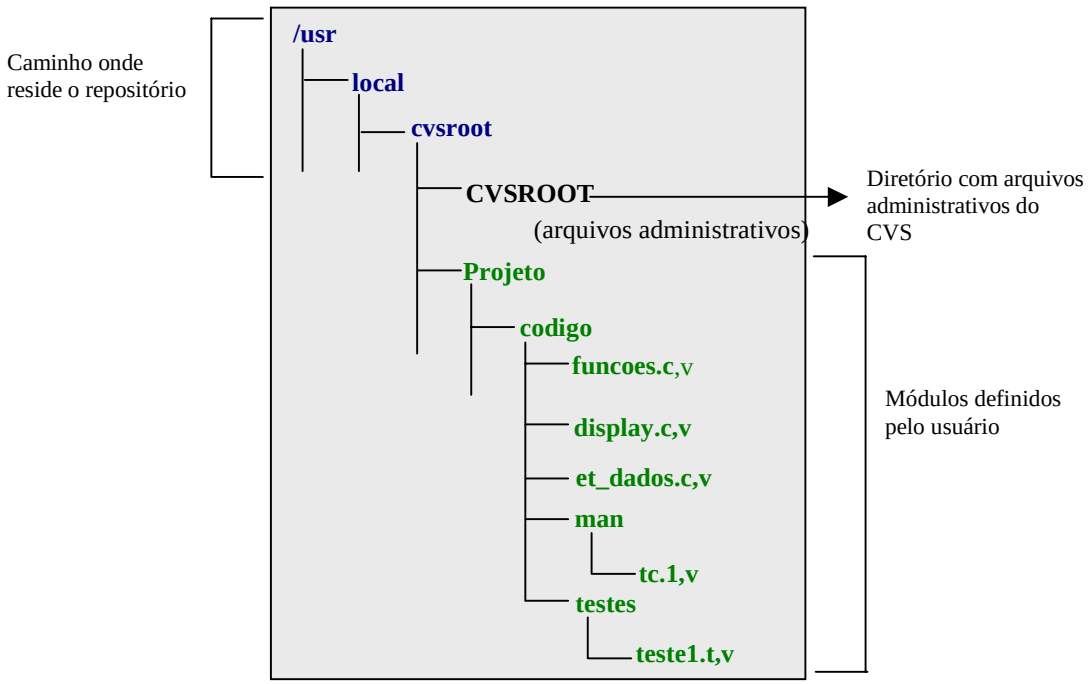

Figura 2.1 - Estrutura de um repositório CVS

No CVS os arquivos, cujos nomes têm acrescido “,v” no final, se referem a históricos, que contêm informações suficientes para recriar qualquer versão do arquivo, localizar o autor, data e hora, além dos comentários da alteração realizada para melhor identificar a razão da

[1] *$CVSROOT* é uma variável de ambiente para o caminho absoluto do repositório, por exemplo, /usr/local/cvsroot

geração daquela revisão. Todos os arquivos históricos são criados apenas para leitura e essa permissão não deve ser alterada pois contém informações que possibilitam a recuperação de versões. As pessoas responsáveis por modificar os arquivos dos diretórios que estão dentro do repositório devem possuir permissão de escrita em tais diretórios.

O CVS mantém as permissões de arquivo dos novos diretórios que são adicionados na árvore de diretórios do repositório; para mudar as permissões deve-se alterar manualmente, quando um novo diretório tiver permissões diferentes do seu diretório pai.

Para criar um repositório CVS deve-se executar o comando *cvs init*, como explicado na Seção 4.1. Após a execução desse comando, o usuário poderá definir os módulos que irão ficar sob controle de versões, no repositório, e também importar projetos já existentes para o repositório (os arquivos dos "projetos" devem ser adicionados ao repositório, um de cada vez, através do comando *cvs add*, como descrito na Seção 5.2.4.

Nas Seções 3 e 4 a seguir, são descritos os requisitos necessários para instalação e configuração do CVS, respectivamente. Somente após corretamente executados esses requisitos, estarão disponíveis os comandos de CVS (Seção 5) para realização do controle de versões no repositório. Na Seção 6 é descrito um exemplo de utilização inicial de CVS, e na Seção 7 são apresentados exemplos de ferramentas que implementam interfaces para utilização de CVS.

3- Instalação do CVS

A instalação do CVS é relativamente simples. Primeiramente, é necessário obter os arquivos de instalação (CD de instalação do *Linux* ou Internet, em http://www.cvshome.org). Caso não possua o CD de instalação, pode-se fazer o *download* do arquivo compactado em http://www.cvshome.org.

3.1- Instalação a partir do CD

Se a distribuição do *Linux* que está sendo utilizada possui um gerenciador de pacotes (como a *Red Hat*) e uma ferramenta de instalação, deve-se executar o arquivo de instalação (no caso do *Red Hat Linux*, o *boot* pode ser feito pelo próprio CD), e escolher a opção "Atualizar Sistema já instalado". Em seguida, o pacote CVS deve ser acrescentado, finalizando a instalação. Após a instalação, é necessário fazer a configuração do CVS, como apresentado na Seção 4.

3.2- Instalação a partir do arquivo compactado

Caso tenha sido feito o *download* do arquivo compactado (.tar.gz), é necessário, primeiramente, descompactar o arquivo em algum diretório. No diretório *home* do usuário, deve-se criar o diretório *CVS*, e descompactar o arquivo .tar.gz nesse diretório. Para isso, os seguintes comandos devem ser executados:

```
cd ~           (para ir para o home)
mkdir cvs
cd cvs
tar -zxfv /diretorio_do_arquivo_compactado/arquivo.tar
```

Nesse ponto, o arquivo já estará descompactado. Deve ser criado um diretório novo, com o nome cvs-1.11.1p1 ou semelhante (dependendo da versão utilizada). É necessário entrar nesse diretório e executar o programa de configuração:

```
cd cvs-1.11.1p1
./configure
```

As instruções da ferramenta de configuração devem ser seguidas. Sugere-se que sejam mantidas as opções *default*. Após a execução do programa *configure*, deve-se executar o comando *make*:

```
make
make install
```

Após a instalação, é necessário fazer a configuração do CVS, como apresentado na Seção 4. Os documentos de referência que fazem parte do pacote CVS (*readme* e *Install*, além do diretório *doc*) devem ser consultados caso o usuário tenha problemas ou dúvidas específicas durante a instalação.

