Inter@ctivate Consulting Group
2244b carmel valley road
del mar, ca 92014
619/793–4060
mailto:sean@interactivate.com
http://interactivate.com/cvswebsites/

CVS Version Control for Web Site Projects

sean dreilinger

April 13, 1998

Contents

1 Introduction

Version control is a special kind of software used to track and manage changes. In our case, CVS version control is used to track any sort of change made to our web sites, whether it's a single edit of one file to fix a typo, or a series of adjustments to a project where several files, folders, and graphics are added to (or removed from) the site.

In an *uncontrolled* site where multiple authors have access to edit and contribute, the potential for conflict and problems arises—more so when these authors work from different offices at different times of day and night. You may spend the day improving the file *index.html* for a customer. After you've made your changes, another developer who works at home after hours, or in another office, may spend the night uploading their own newly revised version of the file *index.html*, completely overwriting your work with no way to get it back! With the same site under CVS version

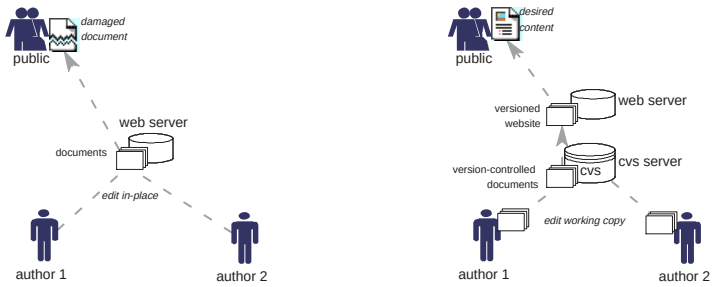

Figure 1: uncontrolled vs. controlled site development

control, the late–night author will be alerted to a conflict with the file *index.html*, presented with the exact parts of the *index.html* file that are causing a problem, and asked to adjust their work to incorporate anything you added and committed to the site while working on it earlier in the day.

If a customer needs to remove a recently added page or content area for legal reasons—or if they simply prefer an earlier version of their site—CVS can be used to restore the entire site to any previous state of their choosing, rolling back multiple variations and edits by all authors until a satisfactory site can be put back in place.

To work as a web author on a site under CVS version control, you install the CVS program and use it to transfer a working copy of the site (or subsite) to your desktop computer. Whenever your work is ready to *commit* to the live site, you issue a single instruction that checks for changes by other authors and uploads your work across the network to a central *site repository*. If there are potential *conflicts* between your work and another author, you are given a chance to resolve those problems by editing the files in question and sending them back. The CVS program also makes it easy to *update* your working copy of a site (or subsite) and *merge* any changes from the master copy—proactively preventing version conflicts.

The Apache HTTP Server Project[1] and the upcoming source release of Netscape Communicator[2] are two of many well–known computer programs with countless lines of code and dozens of authors who entrust the management of their work to a CVS–based system. Our needs, to manage a few web sites, are humble by comparison and CVS serves us well. Basic version control features are beginning to appear in the advanced (expensive, proprietary, and platform–specific) web authoring packages such as NetObjects TeamFusion and Macromedia DreamWeaver. Companies like Netscape Communications, Inc. have used CVS to manage their own web development since the early days.[3] Understanding version control, particularly CVS, is an essential ability in distributed collaborative work such as web development. Using CVS should improve your cooperation with other ICG consultants and add another bullet of impressive jargon to your resumé.

2 Software Setup

This section describes some one–time setup needed to prepare your computer and working environment to use CVS.[4] This assumes you have the use of a desktop computer running Macintosh, Microsoft Windows 95/NT, or some flavor of the UN⋆X operating system. Your computer must be able to connect to the Internet in order to exchange files with the site repository. Your Internet connection does not need to be dedicated or high–speed, although that is helpful.

2.1 Get CVS

CVS version control software is widely available (and free) for Macintosh, UN⋆X, Microsoft, and other operating platforms. To work as a web author and contribute work to one of our sites under version control, you need to obtain and install the CVS client software for your operating platform. You can get general information and updates from the Cyclic Software home page. Here are the URL's:

Cyclic Software Provides a home for CVS and commercial support: http://www.cyclic.com/. Download the newest release of CVS from Cyclic: http://download.cyclic.com/pub/.

