LABORATÓRIO DE MEIOS POROSOS E PROPRIEDADES TERMOFÍSICAS

e

NÚCLEO DE PESQUISA EM CONSTRUÇÃO

Apostila de CVS

Sistema de Controle de Versões

André Duarte Bueno, UFSC-LMPT-NPC
http://www.lmpt.ufsc.br/~andre
email: andre@lmpt.ufsc.br

Versão 0.4.5

12 de dezembro de 2002

Apostila de CVS. **Versão 0.4.5**

Desenvolvida no
Laboratório de Meios Porosos e Propriedades Termofísicas (http://www.lmpt.ufsc.br)
e no Núcleo de Pesquisa em Construção (http://www.npc.ufsc.br), com apoio do
Curso de Pós-Graduação em Engenharia Mecanica (http://www.posmec.ufsc.br)
e da Universidade Federal de Santa Catarina (http://www.ufsc.br).

Sumário

Lista de Figuras

Listings

Capítulo 1

Introdução ao Controle de Versões Com o CVS

Neste capítulo vamos apresentar o CVS, um sistema para controle das versões de seu programa ou projeto. Vamos ver o que é o cvs, os comandos e a seqüencia de trabalho.

Este capítulo foi escrito usando as referências
[Cederqvist, 1993, Nolden and Kdevelop-Team, 1998, Hughs and Hughes, 1997, Wall, 2001].

1.1 O que é o CVS[1]?

CVS é um sistema de controle de versões (Concurrent Versions System).

- Com CVS você pode gerenciar diferentes versões de um programa (ou projeto).
- Pode atualizar, adicionar e eliminar arquivos e diretórios ao programa.
- Pode criar ramificações de um projeto.
- Múltiplos programadores podem trabalhar ao mesmo tempo no mesmo projeto.
- Informações recentes sobre o CVS você encontra no site (`http://www.cvshome.org/`).

O que é o repositório?

É um diretório com todos os arquivos e subdiretórios do projeto. Adicionalmente, contém arquivos criados pelo programa cvs para o gerenciamento das versões.

O que é uma versão, um tag, um release ?

Todo arquivo tem uma **versão** que é automaticamente definida pelo cvs. Um **tag** é um nome simbólico dado a uma determinada versão do projeto, pode ser usado para delimitar etapas do desenvolvimento de um projeto. Um **release** é uma versão definitiva de todos os arquivos do projeto.

[1] O que o CVS não é ? CVS não é um sistema para construção do soft. Não substitui o gerenciamento do soft. Não substitui a necessidade de comunicação entre o grupo de desenvolvimento. Não serve para testar o soft.

O que é um branch (ramo)?

Um branch (ramo) é usado para dividir um projeto. Normalmente existe o ramo mestre e os ramos secundários.

1.2 Comandos do cvs

Veja a seguir o protótipo do programa cvs. Observe que você passa um conjunto de opções para o cvs; depois, o nome do comando a ser executado e um conjunto de argumentos relativos ao comando.

Protocolo:
cvs [cvs-options] command [command-options-and-arguments]

Os principais comandos do cvs são o **cvs checkout** que baixa os arquivos do repositório para seu local de trabalho, o **cvs update** que atualiza os arquivos do local de trabalho, e o **cvs commit,** que devolve ao repositório os arquivos que você modificou.
Lista-se a seguir a saída do comando **cvs - -help-options** que mostra a lista de opções do programa cvs**.**

Listing 1.1: Saída do comando: cvs - -help-options

```
[andre@mercurio cvs]$ cvs --help-options
CVS global options (specified before the command name) are:
    -H           Displays usage information for command.
    -Q           Cause CVS to be really quiet.
    -q           Cause CVS to be somewhat quiet.
    -r           Make checked-out files read-only.
    -w           Make checked-out files read-write (default).
    -l           Turn history logging off.
    -n           Do not execute anything that will change the disk.
    -t           Show trace of program execution -- try with -n.
    -v           CVS version and copyright.
    -T tmpdir    Use 'tmpdir' for temporary files.
    -e editor    Use 'editor' for editing log information.
    -d CVS_root  Overrides $CVSROOT as the root of the CVS tree.
    -f           Do not use the ~/.cvsrc file.
    -z #         Use compression level '#' for net traffic.
    -x           Encrypt all net traffic.
    -a           Authenticate all net traffic.
    -s VAR=VAL   Set CVS user variable.
```

Lista-se a seguir a saída do comando **cvs - -help-commands** o mesmo apresenta a lista de comandos do cvs.

Listing 1.2: Saída do comando: cvs - -help-commands

```
CVS commands are:
        add          Add a new file/directory to the repository
        admin        Administration front end for rcs
        annotate     Show last revision where each line was modified
        checkout     Checkout sources for editing
        commit       Check files into the repository
        diff         Show differences between revisions
        edit         Get ready to edit a watched file
```

```
        editors      See who is editing a watched file
        export       Export sources from CVS, similar to checkout
        history      Show repository access history
        import       Import sources into CVS, using vendor branches
        init         Create a CVS repository if it doesn't exist
        kserver      Kerberos server mode
        log          Print out history information for files
        login        Prompt for password for authenticating server
        logout       Removes entry in .cvspass for remote repository
        pserver      Password server mode
        rannotate    Show last revision where each line of module was modified
        rdiff        Create 'patch' format diffs between releases
        release      Indicate that a Module is no longer in use
        remove       Remove an entry from the repository
        rlog         Print out history information for a module
        rtag         Add a symbolic tag to a module
        server       Server mode
        status       Display status information on checked out files
        tag          Add a symbolic tag to checked out version of files
        unedit       Undo an edit command
        update       Bring work tree in sync with repository
        version      Show current CVS version(s)
        watch        Set watches
        watchers     See who is watching a file
```

Para obter um help específico sobre um determinado comando use o comando: **cvs -H comando.**

Como alguns comandos podem ser repetidos com freqüência, os mesmos possuem sinônimos. A listagem a seguir apresenta estes sinônimos.

Listing 1.3: Saída do comando: cvs- -help-synonyms

```
[andre@mercurio cvs]$ cvs --help-synonyms
CVS command synonyms are:
        add          ad new
        admin        adm rcs
        annotate     ann
        checkout     co get
        commit       ci com
        diff         di dif
        export       exp ex
        history      hi his
        import       im imp
        log          lo
        login        logon lgn
        rannotate    rann ra
        rdiff        patch pa
        release      re rel
        remove       rm delete
        rlog         rl
        rtag         rt rfreeze
        status       st stat
        tag          ta freeze
        update       up upd
        version      ve ver
```

1.3 Seqüencia de trabalho

Apresenta-se nas seções que seguem os comandos e exemplos de uso do cvs.

Primeiro vamos criar o repositório, a seguir vamos importar um projeto antigo (que já existia) para dentro do repositório. Definido o repositório e importado o projeto, podemos iniciar o uso efetivo do cvs. Vamos criar um diretório de trabalho e com o comando checkout copiar os arquivos do repositório para dentro de nosso diretório de trabalho. Vamos aprender a adicionar novos arquivos e diretórios ao projeto. Finalmente, vamos devolver para o repositório os arquivos modificados com o comando commit.