4- Configuração do CVS

A configuração do CVS deve ser realizada em duas etapas: criação do repositório e configuração das variáveis de ambiente, como apresentadas a seguir.

4.1- Criação do repositório

O repositório é o local (*path*) em que o CVS armazena os arquivos que ficam sob o controle de versões. O usuário pode escolher qualquer local para associar ao repositório.

No exemplo apresentado aqui, o diretório /usr/local/cvsroot é utilizado (esse é o caminho_do_repositório suposto nos exemplos posteriores).

Primeiramente, é preciso criar o diretório ao qual o repositório estará associado. É preciso executar um comando CVS que efetivamente define o diretório como o repositório. Isso pode ser feito utilizando-se os comandos:

```
mkdir /usr/local/cvsroot         (criação do diretório)
cvs –d /usr/local/cvsroot init   (criação do repositório)
```

Após a execução do comando (*init*), o CVS cria um diretório CVSROOT (observar as letras maiúsculas) dentro do repositório, com alguns arquivos próprios dele.

4.2- Configuração das variáveis de ambiente

Após a criação do repositório, é necessário criar a variável de ambiente CVSROOT. Para isso, se for utilizado um *shell* POSIX, deve ser editado o arquivo /etc/profile, /etc/bashrc ou /etc/.bashrc (depende do sistema operacional da máquina hospedeira), acrescentando-se as seguintes linhas:

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

Se for utilizado *C shell*, basta digitar na linha de comando:

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

Observar que:

- /usr/local/cvsroot foi o local (*path*) escolhido para ser o repositório.

- Dependendo da utilização do CVS, geralmente é preciso mudar e acrescentar configurações. Uma das aplicações normalmente necessária é a inicialização do serviço de CVS por algum servidor ou *daemon* como o *inetd* ou *xinetd*. Dessa forma, os comandos CVS ficam habilitados para execução de CVS via outros programas / aplicações. Caso sejam necessárias outras configurações particulares, deve-se consultar a documentação do CVS (CVS, 2002).

Se o sistema operacional utiliza o serviço *inetd*, a seguinte linha deve ser inserida no arquivo /etc/inetd.conf:

```
   2401 stream tcp nowait root /usr/bin/cvs cvs --allow-root=caminho_repositorio
pserver
```

Se for utilizado o *xinetd*, as seguintes linhas devem ser inseridas no arquivo /etc/xinetd.conf:

```
service cvspserver
{
      socket_type       = stream
      protocol          = tcp
      wait              = no
      user              = root
      server            = /usr/bin/cvs
      server_args       = --allow-root=repository_path pserver
}
```

5- Comandos do CVS

Conforme visto, para trabalhar com controle de versões do CVS, deve-se criar um repositório, que é o local (diretório) em que os arquivos, que terão suas versões controladas, ficarão "depositados". No entanto, os arquivos não devem ser diretamente atualizados no repositório. A filosofia relacionada ao controle de versões pressupõe que os desenvolvedores e usuários das versões dos arquivos realizem quaisquer modificações nos mesmos em seu próprio "diretório de trabalho". O repositório pode se encontrar no computador local (mesmo que o do diretório de trabalho), ou em qualquer outro computador que ofereça acesso pela Internet ou outro meio.

Para controlar as versões de um arquivo deve-se, primeiramente, fazer uma cópia para o diretório de trabalho, ou seja, fazer um *checkout* do arquivo do repositório para o diretório de trabalho do usuário, editar os arquivos que serão modificados (no diretório de trabalho) e, posteriormente, atualizar o repositório (enviando os arquivos de volta ao repositório por meio de *commit*). Os principais comandos do CVS para executar essas atividades, o *commit* e o *checkout*, são descritos adiante.

O formato geral dos comandos do CVS é:

```
cvs [opções_cvs] comando [opções_comando] [argumentos_comando]
```

- `cvs`: nome do programa - obrigatório
- *`opções_cvs`*: algumas opções que interferem na execução dos comandos do CVS - opcional
- *`comando`*: o comando que o CVS executará - obrigatório
- *`opções_comando`*: opções específicas de cada comando - opcional
- *`argumentos_comando`*: argumentos do comando - opcional

Nas subseções seguintes, são descritas cada uma dessas variações que compõem o formato geral do CVS.

5.1 Opções globais (opções_cvs):

`--allow-root=`*`rootdir`*: utilizando-se essa opção é possível especificar o diretório CVSROOT no qual os usuários poderão fazer login. É nesse diretório que ficará o arquivo de senha dos usuários (no diretório *rootdir*/CVSROOT).

Exemplo:

Se o sistema utiliza o serviço *inetd*, a opção *--allow-root* pode ser especificada no arquivo /etc/inetd.conf:

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

No exemplo, o diretório */usr/local/cvsroot* é utilizado como diretório raiz do CVS, no qual devem ser permitidas as autenticações. Dentro desse diretório será criado o diretório CVSROOT, que contém as informações do CVS e o arquivo de senhas.

Se o *inetd* usa o nome do serviço, ao invés de um número de porta, então a seguinte linha deve ser colocada em */etc/services*:

```
cvspserver 2401/tcp
```

e deve ser colocado *cvspserver*, ao invés de *2401*, no arquivo inetd.conf.

