Version Management with CVS

for cvs

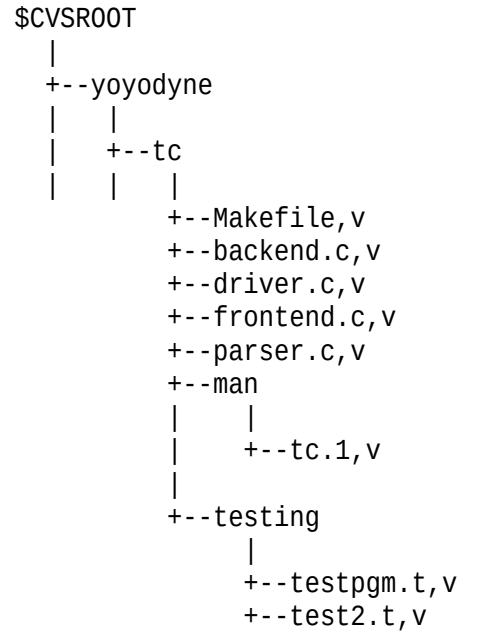
1.2 What is CVS not?

cvs

```
$ cd ..
$ cvs release -d tc
M driver.c
? tc
You have [1] altered files in this repository.
Are you sure you want to release (and delete) module `tc': n
** `release' aborted by user choice.
```

2.2 How data is stored in the repository

For most purposes it isn't important *how* cvs



The history files contain, among other things, enough information to recreate any revision of the file, a log of all commit messages and the user-name of the person who committed the revision. The history files are known as *RCS files*, because the first program to store files in that format was a version control system known as

cvs tries to set up reasonable file permissions for new directories that are added inside the tree,

```
/usr/local/cvsroot/yoyodyne/tc/Attic/backend.c,v
```

instead. It should not matter from a user point of view whether a file is in the attic; cvs keeps track of this and looks in the attic when it needs to. But in case you want to know, the rule is that the RCS file is stored in the attic if and only if the head revision on the trunk has state dead. A

version control. To lock an entire tree, you need to lock each directory (note that if you fail to obtain any lock you need, you must release the whole tree before waiting and trying again, to avoid deadlocks).

Note also that cvs expects writelocks to control access to individual `foo,v' les. rcs has a scheme where the `,foo,' le serves as a lock, but cvs does not implement it and so taking out a cvs writelock is recommended. See the comments at rcs_internal_lock le in the cvs source code for further discussion/rationale.

2.2.7 How les are stored in the CVSROOT directory

The `$CVSROOT/CVSROOT' directory contains the various administrative les. In some ways this directory is just like any other directory in the repository; it contains rcs les whose names end in `,v', and many of the cvs commands operate on it the same way. However, there are a few di erences.lhe287(lo)[(rcs)-994(infac408(troi(`7(k)-994(the)]TJ/F52 05u78.f 25.45c24 0 Td[VCy)]TJ/F51 10.9

```
`Repository
```

B*name*/*rev*/*expansion*

where *expansion* should be ignored, to allow for future expansion.

`Baserev.tmp'

2.5 Multiple repositories

2.9.1 Server requirements

The quick answer to what sort of machine is suitable as a server is that requirements are modest | a server with 32M of memory or even less can handle a fairly large source tree with a fair

There is no need to edit `inetd.conf` or start a `cvs` server daemon.

There are two access methods that you use in CVSROOT for rsh. `:server:` specifies an internal

```
cvspserver      2401/tcp
```

and put `cvspserver` instead of `2401` in `inetd.conf'.

Once the above is taken care of, restart your `inetd`, or do whatever is necessary to force it to reread its initialization files.