1.3.1 Roteiro para criar um repositório

1. Setar a variável CVSROOT no arquivo profile (ou no arquivo ~/.bash_profile):

   ```
   CVSROOT=/home/REPOSITORY
   export CVSROOT
   ```

   Se estiver usando o cshel

   ```
   setenv CVSROOT = /home/REPOSITORY
   ```

2. A seguir, você deve criar o diretório onde o repositório vai ser armazenado (se necessário, como root):

   ```
   mkdir /home/REPOSITORY
   ```

   - Todos os usuários que vão usar o cvs devem ter acesso a este diretório. A dica é criar um grupo de trabalho com permissão de leitura e escrita ao diretório do repositório.

3. Você pode criar um grupo cvs, adicionar ao grupo cvs os usuários que terão acesso ao repositório e mudar as permissões de acesso ao repositório.

   ```
   chown -R cvs /home/REPOSITORY
   chmod g+rwx /home/REPOSITORY
   ```

   - [2]A variável CVSUMASK é usada para controlar a forma como os arquivos e diretórios são criados. Consulte um manual de GNU/Linux, Unix, MacOs X para maiores detalhes.

4. O comando **init** inicializa o uso do cvs, adicionando ao diretório do repositório (/home/REPOSITORY) alguns arquivos de controle do programa cvs.

   ```
   cvs init
   ```

   Dê uma olhada no diretório /home/REPOSITORY, observe que foi criado o subdiretório /home/REPOSITORY/CVSROOT. Este subdiretório contém os arquivos de administração do cvs.

   - Os arquivos com *,v são read-only.

1.3.2 Para importar os arquivos de seu projeto antigo para dentro do repositório

Você provavelmente já tem um diretório com projetos antigos e com arquivos de programação (*.h, *.cpp). O comando **import** copia o seu diretório para dentro do repositório.

Protótipo:

cd path_completa_projeto_antigo
cvs import -m ”mensagem” path_proj_no_repositorio nome_release nome_tag

-m ”msg” É uma mensagem curta contento informação sobre o projeto.

path_proj_no_repositorio É a path para o diretório do projeto no repositório.

nome_release É o nome do release inicial.

nome_tag Informa o tag inicial do projeto (normalmente = start).

Vamos adicionar ao repositório o projeto exemplo-biblioteca-gnu localizado, em minha máquina, no diretório:
~/ApostilaProgramacao/Exemplos/Cap-GNU/biblioteca.

```
cd ~/ApostilaProgramacao/Exemplos/Cap-GNU/biblioteca
cvs import -m "Exemplo de biblioteca usando ferramentas gnu"
          exemplo-biblioteca-gnu R1 start
```

A saída gerada pelo comando import é apresentada na listagem a seguir. Observe que a letra N indica um arquivo novo, a letra I um arquivo ignorado (arquivos *.bak *.~são ignorados pelo cvs). A biblioteca recebe um L de library.

Listing 1.4: Saída do comando: cvs import

```
[andre@mercurio biblioteca]$ cvs import -m "Exemplo de biblioteca usando
   ferramentas gnu" exemplo-biblioteca-gnu R1 start
N exemplo-biblioteca-gnu/e87-Polimorfismo.cpp
N exemplo-biblioteca-gnu/e87-Programa.cpp
N exemplo-biblioteca-gnu/e87-TCirculo.cpp
I exemplo-biblioteca-gnu/doxygem.config.bak
N exemplo-biblioteca-gnu/makefile
N exemplo-biblioteca-gnu/e87-TCirculo.h
N exemplo-biblioteca-gnu/doxygem.config
N exemplo-biblioteca-gnu/uso-makefile
N exemplo-biblioteca-gnu/e87-PolimorfismoStatic.cpp
N exemplo-biblioteca-gnu/e87-TElipse.cpp
N exemplo-biblioteca-gnu/e87-TElipse.h
N exemplo-biblioteca-gnu/e87-PolimorfismoDinamic.cpp
N exemplo-biblioteca-gnu/Makefile
N exemplo-biblioteca-gnu/e87-TPonto.cpp
N exemplo-biblioteca-gnu/e87-TPonto.h
N exemplo-biblioteca-gnu/e87-Polimorfismo
I exemplo-biblioteca-gnu/e87-Polimorfismo.cpp~
N exemplo-biblioteca-gnu/makefile-libtool
cvs import: Importing /home/REPOSITORY/exemplo-biblioteca-gnu/.libs
N exemplo-biblioteca-gnu/.libs/libTPonto.al
```

```
L exemplo-biblioteca-gnu/.libs/libTPonto.la

No conflicts created by this import
```

Você pode executar o comando **ls /home/REPOSITORY** ou **tree /home/REPOSITORY** para ver como os arquivos foram importados para dentro do repositório.

Listing 1.5: Como fica o repositorio após a importação

```
/home/REPOSITORY/
|-- CVSROOT
|   |-- modules
|   |-- notify
|   |-- .......
|   `-- verifymsg,v
`-- exemplo-biblioteca-gnu
    |-- Makefile,v
    |-- doxygem.config,v
    |-- doxygem.configold,v
    |-- e87-Polimorfismo,v
    |-- e87-Polimorfismo.cpp,v
    |-- e87-PolimorfismoDinamic.cpp,v
    |-- e87-PolimorfismoStatic.cpp,v
    |-- e87-Programa.cpp,v
    |-- e87-TCirculo.cpp,v
    |-- e87-TCirculo.h,v
    |-- e87-TElipse.cpp,v
    |-- e87-TElipse.h,v
    |-- e87-TPonto.cpp,v
    |-- e87-TPonto.h,v
    |-- makefile,v
    |-- makefile-funciona,v
    |-- makefile-libtool,v
    |-- makefile-ok,v
    `-- uso-makefile,v
```

Dica: Depois de importar seus projetos para dentro do repositório, faça um backup dos projetos (tar -cvzf NomeProjeto.tar.gz NomeProjeto) e remova os arquivos do projeto (rm -fr NomeProjeto). Desta forma você elimina a possibilidade de trabalhar acidentalmente nos arquivos de seu projeto em vez de trabalhar com os arquivos do repositório.

1.3.3 Para baixar o projeto

O nosso repositório já foi criado, já definimos um grupo de trabalho e já copiamos para dentro do repositório um projeto. Agora vamos iniciar o uso efetivo do cvs.