Macintosh We recommend *MacCVSClient*, a special version of CVS for the Macintosh by Jörg Bullmann (joerg@fontworks.com): http://download.cyclic.com/pub/macintosh/

UN⋆X Source code and some binaries for UN⋆X users: http://download.cyclic.com/pub/cvs-1.9.26/

[1] "Almost all files relating to Apache, either the actual sources or the files that aren't part of the distribution, are maintained in a CVS repository." http://dev.apache.org/devnotes

[2] "We will be making the source available through other methods, including public (read-only) CVS servers" http://www.mozilla.org/download.html

[3] For an interesting account, read: *Lessons Learned Administering Netscape's Internet Site* at this URL: http://www.computer.org/internet/ic1997/w2toc.htm

[4] Except for logging in to the site repository (Section 2.5), which you may have to repeat when prompted on occasion.

Windows 95/NT Users can pick up the command–line *cvs.exe* binary here: http://download.cyclic.com/pub/cvs-1.9.26/windows/

Graphical Java Interface to CVS If you're ambitious and want to experiment, there is an excellent GUI (graphical user interface) for CVS available for computers that are setup to run Java programs.[5] For our requirements, the simple command–line client and three or four basic commands are all that you will need. The command–line client is much easier to set up, use and understand for the basic version control functions. If you've got non–billable hours to burn, you might like to explore this URL: http://www.ice.com/cvs/

2.2 Install CVS

For most platforms, you can find a binary (ready–to–run) copy of CVS to install. The software comes with installation instructions and an excellent, comprehensive user's guide in several formats. For PC and UN⋆X binary users, installation involves dropping the *cvs* or *cvs.exe* executable into a directory that is in your PATH environment, such as *c:/winnt/* or *~/bin/*.

MacCVSClient setup be familiar to Macintosh users. Check our local copy of the *MacCVSClient User Guide* for help getting started:[6] http://interactivate.com/cvswebsites/maccvsclient/

2.3 Configure Your Workspace

Once the CVS client program is installed and available on your system, you need to set up a few things to work with it smoothly. This workspace configuration can also be a one–time preparation if you know how to automate the setting of environment variables when you log on or start up your workstation.

Working Folder Create a folder or directory in which you'll work on version–controlled projects. We recommend a folder called *development* in your private space on the computer. In both UN⋆X and Microsoft Windows 95/NT, you can create such a folder by opening a command prompt window, changing to the desired disk location, and issuing the *mkdir* command. for example:

```
[u:\] mkdir development
[u:\] cd development
[u:\development]
```

CVSROOT Set your CVSROOT environment variable. If possible, set it once and for all using the Windows control panel, *autoexec.bat* file, or UN⋆X script executed on login, such as *~/.cshrc* or *~/.bashrc*.

Here's the command line you can use on a Windows 95/NT machine to set up the CVSROOT environment variable. Be sure to put your own username where the example below contains *username*.

```
[u:\] set CVSROOT=:pserver:username@interactivate.com:/usr/local/cvs-repository
```

[5] "Setup to run Java" means you've installed Sun's JDK for your platform, which is not difficult but is outside the scope of this writeup. See: http://java.sun.com/products/

[6] Can someone with a Macintosh tell me (sean@interactivate.com) how the setup works? Is there any special trick necessary? Is it a complete Macintosh Easy Install deal, or do you just unstuff one program file and click on it?

The few folks who choose to work directly from their **interactivate.com** UN⋆X account can relax, the CVSROOT variable is set for you automatically when you log in for a terminal session (like with *telnet* or *ssh*). If you are working on a project via a different UN⋆X box, you'll want to configure the CVSROOT environment variable to point across the Internet to our server.