Because the client stores and transmits passwords in cleartext (almost | see Section 2.9.3.3 [Password authentication security], page 21, for details), a separate `cvs` password file may be used, so people don't compromise their regular passwords when they access the repository. This file is `$CVSROOT/CVSROOT/passwd`

2.9.3.2 Using the client with password authentication

Before connecting to the server, the client must *log in* with the command `cvs login`. Logging

You need to edit `inetd.conf`

3.1.2 Creating Files From Other Version Control Systems

If you have a project which you are maintaining with another version control system, such as rcs

Then, use add

4 Revisions

For many uses of cvs

```
$ cvs status -v backend.c
===================================================================
File: backend.c         Status: Up-to-date

    Version:            1.4     Tue Dec  1 14:39:01 1992
```

5 Branching and merging

CVS allows you to isolate changes onto a separate line of development, known as a *branch*. When

```
cvs update -j 1.2.2.2 -j R1fix m.c     # Merge changes from 1.2.2.2 to the
                                       # head of the R1 x branch
```

The problem with this is that you need to specify the 1.2.2.2 revision manually. A slightly better approach might be to use the date the last merge was done:

```
cvs update -j R1fix:yesterday -j R1fix m.c
```

Better yet, tag the R1 x branch after every merge into the trunk, and then use that tag for subsequent merges:

```
cvs update -j merged_from_R1fix_to_trunk -j R1fix m.c
```

5.8 Merging differences between any two revisions

With two `-j *revision*' flags, the update (and checkout) command can merge the differences between any two revisions into your working file.

```
$ cvs update -j 1.5 -j 1.3 backend.c
```

will *remove* all changes made between revision 1.3 and 1.5. Note the order of the revisions!

7.2 Removing files

Modules change. New files are added, and old files disappear. Still, you want to be able to retrieve an exact copy of old releases.

Here is what you can do to remove a file, but remain able to retrieve old revisions:

```
$ mv old new
$ cvs remove old
$ cvs add new
$ cvs commit -m "Renamed old to new" old new
```

```
$ cvs tag -d tag2 new
:::
```

8 History browsing

Once you have used cvs

8.4 Annotate command

Command

9 Handling binary files

The most common use for cvs is to store text files. With text files, cvs can merge revisions,

10 Multiple developers

Needs Merge

```
{
    fprintf(stderr, "tc: No args expected.\n");
```

Any number of people can be reading from a given repository at a time; only when someone is

Command

10.6.4 Information about who is watching and editing

cvs watchers [-lR] Command

11 Revision management

12 Keyword substitution

As long as you edit source files inside your working copy of a module you can always find out the state of your files via `cvs status' and `cvs log'. But as soon as you export the files from your development environment it becomes harder to identify which revisions they are.

CVS can use a mechanism known as *keyword substitution* (or *keyword expansion*) to help

13 Tracking third-party sources

If you modify a program to better fit your site, you probably want to include your modifications when the next release of the program arrives. cvs can help you with this task.

In the terminology used in cvs, the supplier of the program is called a *vendor*. The unmodified distribution from the vendor is checked in on its own branch, the *vendor branch*. cvs reserves branch 1.1.1 for this use.

When you modify the source and commit it, your revision will end up on the main trunk. When a new release is made by the vendor, you commit it on the vendor branch and copy the modifications onto the main trunk.

Use the

14 How your build system interacts with CVS

As mentioned in the introduction, cvs does not contain software for building your software from

The `PreservePermissions` features do not work with client/server `cvs`. Another limitation is

Appendix A Guide to CVS commands

A.3 Default options and the ~/.cvsrc file

There are some `command_options` that are used so often that you might have set up an alias or

`-w` Make new working files read-write. Overrides the setting of the `$CVSREAD` environment

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

A.6.1 admin options

Some of these options have questionable usefulness for cvs but exist for historical purposes. Some even make it impossible to use cvs until you undo the e ect!

-A*old le* Might not work together with cvs. Append the access list of *old le* to the access list of the

`-N`*name*`[:[`*rev*`]]`
: Act like `-n', except override any previous assignment of *name*. For use with magic

rarely useful. It means to delete revisions up to, and including, the tag R_1_02. But beware! If there are les that have not changed between R_1_02 and R_1_03 the le will have *the same* numerical revision number assigned to the tags R_1_02 and R_1_03. So not only will it be impossible

Use

A.8.2 commit examples

A.8.2.1 Committing to a branch

You can commit to a branch revision (one that has an even number of dots) with the `-r' option. To create a branch revision, use the `-b' option of the `rtag' or `tag' commands (see Section A.17 [tag], page 102 or see Section A.16 [rtag], page 101). Then, either `checkout' or `update' can be

A.9 di | Show di erences between revisions

Synopsis: di [-lR] [format_options] [[-r rev1

A.11 history | Show status of les and users

Synopsis: history [-report] [- ags] [-options args] [les: : :]

Requires: the le `

One of three record types results from commit:

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

`M` A file was modified.