Para copiar os arquivos de dentro do repositório para o diretório onde você deseja trabalhar, usa-se o comando **checkout**. Veja na listagem a seguir o protótipo e os parâmetros do comando **checkout.**

Listing 1.6: Saída do comando: cvs -H checkout

```
[andre@mercurio cvs]$ cvs -H checkout
Usage:
  cvs checkout [-ANPRcflnps] [-r rev] [-D date] [-d dir]
    [-j rev1] [-j rev2] [-k kopt] modules...
```

```
        -A      Reset any sticky tags/date/kopts.
        -N      Don't shorten module paths if -d specified.
        -P      Prune empty directories.
        -R      Process directories recursively.
        -c      "cat" the module database.
        -f      Force a head revision match if tag/date not found.
        -l      Local directory only, not recursive
        -n      Do not run module program (if any).
        -p      Check out files to standard output (avoids stickiness).
        -s      Like -c, but include module status.
        -r rev  Check out revision or tag. (implies -P) (is sticky)
        -D date Check out revisions as of date. (implies -P) (is sticky)
        -d dir  Check out into dir instead of module name.
        -k kopt Use RCS kopt -k option on checkout. (is sticky)
        -j rev  Merge in changes made between current revision and rev.
(Specify the --help global option for a list of other help options)
```

Vá para o diretório onde deseja trabalhar e crie uma cópia de trabalho com **checkout.**

```
Exemplo
mkdir /tmp/workdir
cd /tmp/workdir
cvs checkout exemplo-biblioteca-gnu
cd exemplo-biblioteca-gnu
ls -la
```

Observe que todos os arquivos do projeto foram copiados para o diretório /tmp/workdir/exemplo-biblioteca-gnu. Também foi criado o diretório cvs. Este diretório é mantido pelo programa cvs.

1.3.4 Para criar módulos

Bem, com o comando checkout, fizemos uma cópia de trabalho do projeto exemplo-biblioteca-gnu. Mas o nome exemplo-biblioteca-gnu é muito extenso e seria melhor um nome abreviado. Um módulo é exatamente isto, um nome abreviado para uma path grande no diretório do repositório. Veja a seguir como criar um módulo.

1. Baixa o arquivo modules, localizado em /home/REPOSITORY/CVSROOT/modules

   ```
   cvs checkout CVSROOT/modules
   ```

2. Edita o arquivo modules

   ```
   emacs  CVSROOT/modules
   ```

3. Inclua a linha abaixo (nome_módulo path)

   ```
   lib-gnu exemplo-biblioteca-gnu
   ```

4. Salva o arquivo e envia para o repositório com o comando

```
cvs commit -m
       "adicionado o módulo exemplo-biblioteca-gnu ->lib-gnu"
```

O comando **commit** é usado para devolver para o repositório todos os arquivos novos ou modificados. Veja na listagem a seguir o protótipo do comando commit.

Listing 1.7: Saída do comando: cvs -H commit

```
[andre@mercurio cvs]$ cvs -H commit
Usage: cvs commit [-nRlf] [-m msg | -F logfile] [-r rev] files...
    -n          Do not run the module program (if any).
    -R          Process directories recursively.
    -l          Local directory only (not recursive).
    -f          Force the file to be committed; disables recursion.
    -F logfile  Read the log message from file.
    -m msg      Log message.
    -r rev      Commit to this branch or trunk revision.
(Specify the --help global option for a list of other help options)
```

Veja na listagem a seguir a saída do comando commit executada no diretório de trabalho após a modificação do arquivo CVSROOT/modules.

Listing 1.8: Saída do comando cvs commit após adição de um módulo

```
[andre@mercurio workdir]$ cvs commit -m "adicionado o módulo exemplo-biblioteca-
   gnu -> lib-gnu"
cvs commit: Examining CVSROOT
cvs commit: Examining exemplo-biblioteca-gnu
cvs commit: Examining exemplo-biblioteca-gnu/.libs
Checking in CVSROOT/modules;
/home/REPOSITORY/CVSROOT/modules,v  <--  modules
new revision: 1.2; previous revision: 1.1
done
cvs commit: Rebuilding administrative file database
```

Agora você pode executar o comando checkout de forma abreviada, usando o nome do módulo.

```
mkdir /tmp/workdir2
cd /tmp/workdir2
cvs checkout lib-gnu
```

Para que o comando ficasse ainda mais curto, poderia-se ter utilizado a forma abreviada de checkout.

```
cvs co lib-gnu
```

1.3.5 Para adicionar/remover arquivos e diretórios

O comando **add** agenda a adição de arquivos e diretórios que só serão copiados para o repositório com o comando **commit**. Da mesma forma, o comando **remove** agenda a remoção de arquivos e diretórios que só serão removidos do repositório com o comando **commit**.

Veja a seguir o protótipo destes comandos. Observe que para os comandos funcionarem, você deve estar no diretório de trabalho (/tmp/workdir).

Para adicionar um arquivo

Vamos criar um arquivo leiame.txt, o mesmo contém alguma informação sobre o projeto. Vamos criá-lo com o editor emacs (use o que lhe convier).

```
emacs leiame.txt
...inclui observações no arquivo leiame.txt...
```

Agora vamos agendar a adição do arquivo com o comando add. A saída do comando é apresentada em itálico.

```
cvs add -m "adicionado arquivo leiame.txt" leiame.txt
```

cvs add: scheduling file ‘leiame.txt’ for addition
cvs add: use ’cvs commit’ to add this file permanently

Depois de modificar outros arquivos, podemos efetivamente adicionar o arquivo leiame.txt no repositório usando o comando commit. Observe, em itálico, a saída gerada pelo comando commit.

```
cvs commit
```

cvs commit: Examining .
cvs commit: Examining .libs
cvs commit: Examining novoDir
RCS file:/home/REPOSITORY/exemplo-biblioteca-gnu/leiame.txt,v done
Checking in leiame.txt;
/home/REPOSITORY/exemplo-biblioteca-gnu/leiame.txt,v <--
leiame.txt initial revision: 1.1
done

Alguns comandos do programa cvs podem abrir um editor de texto para que você inclua alguma mensagem relativa a operação que foi realizada. No exemplo acima, depois do **cvs commit,** o cvs abriu o editor emacs. Na sua máquina provavelmente irá abrir o vi. Você pode alterar o editor a ser aberto pelo cvs, setando no arquivo ~./.bash_profile a variável de ambiente CVSEDITOR (Em minha máquina: export CVSEDITOR=emacs).

Para adicionar vários arquivos:

O procedimento é o mesmo, primeiro agenda a adição com add e depois adiciona efetivamente com commit.

```
cvs add -m "adicionados diversos arquivos" *
cvs commit
```

Para adicionar um diretório:

A seqüencia envolve a criação do diretório (mkdir novoDir), o agendamento da adição (cvs add novoDir), e a efetiva adição do diretório com commit.

```
mkdir novoDir
cvs add novoDir
cvs commit -m "adicionado novo diretório" novoDir
```

Para adicionar toda uma estrutura de diretórios num projeto existente:
É o mesmo procedimento utilizado para importar todo um projeto. A única diferença é que a path de importação no repositório vai estar relativa a um projeto já existente. Veja o exemplo:

```
cd novoDir
cvs import -m "msg" path_proj_no_repositorio/novodir
                        nome_release nome_tag.
```

Para remover um arquivo:
Você deve remover o arquivo localmente, agendar a remoção e então efetivar a remoção com commit.

```
rm leiame.txt
cvs remove leiame.txt
cvs commit leiame.txt
```

O comando a seguir remove o arquivo localmente e no cvs ao mesmo tempo.

```
cvs remove -f leiame.txt
```

Para remover vários arquivos:
Você deve remover os arquivos, agendar a remoção e então remover efetivamente com commit.

```
rm -f *
cvs remove
cvs commit -m "removidos diversos arquivos"
```

Dica: Se você fizer alterações locais em um arquivo e depois remover o arquivo, não poderá recuperá-las. Para que possa recuperar as alterações, deve criar uma versão do arquivo usando o comando commit.