Here's the command line you can use on a remote UN⋆X machine with a *bash* or *sh* shell to set up the CVSROOT environment variable. Be sure to put your own username where the example below contains *username*.

```
$ export CVSROOT=:pserver:username@interactivate.com:/usr/local/cvs-repository
```

If you use a *csh* shell or the enhanced c–shell *tcsh* under UN⋆X on a remote machine, the CVSROOT environment can be set with a command like this:

```
% setenv CVSROOT ":pserver:username@interactivate.com:/usr/local/cvs-repository"
```

2.4 Time Counts! Synchronize Your Clock

To help the CVS software better recognize and resolve conflicts (i.e. who wrote what, when), the CVS server computer needs to talk with client machines (your desktop computer) running an accurate, synchronized clock. With Internet access, it is possible to synchronize your computer's clock with one of many atomic–clock time servers around the world. You can learn more about the NTP protocol and how at works at this URL address: http://www.eecis.udel.edu/~ntp/

Free client software to keep your desktop computer in synch with a time–server is available from the same resource at this URL address: http://www.eecis.udel.edu/~ntp/software.html If you don't already use such a utility, you should select, install, and configure one that works with your operating system and type of network connection. Try using the Inter@ctivate server IP address as your time–server: **206.40.197.2**. If the IP address for ICG fails, try using the one we set our server's clock from: **clock.llnl.gov**. The client software tends to be small and easy to configure. If possible, set up time-synchronizing system service, daemon, or program that automatically loads when you start your computer.

2.5 Log In to the CVS Site Repository

To start exchanging files and information with the CVS server computer, you need to send your login credentials, which include the same username and password you use to get your email from **interactivate.com**. Here's how to log in to the CVS server at the command prompt:

```
$ cd development
cvs login (Logging in to sean@durak.dyn.ml.org) CVS
password:
$
```

Depending on your working environment, you may need to execute the `cvs login` command just once, or more often. CVS will provide an instructive message reminding you to login when it is necessary. The graphical CVS interfaces are pretty self explanatory, such as *jCVS* (Figure 2). If you're using CVS with our server for the very first time, you may experience an error message similar to:

```
cvs checkout: Sorry, you don't have read/write access to the history file
cvs [checkout aborted]: /usr/local/cvs-repository/CVSROOT/history:
Permission denied
```

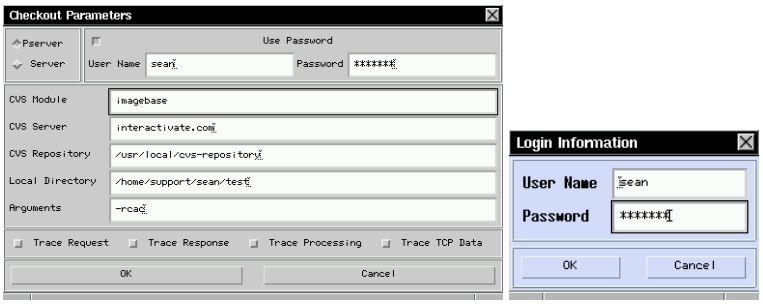

Figure 2: jCVS Login Dialogs

This is a dead giveaway the we have not added your username to the special *cvs* group on the CVS site server. Please send email to: help@interactivate.com and include a copy of the error message, including the command that got you there.

3 Creating a New Web Site From Scratch

Once in a *rare* while, you may be asked to create a new web site and *import* it into our CVS repository. This need only arises when no one else in our company has ever edited or created web documents or contributed to a site for a particular client. If you're sure this is the case (please don't hesitate to double–check!), you can use the `cvs import` process to put your work into the repository as the very first version of a web site. If any web–work already exists for the client or site you're interested in, skip directly to Section 4.

Web Development and Delivery Guidelines If you're not already familiar with the brief *Guidelines for W^3 Projects* for ICG contractors, please read this short list of guidelines for use in developing usable and maintainable web sites. You can view them online: http://www.interactivate.com/public/standards/

File Naming Scheme If you're creating a brand–new a site for a client, please create everything in a folder named for their *domain*. For example, if the customer's web site is accessed via the *host* name: **www.sandiegozoo.org**, then their *domain* name (and the name of the folder you should create and work within) is: *sandiegozoo.org*. Similarly, our client with a web site at URL: http://www.provenwinners.com/ is checked out and creates its own working folder named: *provenwinners.com*.

Build the Site Until you need to share your work on this brand–new site with the client or other authors, go ahead and develop the new site in the development area of your desktop computer. Unless you have a good reason not to, you should name your working folder with the client domain name convention described above.