Caso o sistema esteja usando *xinetd* ao invés de *inetd*, o procedimento é um pouco diferente. O usuário deve criar em */etc/xinetd.d/* um arquivo chamado *cvspserver* contendo as seguintes linhas:

```
service cvspserver
{
   port        = 2401
   socket_type = stream
   protocol    = tcp
   wait        = no
   user        = root
   passenv     = PATH
   server      = /usr/local/bin/cvs
   server_args = -f --allow-root=/usr/cvsroot pserver
}
```

O serviço *inetd* ou *xinetd* devem ser reinicializados (*restart*) após essas modificações.

`-T tempdir`: utilizando-se esta opção é possível especificar *tempdir* como sendo o diretório onde serão armazenados os arquivos temporários. Prevalece sobre a variável $TMPDIR e qualquer outro diretório pré-compilado. Deve ser especificado o caminho completo do diretório.

Exemplo:

```
cvs –T /tmp
```

Neste exemplo, o diretório */tmp* será utilizado como temporário.

`-d diretório_cvsroot`: esta opção permite usar o *diretório_cvsroot* como o caminho do diretório raiz do repositório. Prevalece sobre a variável de ambiente $CVSROOT.

Exemplo:

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

Nesse exemplo, o repositório será criado no diretório */usr/local/cvsroot*. Se a variável de ambiente $CVSROOT já estiver apontando para o mesmo local, basta digitar o comando *cvs init* que será obtido o mesmo efeito.

`-e editor`: esta opção permite ao usuário digitar as informações de revisão utilizando o *editor* especificado. Prevalece sobre as variáveis $CVSEDITOR e $EDITOR. O editor especificado é usado ao enviar de volta ao repositório arquivos que foram recuperados e editados (comando *commit*) e para gravar informações de *log* de revisão. Se não for colocada essa opção, será utilizado o editor definido em $CVSEDITOR. Se essa variável também não existir, será utilizado $EDITOR. Caso essa última também não exista, será

utilizado um editor padrão, que varia conforme o sistema operacional: no *Linux*, é utilizado o *vi*, no *Windows*, o *Bloco de Notas*.

Exemplo:

```
cvs –e pico commit arquivo.c
```

Nesse caso, o editor utilizado para gravar a mensagem será o *pico*. Se esta opção for suprimida, será utilizado o editor padrão. Outra possibilidade é a digitação da mensagem de log diretamente na linha de comando, através da opção *-m*, que é uma opção do próprio comando *commit*, como apresentado adiante.

`-H ou --help`: utilizando-se esta opção, o usuário pode visualizar informações relativas ao comando especificado. Se não for especificado qualquer comando (cvs –help), será apresentado o texto de ajuda do CVS.

Exemplo:

```
cvs --help commit
```

`-Q`: utilizando-se esta opção, o CVS executará o comando sem apresentar qualquer resultado da execução. Só será gerada alguma saída em caso de erros graves.

Exemplo:

```
cvs –Q checkout arquivo.c
```

Neste exemplo, em que é executado o comando *checkout*, não será exibida qualquer mensagem na tela, a menos que ocorra algum problema grave.

`-q`: esta opção é semelhante à opção *-Q*, no entanto, algumas mensagens são exibidas.

`-r`: utilizando-se esta opção o usuário determina que os novos arquivos de trabalho serão acessíveis para leitura apenas. A opção padrão do CVS é que todos os novos arquivos tenham permissão de escrita e de leitura.

`-s variável=`*`valor`*: utilizando-se esta opção é possível definir uma variável de usuário. Esse tipo de variável geralmente é utilizado para passar um valor, especificado pelo usuário, para os arquivos administrativos. Para incluir uma variável de usuário, o arquivo administrativo (arquivos do sistema CVS) contém ${=variável}. As variáveis de sistema são da forma ${variável}. Ambas as variáveis devem começar com uma letra e ter caracteres alfa-numéricos. Pode ser particularmente útil especificar essa opção via ‘.cvsrc’.

Exemplo:

```
cvs –s TESTDIR=/work/local/tests
```

Neste exemplo, é criada uma referência (como uma variável de ambiente para o CVS) para um diretório de teste (TESTDIR). Se no arquivo administrativo tiver `sh`

`${=TESTDIR}/runtests`, essa string é expandida para `sh /work/local/tests/runtests` .

`-t:` utilizando-se esta opção, é possível visualizar passo a passo a execução dos comandos do CVS. É uma opção muito útil quando usada com *-n* para verificar qual é o efeito de um comando com o qual o usuário não está familiarizado.

Exemplo:
A execução do comando:

```
cvs -n -t commit teste.txt
```

Apresenta ao usuário as informações a seguir:

```
-> main loop with CVSROOT=/usr/local/cvsroot
->   checkout   (/usr/local/cvsroot/proj/teste.txt,v,   1.1.1.1,   ,
function))
-> ParseInfo(/usr/local/cvsroot/CVSROOT/commitinfo, proj, ALL)
```

Caso fosse executado apenas o comando:

```
cvs commit teste.txt
```

Seriam apresentadas as informações:

```
Checking in teste.txt;
/usr/local/cvsroot/proj/teste.txt,v  <--  teste.txt
new revision: 1.2; previous revision: 1.1
done
```

`-v, --version:` utilizando-se esta opção é possível visualizar a versão do CVS que está instalada.

`-w:` utilizando-se esta opção o usuário determina que os novos arquivos de trabalho terão permissão para escrita e leitura. Para criar arquivos somente-leitura, utilize a opção *-r* ou defina a variável CVSREAD, como já descrito anteriormente (via '.cvsrc'). Por *default*, os novos arquivos têm permissão para escrita e leitura.

Essas são as principais opções para utilização do CVS. Para obter mais informações sobre outros comandos ou verificar mais detalhes, deve ser consultada a documentação do CVS (CVS, 2002).

5.2 Principais Comandos

Nesta seção, a definição e exemplos de utilização dos principais comandos do CVS são apresentados. Alguns deles, já citados anteriormente, são também definidos a seguir.

5.2.1 Import

O comando *import* é utilizado para incorporar diretórios ao repositório. Este comando pode ser utilizado também para a criação inicial do repositório.

```
cvs import  [opções] repositório vendortag releasetag
```

em que `vendortag` é um nome simbólico para a *branch* e `releasetag` é um nome simbólico para uma *release* particular.