Para remover diretórios:
Vá para dentro do diretório que quer deletar, e delete todos os arquivos e o diretório usando:

```
cd nomeDir
cvs remove -f *
cvs commit
//A seguir delete o diretório:
cd ..
cvs remove nomeDir/
cvs commit
```

Para renomear arquivos:
Vá para dentro do diretório onde esta o arquivo a ser renomeado e execute os passos:

```
cd diretorio
mv nome_antigo nome_novo
cvs remove nome_antigo
cvs add nome_novo
cvs commit -m "Renomeado nome_antigo para nome_novo"
```

1.3.6 Para atualizar os arquivos locais

Como o cvs permite o trabalho em grupo. Um segundo usuário pode ter copiado e alterado os arquivos do projeto no repositório.

Um segundo usuário realizou as tarefas a seguir[2]:

```
mkdir /tmp/workdir3
cd /tmp/workdir3
cvs checkout lib-gnu
cd lib-gnu
emacs arquivo-usuario2.txt
cvs add arquivo-usuario2.txt
cvs commit -m "arquivo adicionado pelo usuario2"
```

Se outros usuários do projeto modificaram os arquivos do repositório, então os arquivos com os quais você esta trabalhando podem estar desatualizados. Isto é, se um outro usuário modificou algum arquivo do repositório, você precisa atualizar os arquivos em seu diretório de trabalho.

Bastaria realizar um comando **cvs commit** devolvendo para o repositório todos os arquivos que você modificou, e um comando **cvs checkout**, que copiaria todos os arquivos do repositório, atualizados, para seu diretório de trabalho. Mas este procedimento pode ser lento. Seria mais rápido se o cvs copia-se para seu diretório de trabalho apenas os arquivos novos e modificados. É exatamente isto que o comando **update** faz. O protótipo do comando update é listado a seguir.

Listing 1.9: Saída do comando: cvs -H update

```
[andre@mercurio cvs]$ cvs -H update
Usage: cvs update [-APCdflRp] [-k kopt] [-r rev] [-D date] [-j rev]
    [-I ign] [-W spec] [files...]
        -A      Reset any sticky tags/date/kopts.
        -P      Prune empty directories.
        -C      Overwrite locally modified files with clean repository copies.
        -d      Build directories, like checkout does.
        -f      Force a head revision match if tag/date not found.
        -l      Local directory only, no recursion.
        -R      Process directories recursively.
        -p      Send updates to standard output (avoids stickiness).
        -k kopt Use RCS kopt -k option on checkout. (is sticky)
        -r rev  Update using specified revision/tag (is sticky).
        -D date Set date to update from (is sticky).
        -j rev  Merge in changes made between current revision and rev.
        -I ign  More files to ignore (! to reset).
        -W spec Wrappers specification line.
```

Veja no exemplo como deixar seu diretório de trabalho com os arquivos atualizados.

```
cd /tmp/workdir
cvs update
```

[2]Observe que o nome do diretório obtido pelo usuário 1 é exemplo-biblioteca-gnu e do usuário 2 lib-gnu. Isto é, se você usa *cvs* checkout *path_proj_no_repositorio* o cvs cria o diretório *path_proj_no_repositorio.* Se você usa *cvs* checkout *nome_modulo*, o cvs cria o diretório *nome_modulo.*

```
cvs update: Updating . U arquivo-usuario2.txt
cvs update: Updating .libs
cvs update: Updating novoDir
```

Observe que o arquivo "*arquivo-usuario2.txt*" criado pelo usuário 2 foi adicionado a sua cópia de trabalho.

1.4 Versões, tag's e releases

Descrevemos no início deste capítulo o que é um release e um tag. Apresenta-se a seguir como criar e usar releases e tags.

1.4.1 Entendendo as versões

Todos os arquivos do projeto que foram importados ou adicionados ao repositório tem uma versão. A versão é definida automaticamente pelo programa cvs e se aplica aos arquivos individualmente, isto é, cada arquivo tem sua versão.

De uma maneira geral a versão do arquivo é redefinida a cada alteração do arquivo que foi comutada com o repositório. Assim, se o arquivo leiame.txt, que tem a versão 1.1, foi alterado. Quando o mesmo for devolvido ao repositório com o comando **cvs commit**, o mesmo passa a ter a versão 1.2. Veja Figura 1.1.

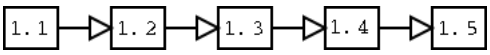

Figura 1.1: Versões de um arquivo.

No exemplo a seguir vai para o diretório de trabalho e modifica o arquivo leiame.txt. Depois realiza um commit. Observe a alteração na versão.

```
cd /tmp/workdir/exemplo-biblioteca-gnu
emacs leiame.txt
...faça alterações no arquivo leiame.txt...e depois salve o arquivo.
cvs commit
```

```
cvs commit: Examining .
cvs commit: Examining .libs
cvs commit: Examining novoDir
Checking in leiame.txt;
/home/REPOSITORY/exemplo-biblioteca-gnu/leiame.txt,v  <--  leiame.tx
new revision: 1.2; previous revision: 1.1 done
```

1.4.2 Para criar tag's

Como dito acima, cada arquivo do repositório vai ter uma versão. Entretanto, você pode realizar diversas modificações no arquivo leiame.txt (1.1 -> 1.2 -> 1.3 -> 1.4 -> 1.5), algumas modificações no arquivo makefile (1.1 -> 1.2 -> 1.3) e nenhuma modificação no arquivo NomePrograma.cpp (1.1). Ou

seja, cada arquivo tem um número de versão diferente. Seria interessante se você pudesse se referir a todos os arquivos do projeto em uma determinada data com um mesmo nome simbólico. Um **tag** é exatamente isto, um nome simbólico usado para obter os arquivos do projeto em determinada data.

Veja na Figura 1.2 como é criado um novo tag. Observe que a versão de cada arquivo não é alterada.

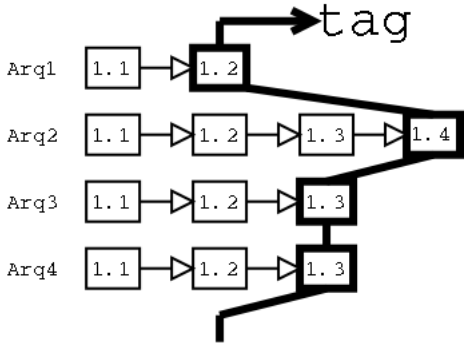

Figura 1.2: Criando um tag.