`R` A file was removed.

The options shown as `-flags' constrain or expand the report without requiring option arguments:

`-a` Show data for all users (the default is to show data only for the user executing

If cvs decides a file should be ignored (see Section C.9 [cvsignore], page 126), it does not import it and prints `I ' followed by the filename (see Section A.12.2 [import output], page 96, for a complete description of the output).

If the file `$CVSROOT/CVSROOT/cvswrappers

U *le*

<d
d> Select all revisions dated *d* or earlier.

d<
>d Select all revisions dated *d* or later.

d Select the single, latest revision dated *d* or earlier.

The `>' or `<' characters may be followed by `=' to indicate an inclusive range rather than an exclusive one.

Note that the separator is a semicolon (;).

`-h` Print only the name of the rcs le, name of the le in the working directory, head, default bJ -edirons list, locks, symbolic names, and

`-l` Local; run only in current working directory. (Default is to run recursively).

`-N` Do not print the list of tags for this le. This option can be very useful when your site uses a lot of tags, rather than "more"'ing over 3 pages of tag information, the log information is presented without tags at all.

`-R` Print only the name of the rcs le.

`-r`*revisions*

Print information about revisions given in the comma-separated list *revisions* of revisions and The following table explains the available range formats:

rev1:*rev2* Revisions *rev1* to *rev2*

A.14 rdi | 'patch' format diffs between releases

rdiff [-flags] [-V vn] [-r t|-D d [-r t2|-D d2]] modules

The symbolic tags are meant to permanently record which revisions of which files were used in creating a software distribution. The `checkout` and `update` commands allow you to extract an exact copy of a tagged release at any time in the future, regardless of whether files have been changed, added, or removed since the release was tagged.

A.18.1 update options

These standard options are available with `update` (see Section A.5 [Common options], page 80, for a complete description of them):

`-D date` Use the most recent revision no later than *date*. This option is sticky, and implies `-P'. See Section 4.5 [Sticky tags], page 32, for more information on sticky tags/dates.

`-f` Only useful with the `-D *date*`

`-j`*revision* With two `-j`

? *file* *file* is in your working directory, but does not correspond to anything in the source repository, and is not in the list of files for `cvs` to ignore (see the description of the `-I' option, and see Section C.9 [cvsignore], page 126).

Appendix B Quick reference to CVS commands

`-t-`*string* Set file description to *string*.

`-u[`*rev*`]` Unlock revision *rev*, or latest revision.

`annotate [`*options*`] [`

`-l` Local; run only in current working directory. See Chapter 6 [Recursive behavior], page 41.

`-R` Operate recursively (default). See Chapter 6 [Recursive behavior], page 41.

`export` [*options*] *modules*`:::`
Export files from CVS. See Section A.10 [export], page 93.

`-D`

`-o` Report on checked out modules. See Section A.11.1 [history options], page 94.

`-r` *rev* Since revision *rev*

`-r`*revs* Only list revisions *revs*

`rtag` [*options*] *tag modules:::*

Add a symbolic tag to a module. See Section A.16 [rtag], page 101.

`-a` Clear tag from removed files that would not otherwise be tagged. See Section A.16.1 [rtag options], page 102.

`-b` Create a branch named *tag*. See Section A.16.1 [rtag options], page 102.

`-D` *date* Tag revisions as of *date*. See Section A.16.1 [rtag options], page 102.

`-d` Delete the given tag. See Section A.16.1 [rtag options], page 102.

`-F` Move tag if it already exists. See Section A.16.1 [rtag options], page 102.