O comando *import* possui as seguintes opções básicas:

`-m msg`: coloca *msg* como mensagem de *log*, ao invés de invocar um editor.

`-I name`: especifica nomes de arquivos que devem ser ignorados durante a execução do comando *import*.

Exemplo:

```
cd codigo
cvs import –m "Imported Sources" projeto/funcoes jrandom start
```

Este comando cria na hierarquia do repositório a estrutura de diretório projeto/funcoes com todos os arquivos e diretórios contidos no diretório `código`.

5.2.2 Checkout

Para fazer o controle de versões no CVS, primeiramente deve-se copiar os arquivos que se encontram armazenados no repositório para um diretório de trabalho. O comando que faz essa cópia é o *checkout*, que possui a forma geral:

```
cvs checkout [opções] [arquivos...]
```

O comando *checkout* possui as seguintes opções básicas:

`-D data`: faz *checkout* da revisão mais recente até a *data* especificada.

`-p`: envia os arquivos para a saída padrão (monitor de vídeo, por exemplo).

`-r tag`: recupera a revisão especificada em *tag*.

`-f`: útil apenas quando utilizado com as opções *–D data* ou *–r tag*. Se a revisão especificada não é encontrada, recupera a mais recente (ao invés de ignorar o arquivo).

`-d dir`: cria um diretório (caso ele não exista) chamado *dir* no qual os arquivos de trabalho que estão sendo recuperados serão inseridos.

Exemplos:

1. Para obter uma cópia do arquivo *tc.txt*:

```
cvs checkout tc.txt
```

2. Para obter uma cópia do arquivo *tc.txt* do dia anterior:

```
cvs checkout -D yesterday tc.txt
```

3. Para obter uma cópia da revisão 1.3 do arquivo *tc.txt* que está no diretório *projeto*. Se não existir a revisão 1.3, irá recuperar a revisão mais recente.

```
cvs checkout –fr 1.3 projeto/tc.txt
```

4. Para obter uma cópia do arquivo *tc.txt* e inseri-la no diretório chamado *rascunho*. No repositório o arquivo solicitado está no diretório *projeto*.

```
cvs checkout -d rascunho projeto/tc.txt
```

5.2.3 Commit

Após ter sido feito o *checkout* do arquivo e o mesmo ter sido editado, o usuário deve enviá-lo novamente ao repositório (arquivo modificado). Assim, é criada uma versão do arquivo no repositório. Para isso, deve ser utilizado o comando *commit*.

```
cvs commit [opções] [arquivos...]
```

O comando *commit* possui as seguintes opções básicas:

`-F` *`arquivo`*: faz a leitura da mensagem de *log* de um *arquivo*, ao invés de invocar um editor. A mensagem de *log* é utilizada como um "comentário" daquela atualização do arquivo.

`-m` *`msg`*: coloca *msg* como mensagem de *log*. Semelhante à opção '-F', mas ao invés de criar um arquivo contendo as informações, o comentário é digitado na própria linha de comando.

`-r revisão`: faz o *commit* para a revisão especificada em *revisão*. O usuário deve especificar um valor para uma ramificação (*branch*) ou para a linha principal (*trunk*).

Exemplos:

1. Suponha que tenha sido feito o *checkout* do arquivo teste.c e, em seguida, foram modificadas algumas linhas desse arquivo. Então o autor cria um arquivo contendo alguns comentários sobre as modificações feitas, e envia-o como mensagem de *log*. Observar a seqüência dos comandos CVS:

```
cvs checkout teste.c  {Obtenção de uma cópia do arquivo teste.c}
{São realizadas modificações no arquivo teste.c}
{É criado o arquivo de log comentarios.txt}
```

`cvs commit –F comentarios.txt teste.c` {Envia o arquivo teste.c atualizado para o repositório e o arquivo comentários.txt, contendo as informações sobre a nova revisão de teste.c}

Neste exemplo, caso o arquivo comentários.txt não tivesse sido criado, a mensagem sobre a atualização do arquivo teste.c poderia ser enviada na própria linha de comando:

```
commit –m 'Acrescentadas linhas 40 e 52...' teste.c
```

2. Suponha que o usuário deseja fazer *commit* de um determinado arquivo (por exemplo o arquivo tc.txt) e especificar que a revisão será 2.0. Neste caso, o comando *commit* deve ser utilizado da seguinte forma:

```
cvs commit -r 2.0 tc.txt
```

5.2.4 Add

Este comando permite adicionar um novo arquivo ou diretório ao controle de versões.

Para adicionar arquivos, o usuário deve ter uma cópia de trabalho do diretório ao qual o novo arquivo pertencerá (obtida através da utilização do comando *checkout*). Em seguida, o usuário deve criar o novo arquivo dentro de sua cópia de trabalho do diretório e deve utilizar o comando *cvs add arquivo*, para que o CVS coloque o arquivo especificado sob o controle de versões. O comando *cvs commit filename* deve ser usado para garantir que os outros desenvolvedores possam "enxergar" o arquivo adicionado.

```
cvs add [opções] [arquivos...]
```

O comando *add* possui as seguintes opções:

`-kb:` esta opção deve ser utilizada se o arquivo a ser adicionado contém dados binários.

`-m` *`msg`*: especifica uma descrição para o arquivo (como a mensagem de *log* no caso do comando *commit*).

Exemplos:

1. Considerando-se que o usuário deseja adicionar o arquivo *imagem.jpg* ao repositório, no diretório *projeto*. Os seguintes comandos devem ser executados:

```
cvs checkout projeto
cd projeto
cvs add –kb –m "só um exemplo... " imagem.jpg
cvs commit imagem.jpg
```

2. Considerando-se que o usuário deseja adicionar o arquivo *fatorial.c* ao repositório, no diretório *codigo*. Os seguintes comandos devem ser executados:

```
cvs checkout codigo
cd codigo
cvs add fatorial.c
cvs commit -m "Primeira versao..." fatorial.c
```

5.2.5 Annotate

Este comando permite ao usuário visualizar as últimas modificações feitas em cada linha do arquivo especificado.

```
cvs annotate [opções] [arquivos...]
```

O comando *annotate* possui as seguintes opções básicas:

`-D` *`data`*: mostra a revisão mais recente até *data*.