Assim, em determinado dia eu quero criar um tag simbólico, um nome que vou utilizar para todos os arquivos do projeto naquela data.

Protótipo para criar um tag para um único arquivo:

cd /tmp/workdir
cvs tag nome_release_simbólico nome_arquivo

Protótipo para criar um tag para todos os arquivos do projeto:

cd /tmp/workdir
cvs tag nome_release_simbólico

Veja na listagem a seguir a saída do comando, cvs tag tag1 * executada em nosso diretório de trabalho.

Listing 1.10: Saída do comando: cvs -tag nome

```
[andre@mercurio exemplo-biblioteca-gnu]$ cvs tag tag1 *
cvs tag: warning: directory CVS specified in argument
cvs tag: but CVS uses CVS for its own purposes; skipping CVS directory
T arquivo-usuario2.txt
T doxygem.config
T e87-Polimorfismo
T e87-Polimorfismo.cpp
T e87-PolimorfismoDinamic.cpp
T e87-PolimorfismoStatic.cpp
T e87-Programa.cpp
T e87-TCirculo.cpp
T e87-TCirculo.h
T e87-TElipse.cpp
T e87-TElipse.h
T e87-TPonto.cpp
T e87-TPonto.h
T leiame.txt
```

```
T makefile
T Makefile
T makefile-libtool
T uso-makefile
cvs tag: Tagging novoDir
```

Para recuperar a versão completa do projeto usando o tag que acabamos de criar:

```
cd /tmp/workdir/exemplo-biblioteca-gnu
cvs checkout -r tag1 lib-gnu
```

Observe que para baixar o módulo lib-gnu usamos **cvs checkout lib-gnu,** e para baixar o tag1 do módulo lib-gnu, usamos, **cvs checkout** -r tag1 **lib-gnu**. Ou seja, apenas adicionamos após o comando checkout, o parâmetro -r e o nome do tag.

1.4.3 Para criar release's

Geralmente utilizamos um tag para criar uma versão do projeto que esteja funcionando, ou que compreenda a finalização de um determinado conjunto de tarefas que estavam pendentes. Assim, com o nome do tag você pode recuperar o projeto naquela data usando um nome abreviado.

Entretanto, depois de finalizado o programa ou uma versão funcional, você pode criar um release do programa. A diferença entre o tag e o release, é que o tag não modifica a versão dos arquivos do projeto. O release modifica a versão de todos os arquivos, dando a todos os arquivos um mesmo número de versão. Veja na Figura 1.3 como fica um novo release.

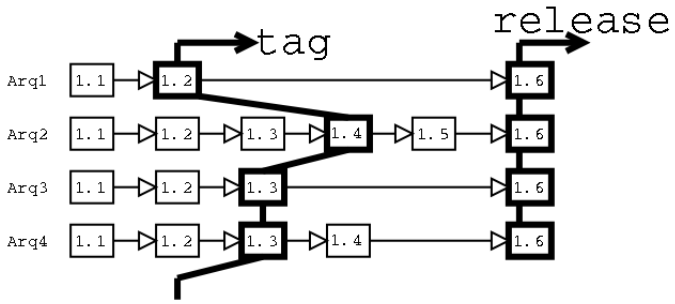

Figura 1.3: Criando um release.

- Um release é geralmente um pacote funcional, se aplica a todos os arquivos do projeto.
- Depois de definido o release o mesmo não pode ser modificado.
- Você deve criar um release sempre que tiver finalizado uma parte importante de seu programa.

Veja a seguir o protótipo para criar um release.

Protótipo:

cvs commit -r número_release

```
cd /tmp/workdir
cvs commit -r 2
```

Além de criar o release, abre o vi[3], para edição de um arquivo de log. Inclua algum comentário a respeito do release que foi criado.

Veja na listagem a seguir a saída do comando **cvs commit -r 2.** Observe que todos os arquivos passam a ter a mesma versão.

Listing 1.11: Saída do comando: cvs commit -r 2

```
[root@mercurio lib-gnu]# cvs commit -r 2
cvs commit: Examining .
cvs commit: Examining .libs
cvs commit: Examining novoDir
Checking in Makefile;
/home/REPOSITORY/exemplo-biblioteca-gnu/Makefile,v  <--  Makefile
new revision: 2.1; previous revision: 1.1
done
Checking in arquivo-usuario2.txt;
/home/REPOSITORY/exemplo-biblioteca-gnu/arquivo-usuario2.txt,v  <--  arquivo-
   usuario2.txt
new revision: 2.1; previous revision: 1.1
done
....
....
Checking in leiame.txt;
/home/REPOSITORY/exemplo-biblioteca-gnu/leiame.txt,v  <--  leiame.txt
new revision: 2.1; previous revision: 1.3
done
```

Protótipo para criar um release e já deletar a cópia do diretório local:

cvs release -d diretório_de_trabalho

1.4.4 Recuperando módulos e arquivos

O cvs permite que tanto os códigos novos como os antigos possam ser recuperados. De uma maneira geral basta passar o nome do arquivo e sua versão (tag, release, módulo).

Protótipo para recuperar um release:

#Pode-se baixar um release antigo, passando o nome do release.
cvs checkout -r nome_release path_projeto_no_cvs
#ou o nome do módulo
cvs checkout -r nome_release nome_modulo

Protótipo para recuperar um arquivo de uma versão antiga:

cvs update *-p* -r nome_release *nome_arquivo > nome_arquivo*

-p Envia atualizações para saída padrão (a tela).

-r nome_release Indica a seguir o nome do release.

[3]ou o editor setado com CVSEDITOR. No vi digite esc :q para sair, esc :q!. para sair sem salvar alterações.

nome_arquivo O nome do arquivo a ser baixado

> *nome_arquivo* Redireciona da tela para o arquivo *nome_arquivo.*

No exemplo a seguir, recupera o arquivo leiame.txt do tag1.

```
cvs update -p -r tag1 leiame.txt > leiame-tag1.txt
```

```
===================================================
Checking out leiame.txt RCS:
/home/REPOSITORY/exemplo-biblioteca-gnu/leiame.txt,v VERS: 1.2
***************
```

1.5 Para verificar diferenças entre arquivos

O programa cvs tem suporte interno ao programa diff (apresentado no Capítulo **??**), permitindo comparar os arquivos que estão sendo usados localmente com os do repositório.

Protótipo:

#Compara arq local e arq do repositório
cvs diff arq
#Verifica diferenças de todos os arquivos
cvs diff

O usuário 2, modificou o arquivo leiame.txt depois de criado o release 2. Veja na listagem a seguir a saída do comando cvs diff, executado pelo usuário 1.

Listing 1.12: Saída do comando: cvs-diff

```
[andre@mercurio exemplo-biblioteca-gnu]$ cvs diff
cvs diff: Diffing .
Index: leiame.txt
===================================================================
RCS file: /home/REPOSITORY/exemplo-biblioteca-gnu/leiame.txt,v
retrieving revision 2.2
diff -r2.2 leiame.txt
7,11d6
< Alteração realizada depois de criado o tag1.
<
<
< Modificações realizadas depois do release.
< Pelo usuário 2.
cvs diff: Diffing .libs
cvs diff: Diffing novoDir
```

1.6 *Verificando o estado do repositório*

O cvs tem um conjunto de comandos que você pode usar para verificar o estado dos arquivos armazenados no repositório.