Import the New Site When you're all set to share the first draft of your site and invite other authors to work on it, you *import* your work into the master CVS site repository. To do the *import*, change into the folder or directory that contains the root of your new site (such as *~/development/sandiegozoo.org/*) and issue a `cvs import` command, with the appropriate modifications to the "brief description" and client domain name as given in the example below.

```
$ cvs -q import -m"brief description" clientdomainname.com interactivate start
```

If you're a contractor with Inter@ctivate Consulting Group, you're welcome to replace the term *interactivate* in the example above with a one–word *tag* that identifies your own company, such as `cyberworks` or `studiomichael`. The "brief description" and details about any web work that you import via CVS will be instantly broadcast via email to a project listserv such as interact@interactivate.com. Please be sure to include a reasonable description of the project that can appear on the customer's invoice.

The `cvs import` procedure is *only* intended to get the very first version of a project onto the CVS site server. For updates and changes to an existing project or site, you'll use the *cvs update* process described in Section 5.1.

Once you have *imported* a project onto our CVS site server, you must *checkout* a *working copy* from the server and make any further modifications or changes to your *working copy* of the project files. Even though you may be the proud parent of the original documents, you should let CVS manage them for you forever after—starting with a fresh *checkout* (See Section 4).

4 Starting Work on an Existing Web Site

The first time *you* ever work on a client's existing, version–controlled web site, you'll need to "check out" a *working copy* of the site for use on your personal computer or in your personal workspace on the UN⋆X server.[7] To check out a *working copy* of a site for your use, change to your *development* folder and issue the *cvs checkout* command:

```
[u:\development] cvs -q checkout kawanhee.org
U kawanhee.org/activity.htm
U kawanhee.org/camp.htm
U kawanhee.org/counselor.htm
U kawanhee.org/act/archery.htm
U kawanhee.org/act/baseball.htm
U kawanhee.org/img/aca_logo.gif
U kawanhee.org/img/cabin2.jpg
U kawanhee.org/img/camp1.jpg
...
[u:\development]
```

This *cvs checkout* command will create a *working copy* of the site for you, including any necessary project folder, its sub–folders, and all associated files—*html*, *gif*, *jpg*, etc. You'll notice in the example above that the hypertext files in this site are named with *.htm* extensions, rather then the preferred *.html*. *Please don't repeat this inconsistent design decision in your own work!*

5 Editing a Web Site With Version Controlled Files

If you work on a particular version–controlled site regularly, you will have a *working copy* of all the files you need checked out and available on your desktop computer. You may edit the files in your *working copy*

[7] Version control software was developed with a *library* model in mind, where each computer file was treated like a unique library book—anyone editing the file had to "check it out" of the library, and before another user could edit the file, the current user had to "check in" the same file back to the library. Although CVS version control has evolved well beyond this original library metaphor, it may help to understand some of the original terminology as you work with it. *MacCVSClient* uses "check in" as a preferred synonym for "commit."

at will, without additional setup or hassle. It is always wise to begin an editing session by making sure your working copy of the site is up–to–date and in tune with the latest copy available from the CVS site repository (especially if you haven't touched anything for a day or two—others may have committed changes to the site in your absence).

5.1 Updating Your Working Copy of a Site

To bring your working copy of a site up–to–date with the CVS site server, use the `cvs update` command to automatically compare your working copy with the site repository and copy across any files that have changed since you last worked with them. To *update* your working copy, change into the folder that contains your working copy of the site and issue the *cvs update* command, like this:

```
$ cd ~/development/interactivate.com/apps
$ cvs -q update -dP
? analog/.htaccess
U auth/htpasswd
U hours/index.phtml
U hours/data/icg-timelog-1998-01.txt
...
$
```

That's it! if there are new files available, you've now got a copy on your desktop, available for review and editing. There's really nothing special you need to do or change about your web development working habits—just edit and create files as you normally would. Use whatever HTML editor and creation tools you're comfortable with.

5.2 Cross–platform and Consistency Considerations

There are a few *precautions* you must observe to ensure the site is usable on all platforms and in situations beyond the web–server, such as when loaded onto a laptop for a trade show demonstration. Here are the precautions:

CVS folders The folders in each directory called *CVS* contain important version control information that is automatically generated. You should not remove these folders or edit their contents by hand.

No Symbolic Links UN⋆X developers may be tempted to create symbolic links between files. To assure that developers (and web servers) on non–UN⋆X platforms can work with the project, *do not create symbolic links* in your working copy of the site. If you do create symbolic links and upload them to the site repository, they will be silently ignored.