`-f` Force a head revision match if tag/date not found. See Section A.16.1 [rtag options], page 102.

`-l`

`unedit` [*options*] [*files*]
: Undo an edit command. See Section 10.6.3 [Editing files], page 60.

  `-a` *actions*

```
$ cvs co ampermod
cvs checkout: Updating first-dir
U first-dir/file1
U first-dir/file2
cvs checkout: Updating first-dir/sdir
U first-dir/sdir/sfile
$
```

Do not rely on this buggy behavior; it may get fixed in a future release of cvs.

C.2 The cvswrappers file

force it to check in the file anyway by specifying the `-f' option to cvs commit (see Section A.8.1 [commit options], page 89).

For another example, the following command imports a directory, treating files whose name ends in `.exe' as binary:

```
cvs import -I ! -W "*.exe -k 'b'" first-dir vendortag reltag
```

C.3 The commit support files

The `-i' flag in the `modules' file can be used to run a certain program whenever files are

C.4 Commitinfo

The `commitinfo' file defines programs to execute whenever `cvs commit' is about to execute. These programs are used for pre-commit checking to verify that the modified, added and removed files are really ready to be committed. This could be used, for instance, to verify that the changed files conform to to your site's standards for coding practice.

As mentioned earlier, each line in the `commitinfo' file consists of a regular expression and a command-line template. The template can include a program name and any number of arguments

If the edit script exits with a non-zero exit status, the commit is aborted.
Note: when

All occurances of the name `ALL

```
RCS     SCCS    CVS     CVS.adm
RCSLOG  cvslog.*
tags    TAGS
.make.state     .nse_depinfo
*~      #*      .#*     ,*      _$*     *$
*.old   *.bak   *.BAK   *.orig  *.rej   .del-*
*.a     *.olb   *.o     *.obj   *.so    *.exe
*.Z     *.elc   *.ln
core
```

`SystemAuth=`*value*

If *value* is `

`$CVS_PASSFILE`
: Used in client-server mode when accessing the `cvs login server`. Default value is `$HOME/.cvspass'. see Section 2.9.3.2 [Password authentication client], page 21

`$CVS_CLIENT_PORT`
: Used in client-server mode when accessing the server via Kerberos. see Section 2.9.5 [Kerberos authenticated

Appendix E Compatibility between CVS Versions

The repository format is compatible going back to cvs 1.3. But see Section 10.6.5 [Watches Compatibility], page 61, if you have copies of cvs

Appendix F Troubleshooting

If you are having trouble with cvs

`cvs [init aborted]: cannot open CVS/Root: No such file or directory`

This message is harmless. Provided it is not accompanied by other errors, the operation

`dying gasps from` *server* `unexpected`

There is a known bug in the server for `cvs` 1.9.18 and older which can cause this. For me, this was reproducible if I used the `-t' global option. It was fixed by Andy Piper's 14 Nov 1997 change to src/filesubr.c, if anyone is curious. If you see the message, you probably can just retry the operation which failed, or if you have discovered information concerning its cause, please let us know as described in Appendix H [BUGS], page 143.

If the error messages are not su cient to track down the problem, the next steps depend largely on which access method you are using.

:ext:

Appendix G Credits

Roland Pesch, then of Cygnus Support <roland@wrs.com> wrote the manual pages which were

Appendix H Dealing with bugs in CVS or this manual

Neither cvs nor this manual is perfect, and they probably never will be. If you are having trouble using cvs, or think you have found a bug, there are a number of things you can do about it. Note that if the manual is unclear, that can be considered a bug in the manual, so these problems are often worth doing something about as well as problems with cvs itself.

If you want someone to help you and x bugs that you report, there are companies which will do that for a fee. Two such companies are:

```
Signum Support AB
Box 2044
```

prefer, but there are not necessarily any maintainers reading bug reports sent anywhere except bug-cvs.

People often ask if there is a list of known bugs or whether a particular bug is a known one. The file bugs in the cvs source distribution is one list of known bugs, but it doesn't necessarily try to be comprehensive. Perhaps there will never be a comprehensive, detailed list of known bugs.

V

W