1.6.1 *Histórico das alterações*

Você pode obter uma lista com o histórico das alterações realizadas.
Mostra: data, hora, usuário, path usada (ou módulo, ou ramo), diretório de trabalho:

Protótipo:

cvs history

1.6.2 *Mensagens de log*

Você pode obter uma lista dos log's do arquivo.
Mostra: path no repositório, versão, nomes simbólicos, revisões e anotações realizadas.

Protótipo:

cvs log arquivo

Veja a seguir a saída do comando **cvs -log leiame.txt**. Observe as diferentes revisões e anotações, o nome do autor. Observe nos nomes simbólicos o tag1 e a referencia a versão 1.2, isto significa que quando o usuário solicitar o tag1, corresponde a versão 1.2 do arquivo leiame.txt.

Listing 1.13: Saída do comando: cvs -log leiame.txt

```
RCS file: /home/REPOSITORY/exemplo-biblioteca-gnu/leiame.txt,v
Working file: leiame.txt
head: 2.2
branch:
locks: strict
access list:
symbolic names:
        tag1: 1.2
keyword substitution: kv
total revisions: 5;     selected revisions: 5
description:
adicionado arquivo leiame.txt
----------------------------
revision 2.2
date: 2002/08/12 23:28:55;  author: andre;  state: Exp;  lines: +4 -0
Modificações realizadas no leiame.txt depois de criado o release.
----------------------------
revision 2.1
date: 2002/08/12 23:12:05;  author: andre;  state: Exp;  lines: +0 -0
Criado o release 2.
----------------------------
revision 1.3
date: 2002/08/12 23:10:32;  author: andre;  state: Exp;  lines: +1 -0
Alterações no leiame.txt depois de criado o tag1.
----------------------------
revision 1.2
date: 2002/08/12 22:45:56;  author: andre;  state: Exp;  lines: +5 -0
Modificações no arquivo leiame.txt
----------------------------
revision 1.1
date: 2002/08/12 21:33:43;  author: andre;  state: Exp;
Efetivamente adicionado o arquivo leiame.txt
=============================================================================
```

1.6.3 *Anotações*

Você pode obter uma lista das anotações realizadas.
Mostra: versão, nome usuário, data, mensagem.

Protótipo:
cvs annotate

1.6.4 Verificando o status dos arquivos

O comando status mostra uma série de informações a respeito do arquivo. O mesmo pode ser utilizado para verificar quais arquivos precisam ser atualizados. Veja a seguir o protótipo.

Protótipo:
cvs status

-v	*Mostra ainda os tag's.*
-R	*Processamento recursivo.*
-l	*Somente este diretório.*

Informações listadas pelo comando status:

Up-to-date O arquivo não foi alterado.

Locally modified O arquivo foi modificado localmente.

Locally added O arquivo foi adicionado localmente.

Locally removed O arquivo foi removido localmente.

Needs checkout O arquivo foi alterado por terceiro e precisa ser atualizado (Com um update baixa o arquivo mesclando-o com o local. Com um commit atualiza no servidor).

File had conflicts on merge O arquivo apresenta conflitos após a mistura.

Veja na listagem a seguir a saída do comando status. Observe que o arquivo foi localmente modificado.

Listing 1.14: Saída do comando: cvs -status leiame.txt

```
[andre@mercurio exemplo-biblioteca-gnu]$ cvs status leiame.txt
===================================================================
File: leiame.txt        Status: Locally Modified

   Working revision:    2.2     Result of merge
   Repository revision: 2.2     /home/REPOSITORY/exemplo-biblioteca-gnu/leiame.
      txt,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

1.7 Ramos e Misturas (Branching and Merging)

O programa cvs permite que você crie um ramo principal para seu projeto e ramos derivados. Posteriormente você pode misturar os diferentes ramos.

Veja na Figura 1.4 a disposição de um novo ramo.

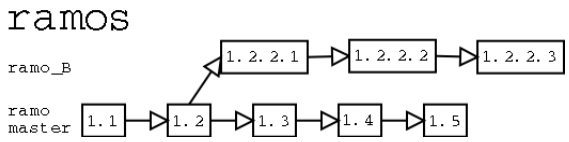

Figura 1.4: Como ficam os ramos.

Depois de finalizado um release de um programa, é bastante usual a criação de três ramos.

Digamos que você esteja trabalhando no projeto gnome, e que o release 1.0 já foi suficientemente testado, podendo ser publicado. Então, você cria o release 1.0.

```
Release gnome-1.0.
```

Observe que é a versão final do gnome 1.0.

Agora você pode criar um ramo de patch, o mesmo vai conter os arquivos da versão 1.0, mas com correções de bugs que tenham sido localizados. Assim, se foi identificado algum bug na versão 1.0, você faz as alterações no ramo gnome 1.0-patch, deixando o release 1.0 inalterado.

```
Ramo: gnome-1.0-patch
```

Você pode criar um ramo novo, onde ficarão os arquivos da nova versão do gnome.

```
Ramo: gnome-1.1
```

Ou seja, vamos ter três ramos. O release 1.0 que não será mais alterado. O patch que vai ter as correções de bugs da versão 1.0 e o 1.1 que terá os arquivos da nova geração do gnome.

1.7.1 Trabalhando com ramos

Para criar um ramo a partir da cópia de trabalho local (-b de branch):

```
cvs tag -b nome_do_ramo
```

Para criar um ramo a partir de um release existente, sem uma cópia de trabalho local:

```
cvs rtag -b -r nome_do_release nome_do_ramo path_no_repositorio
```

Baixando um ramo:

```
cvs checkout -r nome_do_ramo path_no_repositorio
```

Atualização dos arquivos locais de um dado ramo:

```
cvs update -r nome_do_ramo path_no_repositorio
```

ou

```
cvs update -r nome_do_ramo nome_modulo
```

Para saber com qual ramo você esta trabalhando, verifique o nome do ramo em “Existing tags”.