Naming Conventions As mentioned in our *Guidelines for W^3 Projects*, limit filenames to the Macintosh–friendly length of 32 characters. Do *not* use *MiXEd-cAsE*, *under_scored*, or *Capitalized* names for files or folders. Use a simple-hyphen to separate *multi-word-filenames*.

Relative URL's Within any HTML/hyperlinks you create, use *relative* URL addresses (such as *../classes/cvs-101.html*) which do not begin with an absolute path. *Absolute* URL addresses (*/school/classes/cvs-101.html*) are much pickier about where and how they are served, and make it difficult to port our web projects onto other computers.

5.3 Adding and Removing Files

In the course of editing the files in a site, you may choose to create (*add*) or delete (*remove*) any number of files and folders in your *working copy* of the site. You can play with your working copy of the site to your heart's content, even *commit* changes made to existing files and folders—but *nothing* will be created on or removed from the CVS site server until you issue a very specific instruction to *add* or *remove* a list of filenames.

5.3.1 Adding Files

To *add* files to the CVS site repository, first create or add them to your *working copy* of the project, and edit these new files to your liking. Same goes for directories (folders). Just create what you need within your working copy of the site. If your project follows any special navigation structure or naming scheme beyond our *Guidelines for W^3 Projects* for ICG contractors, you'll want to make sure you follow those in the creation of new files.

Once your newly created files and folders are tested and ready in your *working copy* of the site, use the `cvs add` command to schedule the filenames for upload to the CVS server:

```
$ cvs add newfile.html
cvs server: scheduling file 'newfile.html' for addition
cvs server: use 'cvs commit' to add this file permanently
$
```

This will *not* send the actual files to the CVS server yet, it simply make a note to alert the server to expect these files the next time you *commit* the project (Section 6).

5.3.2 Removing Files

To *remove* files from a site, you run the `cvs remove` command on the desired filenames in your *working copy*. As a "safeguard", `cvs remove` will *not* work if the files still exist in your working copy:

```
$ cvs remove oldfile.html
cvs server: file 'oldfile.html' still in working directory
cvs server: 1 file exists; remove it first
$
```

To get around this, you may use the `-f` option with the `cvs remove` command:

```
$ cvs remove -f oldfile.html
cvs server: scheduling 'oldfile.html' for removal
cvs server: use 'cvs commit' to remove this file permanently
$
```

This will *not* delete the actual files from the CVS server yet, it simply makes a note to tell the server to remove these files the next time you *commit* your working copy of the project (Section 6). Although you will remove the selected files from the latest *version* of the site when you *commit* your changes, the "removed" files are still available from CVS site server, to anyone who knows how to request an older *version* of the site (Section 9).

When you're finished editing, adding, and removing files, you must remember to *commit* your work to the CVS site repository—especially if you have completed your task or have reached an important milestone you wish to share with the other site authors. Section 6 shows how to *commit* your work to the site repository.

6 Committing Your Work to the Site Repository

When you're finished editing, *test your work by previewing it in a web browser!* Once you're confident that your changes and improvements on a *working copy* of a site are ready for the live web site (or for review and use by other developers), *commit* your work to the site repository with the `cvs commit` command:[8]

```
[u:\development]cvs commit
cvs commit: Examining .
cvs commit: Examining camp-kawanhee
cvs commit: Examining camp-kawanhee/act
cvs commit: Examining camp-kawanhee/img

CVS: ----------------------------------------------------------------------
CVS: Enter Log.  Lines beginning with 'CVS:' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:    camp-kawanhee/activity.htm camp-kawanhee/camp.htm
CVS: ----------------------------------------------------------------------
i removed the animated mailbox icon that barfs a parcel twice per second
the nav menu now simply says: for more information: info@kawanhee.org

Checking in camp-kawanhee/activity.htm;
/usr/local/cvs-repository/camp-kawanhee/activity.htm,v  <--  activity.htm
new revision: 1.4; previous revision: 1.3
done
Checking in camp-kawanhee/camp.htm;
/usr/local/cvs-repository/camp-kawanhee/camp.htm,v  <--  camp.htm
new revision: 1.2; previous revision: 1.1
done

[u:\development]
```

6.1 Entering a Log Message

The *cvs commit* process recognizes which files have changed and prompts you to enter a description of your work. This *log message* becomes available to other developers so they can better understand why you might have changed something in the site. In fact, your log message is instantly broadcast via email to the interact@interactivate.com listserv and other relevant recipients. Everyone's log messages can be combined later to provide the customer with a detailed account of work we've performed for them to date. Please take this into consideration and spend an extra minute to record an accurate and descriptive log message when prompted.