`-r` *`tag`*: mostra modificações da revisão *tag*.

Exemplo:

1. `cvs annotate` *`arquivo`*

Implicará na seguinte saída:

```
Annotations for arquivo
***************
1.1             (Ana        27-Mar-02): 2+2=4
1.2             (joão       28-Mar-02): 3+5=8
```

Este resultado indica que *arquivo* contém 2 linhas. A primeira linha (2+2=4) foi acrescentada por Ana, no dia 27 de março de 2002. João acrescentou a segunda linha (3+5=8) no dia 28 de março de 2002, sem modificar a linha 1.

2. `cvs annotate –D 2002-08-07 fatorial.c`

Ao executar esse comando são apresentadas as anotações do arquivo *fatorial.c* que ocorreram até a data de 07/08/2002.

Não são mostradas quaisquer informações sobre linhas que foram apagadas ou substituídas. Para isso, deve ser utilizado o comando *diff*.

5.2.6 Diff

Este comando é usado para comparar diferentes revisões de arquivos. A ação *default* é comparar um arquivo no qual o usuário está trabalhando com as revisões que lhe deram origem, e mostrar as diferenças encontradas.

```
cvs diff [opções] [arquivos...]
```

O comando *diff* possui as seguintes opções básicas:

`-D data`: usa a revisão mais recente até *data* para fazer a comparação.

`-R`: examina os diretórios recursivamente (ativado por *default*).

Exemplos

1. `cvs diff -D 2002-08-09 teste.c`

Utilizando este comando, o usuário pode visualizar as diferenças entre a revisão original e a revisão mais recente até 09/08/2002 do arquivo *teste.c*

2. `cvs diff –r 1.1 –r 1.2 teste.c`

Utilizando este comando, o usuário pode visualizar as diferenças entre as revisões 1.1 e 1.2 do arquivo *teste.c*

5.2.7 Update

Este comando é utilizado para atualizar o diretório de trabalho do usuário, deixando a árvore de arquivos/diretórios igual a do repositório. Quando o usuário faz um *checkout* de um arquivo ou diretório, outros usuários podem estar trabalhando no mesmo diretório ou até no mesmo arquivo. Então, deve-se utilizar o comando *update* de tempos em tempos para conciliar o trabalho do primeiro usuário com as revisões enviadas ao repositório desde a última execução do comando *checkout* ou *update*.

```
cvs update [opções] [arquivos...]
```

O comando *update* possui as seguintes opções básicas:

`-C`: sobrescreve os arquivos modificados localmente com cópias originais do repositório (no entanto, o arquivo modificado localmente é salvo como '.#arquivo.revisão').

`-D data`: faz a atualização utilizando a revisão mais recente até *data*.

`-r tag:` recupera a revisão *tag*.

`-f`: útil apenas com as opções '-D data' ou '-r marcador'. Se nenhuma revisão relacionada for encontrada, recupera a revisão mais recente (ao invés de ignorar o arquivo).

`-d`: cria diretórios que existem no repositório e que não estão presentes no diretório de trabalho.

`-P`: apaga os diretórios vazios.

`-p`: direciona arquivos para a saída padrão.

`-R`: atualiza os diretórios de forma recursiva. É utilizado por *default*.

`-j` *`revisão`*: utilizando duas opções '-j', é feito o *merge* das modificações da revisão especificada com o primeiro '-j' em relação à revisão especificada com o segundo 'j', no diretório de trabalho. Com uma opção '–j' é feito o *merge* das mudanças da primeira revisão em relação à revisão especificada com a opção '-j' no diretório de trabalho.

Exemplo:

1. `cvs update teste.c`

Neste comando, o CVS atualiza o arquivo *teste.c* no diretório de trabalho do usuário.

5.2.8 Remove

Este comando é utilizado para remover um arquivo do repositório.

Se o usuário quer ter a possibilidade de recuperar uma revisão mesmo após sua remoção, ele deve executar os seguintes comandos: antes de remover um arquivo, o usuário deve se certificar de que realizou *commit* do arquivo (ou não será capaz de recuperá-lo posteriormente). Então, ele deve remover o arquivo de seu diretório de trabalho (utilizando o comando *rm*, por exemplo) e utilizar o comando *remove* do CVS (*cvs remove arquivo.txt*). Em seguida, o usuário deve executar o comando *commit* (*cvs commit arquivo.txt*) para realmente executar a remoção do arquivo do repositório. Quando o *commit* da remoção do arquivo é executado, o CVS grava a informação de que o arquivo não existe mais. Posteriormente, pode ser criado um arquivo com o mesmo nome, e o CVS guardará todas as versões.

```
cvs remove [opções] [arquivos...]
```

O comando *remove* possui as seguintes opções básicas:

`-f`: apaga o arquivo do diretório de trabalho antes de removê-lo do repositório (faz um *rm*).

`-R`: remove arquivos de forma recursiva.

Exemplo:

```
cd test
rm teste.c
cvs remove teste.c
cvs commit teste.c
```

5.2.9 History

O CVS pode guardar um arquivo de histórico da utilização dos comandos *checkout*, *commit*, *rtag*, *update* e *release*. Através do comando *history* pode-se ter acesso a essas informações. A opção de gerar arquivo de *log* deve estar habilitada através da criação do arquivo '$CVSROOT/CVSROOT/history'.

`cvs history [-report] [-flag] [-opções arg] [arquivos...]`

O comando *history* possui as seguintes opções:

`-c`: reporta cada vez que o *commit* foi usado (ou seja, cada vez que o repositório foi modificado).

`-e` : mostra todos os registros.