```
cvs status -v nome_arquivo
```

1.7.2 Mesclando 2 versões de um arquivo

Com a opção -j, você pode verificar as diferenças entre 2 versões de um arquivo. Veja o protótipo e um exemplo a seguir.

Protótipo:

cvs update -j versãoNova -j versãoVelha nomeArquivo

```
cvs update -j 2 -j tag1 leiame.txt
U leiame.txt
RCS file: /home/REPOSITORY/exemplo-biblioteca-gnu/leiame.txt,v
retrieving revision 2.2
retrieving revision 1.2
Merging differences between 2.2 and 1.2 into leiame.txt
```

Observe a mensagem apresentada. O cvs recupera a versão 2.2 (relativa ao release -j 2) e a versão 1.2 (relativa ao tag1) e mistura as duas no arquivo leiame.txt.

1.7.3 Mesclando o ramo de trabalho com o ramo principal

Digamos que você esta trabalhando no ramo principal. Que um segundo usuário criou o ramo_B e fez alterações no ramo_B. Agora você quer incluir as alterações do ramo_B no ramo principal.

1. Baixa o módulo de trabalho

   ```
   cvs checkout nome_modulo
   ```

2. Baixa o upgrade do ramo_B. Ou seja, atualiza os arquivos locais mesclando os mesmos com os do ramo_B.

   ```
   cvs update -j ramo_B
   ```

3. Resolve os possíveis conflitos. Alguns arquivos que tenham sido modificados por outros usuários podem ter conflitos de código, você precisa resolver estes conflitos.

   ```
   Correção de possíveis conflitos de código...
   ```

4. Copia os arquivos de volta para o repositório, atualizando o repositório.

   ```
   cvs commit -m "Ramo mestre mesclado com ramo_B"
   ```

5. Para deletar o diretório local de trabalho (use com cuidado).

   ```
   rm -f -r path_local/
   ```

1.8 Configuração do cvs no sistema cliente-servidor[3]

Neste tipo de configuração o projeto principal fica na máquina servidora (ou seja o repositório fica no servidor). O usuário baixa o programa para sua máquina local usando checkout, faz modificações e depois copia as modificações para o repositório usando o comando commit.

- O servidor para uso do cvs pode ser um micro pouco potente (133MHz, 32Mb), com HD suficiente (4 vezes o tamanho do projeto).
- O acesso ao repositório é dado por:
  :tipo_de_acesso:path_do_projeto

  onde tipo_de_acesso:

  :local: Você esta na máquina servidora: Se estiver trabalhando na mesma máquina do repositório, você faz um acesso local ao projeto e pode acessar os arquivos do projeto diretamente com **cvs checkout path_no_repositorio**.

  :servidor: Você esta na máquina cliente: Se estiver remoto, deve-se incluir o nome do servidor
  : servidor: user@hostname:/path/to/repository
  Ex:
  export CVSROOT=:pserver: usuario1@nome_servidor:/path_repositorio
  cvs checkout path_no_repositorio.

- Consulte o manual do cvs para ver como configurar o servidor.

Exemplo:

Por default, a conexão do cvs usa o protocolo RSH. Assim, se andre esta na máquina mercurio.lmpt.ufsc.br e o servidor é enterprise.lmpt.ufsc.br no arquivo '.rhosts' deve ter a linha:

```
mercurio.lmpt.ufsc.br andre
```

Para testar:

```
rsh -l bach enterprise.lmpt.ufsc.br 'echo $PATH'
```

Deve-se setar na máquina cliente o endereço do programa cvs no servidor com a variável de ambiente CVS_SERVER.

1.8.1 Variáveis de ambiente

Variáveis de ambiente do cvs definidas no arquivo profile:

$CVSROOT Diretório de trabalho do cvs.

$CVS_SERVER Endereço do programa cvs na máquina servidora.

$CVSEDITOR Editor default do cvs.

$CVSUMASK Define o formato dos arquivos novos a serem criados.

1.9 Como baixar programas de terceiros usando o cvs

Veja a seguir as instruções postadas no site http://www.devel.lyx.org/cvs.php3 e que são utilizadas para baixar o programa LyX usando o cvs.

Anonymous CVS login

In order to make anonymous CVS access easier, you should set your CVSROOT environment variable to

```
:pserver:anoncvs@anoncvs.lyx.org:/usr/local/lyx/cvsroot
```

(Alternatively, you can use the -d option to the cvs commands.).

Now just do:

```
cvs login
```

The password is lyx. Finally,

```
cvs checkout lyx-devel
```

This will make a directory lyx-devel and download lots of files into that directory. Of course you can say cvs checkout lyx-1_0_x instead, depending on which module you'd like to download.

1.10 Frontends

Existem front-ends para o programa cvs. Para o GNU/Linux, de uma olhada no cervisia, encontrado no site (http://cervisia.sourceforge.net/) e ilustrado na Figura 1.5. Um frontend para o Windows é o wincvs, ilustrado na Figura 1.6.

1.11 Sentenças para o cvs

- Quando você passa como parâmetro de algum comando do cvs um diretório. Todos os arquivos do diretório e subdiretórios sofrem o efeito do comando.
- Uma path, um módulo, um ramo são equivalentes. Um tag, um release, uma versão são equivalentes. Ou seja, se o programa cvs espera uma path_do_repositório você também pode passar o nome de um módulo ou de um ramo. Se o cvs espera um nome de versão, você pode passar o nome do tag, ou o nome do release.
- Monte um grupo de trabalho:
  Para trabalhar em grupo em um projeto, você deve-se definir um grupo no GNU/Linux, Unix, MacOS X. O grupo de trabalho terá acesso aos arquivos do repositório num sistema cliente-servidor.
- Pode-se definir diferentes formar de acesso aos arquivos, autenticações e sistemas de segurança. Dê uma olhado no manual de configuração do cvs.
- Links para cvs: http://www.cvshome.org/, a casa do cvs. Outros frontends podem ser encontrados em http://www.cvsgui.org/download.html, e http://www.lincvs.org.

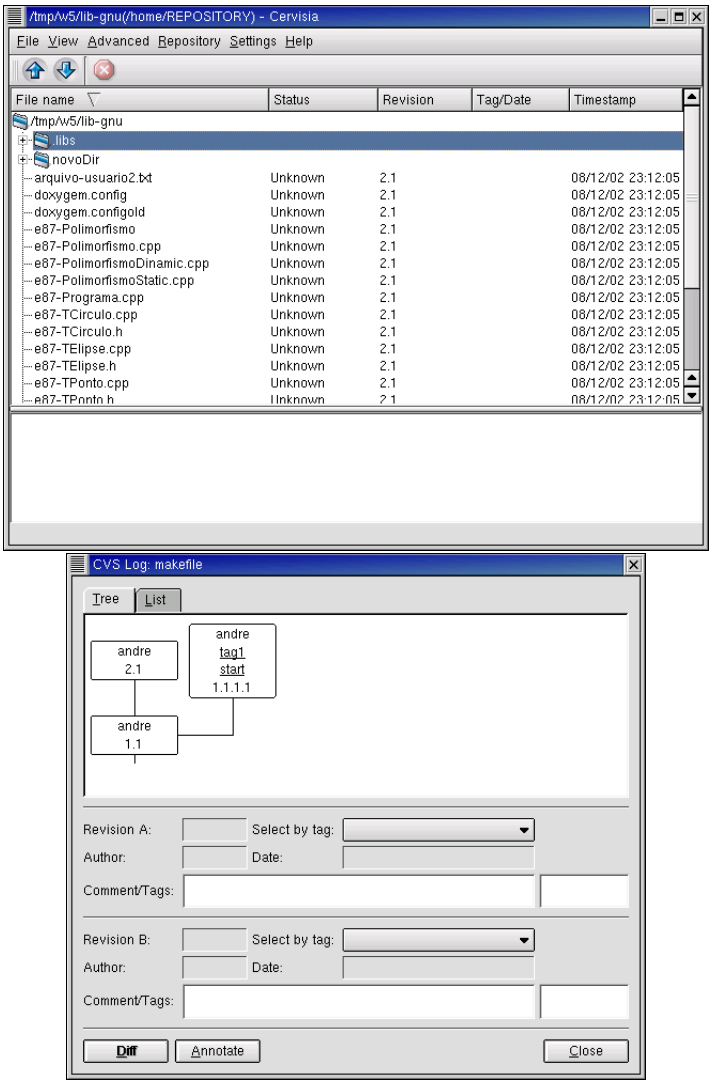

Figura 1.5: Um frontend para o cvs no GNU/Linux, Unix (o cervisia).

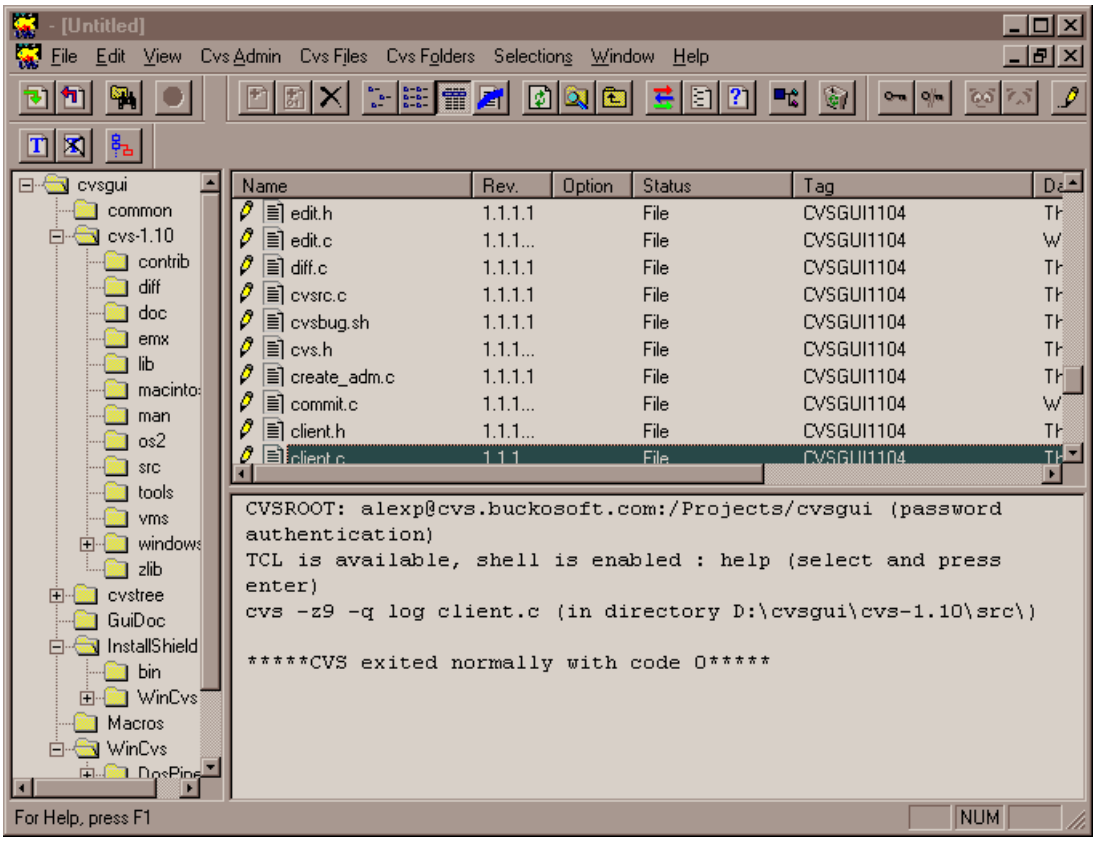

Figura 1.6: Um frontend para o cvs no Windows.

1.12 Um diagrama com os comandos do cvs

Veja na Figura 1.7 um diagrama com os comandos do cvs.

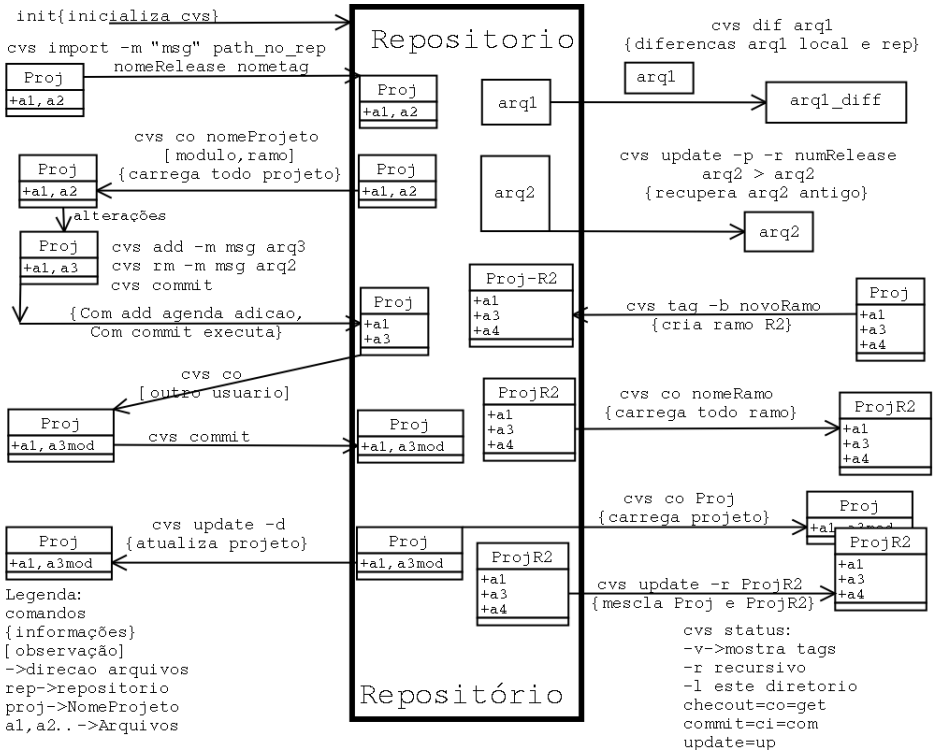

Figura 1.7: Diagrama com os comandos do cvs.

Referências Bibliográficas

[Cederqvist, 1993] Cederqvist, P. (1993). *Version Management with CVS*. GNU.

[Hughs and Hughes, 1997] Hughs, C. and Hughes, T. (1997). *Object Oriented Multithreading using C++: architectures and components*, volume 1. John Wiley Sons, 2 edition.

[Nolden and Kdevelop-Team, 1998] Nolden, R. and Kdevelop-Team (1998). *The User Manual to KDevelop*. kdevelop team.

[Wall, 2001] Wall, K. (2001). *Linux Programming Unleashed*, volume 1. SAMS, 2 edition.