When CVS needs you to enter a log message, the *cvs commit* process starts up your preferred text editor with some instructions pre–loaded into the editing space. In UN⋆X, your choice of editor is set via the `EDITOR` environment variable. On Windows NT, it defaults to the *notepad* text editor. [9]

[8] Having trouble finding a "commit" command in *MacCVSClient*? Remember that "check in" can be used as a synonym for "commit."

[9] Anyone know how this log message entry works on a Macintosh? *SimpleText*?

6.2 I've Committed, I'm Done, Right?

Your committed work is not *automatically published to the customer's live web site. See Section 8 for a description of how to get your modifications to appear on the customer's web site.*

Once your work is *committed* to the site repository, it is available for publishing to the live web site or for other developers and authors to *check out* and work with. In our present setup at Inter@ctivate Consulting Group, *your committed work is not automatically published to the customer's live web site*. Until we're all comfortable with the version control process, changes you commit to the site repository are not directly published onto the web. See Section 8 for a description of how to get your modifications to appear on the customer's web site.

6.3 Resolving Conflicts

We've lost hours, even days of work on web sites when one developer has overwritten the work of another by uploading files without checking for improvements made by the other project authors. Along with good planning and communication, version control helps prevent web development disasters. This section shows you how CVS version control will help you prevent or resolve any development conflicts when uploading your work to the site repository.

When you enter the `cvs commit` command to automatically upload all the files you have changed or added to a site, the site repository server may inform you that your locally–edited files are not up–to–date with the server or that *you* need to manually *merge* one or more files with newer versions that have already been uploaded to the repository by a colleague. Here's a sample *conflict* warning message that occurred during a `cvs commit` process:

```
$ cvs commit
cvs commit: Examining .
cvs commit: Up-to-date check failed for ‘activity.htm’
cvs commit: Up-to-date check failed for ‘camp.htm’
cvs commit: Up-to-date check failed for ‘index.htm’
...
cvs [commit aborted]: correct above errors first!
```

You can use the `cvs update` command to bring your local copy of the site up–to–date with the repository *and* attempt to automatically *merge* your latest work with changes that other web developers may have already made to the site. To *update* your entire *working copy* of the site, open a command prompt, change to the directory containing the site you're developing, and issue the command: `cvs update`. This will update and automatically merge every file that has changed since you last copied over new files from the site repository. Line–by–line updates to individual text files (such as HTML files) can often be handled automatically. CVS will list for you any files that require your attention for manual editing and merging. Let's walk through one *automatic merge* and one *manual merge* as examples.

6.3.1 Automatic Merge

You've edited the home page to refine the "`KEYWORDS`" `META` tag and prepend an `HTML DTD` element. When you attempt to *commit* your newly edited *index.htm* file to the site repository with the *cvs commit* command, you are told:

```
$ cvs commit index.htm
cvs commit: Up-to-date check failed for ‘index.htm’
cvs [commit aborted]: correct above errors first!
```

You can tell that the master copy of this site stored in the site repository must have acquired some new content, so you use the *cvs update* process to refresh your working copy of the web site. In the example

below, we instruct the CVS server to update just the *index.htm* file as a test case, with the command line: `cvs update index.htm`:

```
$ cvs update index.htm
RCS file: /usr/local/cvs-repository/camp-kawanhee/index.htm,v
retrieving revision 1.4
retrieving revision 1.5
Merging differences between 1.4 and 1.5 into index.htm
M index.htm
```