`-m` *`módulo`* : informa sobre um módulo particular.

`-o` : mostra um relatório sobre os módulos em que foi feito *checkout*. É o tipo padrão.

`-x` *`tipo`* : extrai um conjunto particular dos registros do *tipo* do histórico do CVS. Cada tipo é especificado por uma letra, e o usuário pode utilizar combinação de letras. Alguns comandos têm um tipo de registro:

F *release*

O *checkout*

E *export*

T *rtag*

Um comando *update* pode causar um dos quatro tipos de registros:

C Um *merge* era necessário mas colisões foram detectadas (requerendo *merge* manual)

G Um *merge* era necessário e foi feito com sucesso

U Um arquivo de trabalho foi copiado do repositório

W A cópia de trabalho de um arquivo foi removida durante o *update* (porque o arquivo não existe mais no repositório)

Um comando *commit* pode gerar três tipos de registros:

A Um arquivo foi adicionado pela primeira vez

M Um arquivo foi modificado

R Um arquivo foi removido

As opções mostradas como '-flags' resumem ou expandem o relatório e não requerem argumentos opcionais:

`-a`: mostra os dados para todos os usuários (o padrão é mostrar os dados apenas para os usuários que estiverem executando o *history*).

`-l`: mostra apenas as últimas modificações.

`-w`: mostra apenas os registros das modificações feitas no mesmo diretório de trabalho no qual está sendo executado o *history*.

As opções mostradas como '-opções arg' resumem o relatório considerando-se um argumento:

`-b` *`str`*: retorna os dados de um registro que contenha o texto *str* tanto no nome do módulo, do arquivo, ou no caminho do repositório.

`-D` *`data`*: mostra os dados desde *data*. É um pouco diferente do uso normal de '-D data'.

`-f arquivo`: mostra os dados para um arquivo particular. Equivalente a especificar o arquivo na linha de comando. Podem ser incluídos várias opções '-f' no mesmo comando. (Ex.: *cvs history –f arquivo1 –f arquivo2*).

`-n módulo`: parecido com –f, no entanto, ao invés de especificar o arquivo, são especificados os módulos.

`-p repositório` : mostra os dados de um repositório particular.

`-r rev` : mostra os dados referentes às revisões desde que a revisão ou marcador *rev* apareçam em arquivos individuais do RCS.

`-t tag`: mostra os registros adicionados a partir da última inserção do marcador *tag* no arquivo *history*. Difere da opção '-r' pelo fato de que esse comando lê apenas o arquivo *history*, enquanto o '-r' lê os arquivos originais do RCS. Esse comando é mais rápido.

`-u nome`: mostra os registros para o usuário *nome*.

5.2.10 Release

Este comando é útil para cancelar, de forma segura, o efeito do comando *cvs checkout*. Como o CVS não trava os arquivos quando o usuário executa o comando *checkout*, não é estritamente necessário que se use este comando. O usuário pode simplesmente apagar o seu diretório de trabalho, mas há o risco de perder mudanças que ele tenha esquecido e, além disso, não será deixada nenhuma indicação no arquivo de histórico do CVS de que *o checkout* foi abandonado.

Para evitar esses problemas, utilize o comando *cvs release*. Este comando verifica, basicamente, que nenhuma modificação para a qual não foi executado o comando *commit* está presente.

```
cvs release [opção] [diretório...]
```

A única opção de execução deste comando é '-d'. Quando o usuário especifica essa opção, o comando *release* irá apagar a cópia local do diretório. Caso não seja especificada essa opção, o comando *checkout* perderá seu efeito (e o usuário não poderá mais executar o comando *commit* para as mudanças antes de uma nova execução do comando *checkout*), mas os arquivos continuarão em seu diretório local.

Exemplo:

Suponha que o usuário tenha executado o comando *checkout* do diretório 'teste', o qual contém vários arquivos. Ele faz várias mudanças em alguns arquivos, e esquece de executar o comando *commit* do diretório. Então, se ele simplesmente apagar o diretório, precisará executar um novo *checkout*, refazer todas as mudanças, e então executar o comando *commit*. No entanto, se ele executar o camando *release*, o CVS irá verificar e informar que algumas mudanças ainda não estão atualizadas no repositório, e o usuário poderá, então, executar o comando *commit*. Suponha agora que ele tenha executado o comando *commit* antes de remover o diretório. Nesse caso, se ele simplesmente excluir o diretório, terá o mesmo efeito da remoção executada pelo comando *cvs release –d teste*. No entanto, executando o comando *release*, a remoção ficará registrada no *history* do CVS. Portanto, é sempre recomendado o uso do comando *release* ao invés da remoção por outros comandos (como o *rm*).

5.2.11 Status

Considerando-se as operações e as mudanças que os usuários fazem em um arquivo do repositório (para o qual foi executado o comando *checkout*) é possível classificá-lo em alguns estados:

- *Up-to-date*: o arquivo é idêntico à última revisão no repositório para a *branch* em uso.
- *Locally modified*: o usuário editou o arquivo e ainda não executou o comando *commit*.
- *Locally added*: o usuário adicionou o arquivo utilizando o comando *add*, e ainda não executou o comando *commit*.
- *Locally removed*: o usuário removeu o arquivo utilizando o comando *remove* e ainda não executou o comando *commit*.
- *Needs checkout*: um usuário remoto executou o comando *commit* para uma revisão mais nova. Na verdade, para obter a revisão mais nova, os outros usuários devem utilizar o comando *update*, ao invés do comando *checkout*.
- *Needs patch*: este estado é semelhante ao estado *Needs checkout*, mas o CVS envia um *patch* (uma parte do arquivo) ao invés do arquivo inteiro.
- *Needs merge*: um usuário remoto executou o comando *commit* para uma revisão mais nova e outro usuário também fez modificações no arquivo. É necessário fazer *merge* das duas revisões. Para isso, execute o comando *cvs update –j*, para unir as revisões, e então execute o comando *commit*.
- *File have conflicts on merge*: este estado é semelhante ao estado *Locally modified*, exceto pelo fato de que ocorreram conflitos durante um *merge* (fusão, união de arquivos). O usuário precisa editar o arquivo, tirar as marcas de erro de *merging* (<<<< e >>>>) e solucionar o problema. Então, ele pode verificar novamente o status, ou fazer *commit* da revisão atualizada.
- *Unknown*: o CVS não tem informações sobre o arquivo. Ocorre, por exemplo, quando o usuário criou um novo arquivo mas não o adicionou ao controle de versões utilizando o comando *add*.

```
cvs status [opções] [arquivos...]
```

O comando *status* possui as seguintes opções:

`-l`: local; roda apenas no diretório atual.