In this case, the lines you added to the *index.htm* file (near the top of the HTML file) and changes made by other developers (elsewhere in the file) were automatically and successfully merged. The *index.htm* file in your working copy of the site (and only your working copy) now contains all the changes you wanted to make to the file *and* all the that have been made and committed recently by other authors. The CVS program managed to assemble the two versions of the file into one that should work to everyone's satisfaction. You should still open the *index.htm* file in a web browser to make sure the automatic merge did the right thing. When the *index.htm* file works to your satisfaction, use `cvs commit` to put it into the site repository:

```
$ cvs commit index.htm
Checking in index.htm;
/usr/local/cvs-repository/camp-kawanhee/index.htm,v  <--  index.htm
new revision: 1.6; previous revision: 1.5
done
```

6.3.2 Manual Merge

In other cases, your recent work on a file might be so different that the CVS needs your help to integrate everyone's work and put it back into the site repository. In this case, CVS sends you back a hybrid version of the file(s) in question, containing special markers to call your attention to differences between your latest work and changes other developers have made to the same file. This section walks you through such a *manual merge*.

Imagine you've removed some unnecessary HTML <SPACER> tags remaining from an earlier version of a file named *activity.htm* on the site. Meanwhile, a coworker has removed all sorts of gratuitous HTML `<!-- comments -->` from the same file, pursuing a similar clean–up assignment of their own. When you go to commit your work into the site repository, you see your coworker has already finished their comments–clean–up and that your work is in conflict:

```
$ cvs commit activity.htm
cvs commit: Up-to-date check failed for 'activity.htm'
cvs [commit aborted]: correct above errors first!
```

Use the `cvs update` command to bring your local copy of the site up to date and attempt to automatically merge any changes from other authors:

```
$ cvs update
cvs update: Updating .
RCS file: /usr/local/cvs-repository/camp-kawanhee/activity.htm,v
retrieving revision 1.5
retrieving revision 1.6
Merging differences between 1.5 and 1.6 into activity.htm
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in activity.htm
C activity.htm
```

In this case, CVS could not automatically merge your work with changes already made by the other author(s). Instead, the CVS program has created a special version of the file in *conflict* and put it in your working copy of the site in place of the original *activity.htm*.

If you open this file in your HTML editor, you'll find a special marker inserted on a line by itself to show you where the editing conflict begins. This special marker line always begins with a series of eight less–than symbols, followed by the filename:

```
<<<<<<<< filename
```

This marker is followed by your version of the text in question, then another marker—this time a series of eight equals signs on a line of their own:

```
========
```

Below this marker comes another version of the text in question that already exists in the site repository. The end of the text–in–conflict is denoted by a final marker, a series of eight greater–than symbols and a version number on a separate line:

```
>>>>>>>> X.y
```

You must edit this file until your contribution and any changes made by the other author(s) all work smoothly together. It may be a simple matter of formatting, or a complex content issue that you must resolve with other members of the project in a phone call or meeting. In our simple example, you've removed HTML <SPACER> tags from a file, while another author has removed some <!-- link... --> comment tags from the same file. This is what you see upon opening the file in an editor:

```
...
    <p align="left">Which activity are you interested in? <br>
    <br>
<<<<<<< activity.htm
<!-- Link Tag -->    <a href="act/sail.htm">Sailing</a> <br>
<!-- Link Tag -->    <a href="act/baseball.htm">Baseball &amp; Softball</a> <br
<!-- Link Tag -->    <a href="act/basket.htm">Basketball</a> <br>
=======
<spacer type="HORIZONTAL" size="100"><a href="act/sail.htm">Sailing</a> <br>
<spacer type="HORIZONTAL" size="100"><a href="act/baseball.htm">Baseball &amp;
<spacer type="HORIZONTAL" size="100"><a href="act/basket.htm">Basketball</a> <b
>>>>>>> 1.6
    </p>
    <hr align="center">
...
```

To resolve the conflict, simply edit the *activity.htm* file and replace the text between the markers and test the file until it works. You should also delete the markers <<<<<<<<<========>>>>>>>> from the file. Here's an excerpt from the edited, tested, and successfully working version of the file–in–conflict above:

```
...
    <p align="left">Which activity are you interested in? <br>
    <br>
    <a href="act/sail.htm">Sailing</a> <br>
    <a href="act/baseball.htm">Baseball &amp; Softball</a> <br>
    <a href="act/basket.htm">Basketball</a> <br>
    </p>
    <hr align="center">
...
```

Edit any other files that require your attention to resolve a conflict, test your working copy of the site, and use the `cvs commit` command to put your latest and greatest work back onto the site repository:

```
$ cvs commit
Checking in activity.htm;
/usr/local/cvs-repository/camp-kawanhee/activity.htm,v  <--  activity.htm
new revision: 1.7; previous revision: 1.6
done
```

Remember that conflicts requiring manual editing are rare. The explanation is complicated, but CVS does a great job of recognizing and correcting most of the minor editing collisions without human assistance. When there is a problem, you'll be glad to catch it before overwriting another person's hard work.