`-R`: opera recursivamente *(default)*.

`-v`: além das informações normalmente apresentadas, imprime todas as *tags* simbólicas juntamente com o valor numérico da revisão ou da *branch* a que se refere.

Exemplo:

Suponha que a revisão 1.6 do arquivo teste.c contenha o seguinte:

```
#include <stdio.h>

int main(int argc,
```

```
          char **argv)
{
    parse();
    if (argc != 1)
    {
        fprintf(stderr, "tc: Nenhum argumento requerido.\n");
        exit(1);
    }
    if (nerr == 0)
        geracodigo();
    else
        fprintf(stderr, "Nao foi gerado codigo.\n");
    exit(!!nerr);
}
```

A revisão com a qual um usuário está trabalhando, baseada na revisão 1.4 contém o seguinte, antes de ser executado o comando *update*:

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

void main()
{
    init_scanner();
    parse();
    if (nerr == 0)
        geracodigo();
    else
        fprintf(stderr, "Nao foi gerado codigo.\n");
    exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
}
```

Então, o usuário executa o comando *cvs update*:

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

Se o usuário executar o comando status, *cvs status arquivo.c*, o CVS retornará a mensagem *'File have conflicts on merge'*. Será necessário abrir o arquivo 'arquivo.c', que conterá o seguinte:

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

int main(int argc,
         char **argv)
{
    init_scanner();
    parse();
    if (argc != 1)
    {
        fprintf(stderr, "tc: Nenhum argumento requerido.\n");
```

```
        exit(1);
    }
    if (nerr == 0)
        geracodigo();
    else
        fprintf(stderr, "Nao foi gerado codigo.\n");
<<<<<<< driver.c
    exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
=======
    exit(!!nerr);
>>>>>>> 1.6
}
```

O usuário deverá tirar as marcas <<<<<, = = = =, e >>>>, corrigir o arquivo, e então poderá executar o comando *commit*.

5.2.12 Login/Logout

Para fazer login/logout do usuário (com *password*), deve ser utilizado:

```
cvs -d pserver:usuario@host:caminho_repositorio login
cvs -d pserver:usuario@host:caminho_repositorio logout
```

Nas duas linhas, *pserver* é o método de autenticação, usuário é o *username* do usuário.

Nessa seção, foram apresentados os principais comandos do CVS. Com esses comandos, o usuário está apto a trabalhar com o controle de versões, desde a instalação do CVS até a administração do repositório e o trabalho de controlar versões.

Para obter outras informações, deve-se digitar na linha de comando *man cvs*, para consultar o manual do CVS.

6- Exemplo de uma Sessão de Trabalho com o CVS

A seguir, um exemplo simples de utilização de alguns comandos do CVS é apresentado de forma a mostrar o uso inicial de um conjunto básico de comandos do CVS. Tais comandos, utilizados em sua forma mais básica, são usados com poucas opções.

Suponha que um usuário queira começar a usar o CVS. Ele possui um projeto que gostaria de colocar sob o gerenciamento de versões. A primeira tarefa a ser executada, portanto, é a criação do repositório:

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

Neste exemplo foi escolhido o *path* /usr/local/cvsroot como caminho do repositório.

Considerando-se que os arquivos do projeto estão no diretório *proj*, os seguintes comandos devem ser executados em seguida para importá-lo para o repositório:

```
cd proj  {posicionar dentro do diretório}
cvs import –m "Imported Sources"  projeto jrandom start
```

O último comando cria na hierarquia do repositório a estrutura de diretório `projeto` com todos os arquivos e diretórios contidos no diretório `proj`.

Neste ponto, os arquivos que estão no diretório `projeto` do repositório estarão disponíveis a outros usuários do repositório, que deverão executar o comando *checkout* para obterem uma cópia de trabalho dos arquivos, como apresentado a seguir:

```
cvs checkout projeto
```

Supondo que um usuário, após executar o comando *checkout*, fez uma modificação no arquivo `cadastro_usuarios.c`. Ele deve enviar este arquivo para o repositório (criando a revisão 1.2 do mesmo), de forma a torná-lo disponível a outros usuários:

```
cvs commit –m "Inserção do campo filiação" cadastro_usuários.c
```

Para visualizar as modificações feitas em cada linha de um arquivo, por exemplo, `cadastro_departamento.c`, o usuário pode utilizar o comando `annotate`:

```
cvs annotate cadastro_departamento.c
```

O comando *history* pode ser utilizado para visualizar o arquivo de histórico do diretório `projeto`:

```
cvs history projeto
```

Obviamente, outros comandos como `update, add, remove, status`, etc, também podem ser executados de acordo com as necessidades dos usuários.

A Figura 6.1 apresenta um esquema de utilização inicial do CVS, seguindo o exemplo acima. Pode-se observar que após ter sido criado o repositório (*init*) e ter sido colocado sob o controle de versões um conjunto de arquivos (*import*) que estavam localmente armazenados em um subdiretório do usuário, é possível fazer *checkouts e commits*, quantos forem necessários, dos arquivos que estão tendo suas versões controladas pelo CVS. Importante notar que esta figura apenas ilustra os arquivos que estão no repositório, pois de fato não existe uma cópia integral do arquivo que foi feito *commit*, apenas as diferenças em relação ao mesmo arquivo que foi feito *checkout* previamente.

Além disso, deve-se reparar que no caso em que se deseja incluir um novo arquivo para ter sua versão controlada, para compor um mesmo conjunto (diretório no CVS) existente, deve-se utilizar o comando *add*.

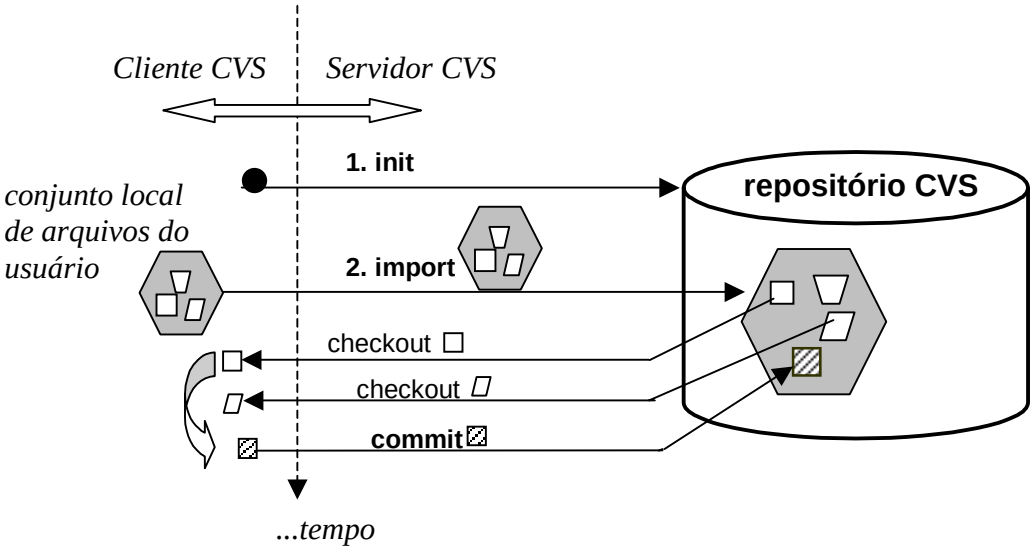

Figura 6. 1 – Esquema de utilização do CVS

7- Exemplos de Ferramentas que Implementam Interface para Repositório CVS

Bonsai

Bonsai é uma ferramenta que permite um bom acompanhamento do progresso de um projeto em termos de *check-ins* realizados. Permite, também, fazer análise estatística de dados relacionados ao repositório, por exemplo, o número de participantes de um projeto.

Algumas características dessa ferramenta são: permite que os usuários consultem os últimos *check-ins* que ocorreram no repositório *CVS*; oferece interface que facilita a visualização de diferenças entre versões de arquivos do repositório CVS; permite identificação visual de quais desenvolvedores são responsáveis por quais seções de um código.

TortoiseCVS

O sistema *TortoiseCVS* [TortoiseCVS, 2002] foi desenvolvido a partir da implementação do *WinCVS*. Como principal característica, o sistema permite que os usuários trabalhem com arquivos armazenados em um repositório CVS diretamente a partir do *Windows Explorer*, ou seja, é possível executar comandos tais como *checkout*, *update*, *commit* e *diff* apenas clicando com o botão direito do mouse em arquivos ou pastas do *Windows Explorer*.

Os objetivos básicos do sistema *TortoiseCVS* são permitir que usuários do sistema operacional *Windows* façam o controle de versões de seus arquivos e disponibilizar uma interface muito simples e fácil de usar. Isso significa oferecer um modo fácil de instalar a ferramenta e executar diversas tarefas automaticamente

SmartCVS

SmartCVS [SmartCVS, 2002] é uma ferramenta desenvolvida em Java que possui como objetivo oferecer as funcionalidades do CVS através de uma interface que seja agradável ao usuário. Algumas das características da ferramenta são: independência de plataforma (pode ser usada em Windows, Unix, Linux, MacOS X ou qualquer plataforma que suporte Java 1.3.1), suporte à administração do repositório (adicionar, excluir usuários; mudar senha, etc) e apresentação gráfica da estrutura de *branches*.

Outras ferramentas:

Exemplos de outras ferramentas que implementam interface para repositório CVS são:

- *MacCvsPro*: http://www.maccvs.org/
- *MacCVSClient*: http://www.heilancoo.net/MacCVSClient/
- *jCVS*: http://www.jcvs.org/
- *TkCVS*: http://www.twobarleycorns.net/tkcvs.html
- *WinCVS*: http://www.wincvs.org/

Referências Bibliográficas

[CVS, 2002] Site oficial do CVS - *Concurrent Versions Systems* [Online]. Acessado em 08/08/2002. http://www.cvshome.org

[Hicks et al., 1998] Hicks, D. L.; Leggett, J. J.; Nürnberg, P. J.; Schnase, J. L. A Hipermedia Version Control Framework. *ACM Transactions on Informations Systems*, vol. 16, Nº. 2, Pages 127-160, 1998.

[Rochkind, 1975] Rochkind M.J. The Source Code Control System IEEE Transactions Software Engineering. Vol.1, nº 4, 364-370, 1975

[SmartCVS, 2002] Site oficial da ferramenta SmartCVS. [Online]. Acessado em 11/09/2002. http://www.smartcvs.com/index.html

[Sommerville, 1995] Sommerville, I. *Software Engineering***,** 5ª edição, Addison Wesley, 1995.

[Tichy, 1985] Tichy, W F. RCS – A System for Version Control. Software Practice and Experience. Vol. 15, nº7, 637-654, 1985.

[TortoiseCVS, 2002] Site oficial do sistema TortoiseCVS. [Online]. Acessado em 11/09/2002. http://www.tortoisecvs.org/

[Vitali e Durand, 1999] Vitali, F.; Durand, D. G. Using versioning to support collaboration on the WWW. Disponível *on-line* em: http://cs-pub.bu.edu/students/grads/dgd/version.html. Visitado em janeiro de 1999.