7 Keeping Track of Who is Doing What

CVS allows several people to alter the same site, even the same file, at once.[10] You should *update* your working copy of a project regularly so that you'll never miss a major change. In addition to requesting regular CVS *updates*, you may want to receive a "heads–up" via email when someone else changes the site, and then go in and browse those changes online (instead of downloading a fresh *working copy* and trying to guess what changed by reading the source of each file). Section 7 describes the automatic email log messages and *cvsweb* tool that we've started using at ICG to help keep track of who is working on a project and what they're up to.

7.1 Automatic Email Updates

Each of our web-development and software projects has an email list. When you use `cvs commit` to contribute work to the site repository, everybody on the listserv for that project receives a copy of your log message via email. The message looks like something like this:

```
        From: uyen nguyen <uyen@interactivate.com>          Wed 08:37
     Subject: 'californiamart.com/html/Directory' update now available
Resent-From: interact@interactivate.com
         To: interact@interactivate.com

Update of /usr/local/cvs-repository/californiamart.com/html/Directory
In directory server1:/mnt/hdb2/www/californiamart.com/html/Directory

Modified Files:
        CLASSIFY.TXT LINES.TXT REPS.TXT index.html

Log Message:
update of directory listings (new REPS, LINES & CLASSIFY)
```

If you are *not* receiving similar email messages when you commit your changes to the site repository, you can write to help@interactivate.com and ask to be added to the listserv for the site(s) you're working on.

These automagic email updates work well to inform everyone the minute changes to a site or project are available. If you receive such a message for a project you are working on, take it as a strong hint to update your working copy of the site by running `cvs update` as soon as possible.

[10] The 'C' in CVS stands for *concurrent*.

7.2 Browsing Changes in Technicolor

Sometimes you *know* a file was changed by another developer, but aren't sure exactly what has changed. *cvsweb* color–codes changes to files in a CVS site repository and displays the old and new versions side–by–side, line–by–line for easy comparison. *cvsweb* for our site repository begins via this URL address: http://interactivate.com/apps/cvs/ *cvsweb* presents you with a web–based interface to browse any and all of the

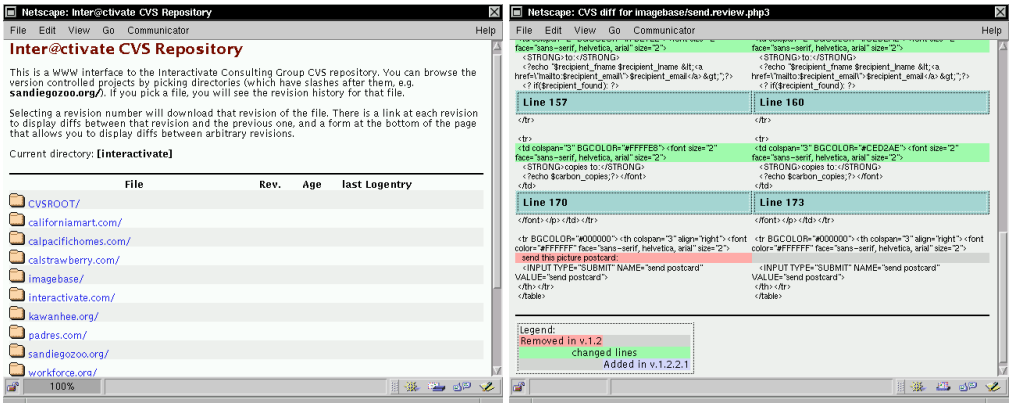

Figure 3: cvsweb shows the differences between two versions of an HTML file

sites and projects we manage via CVS.[11] We've found *cvsweb* especially helpful for:

Visual Comparison Compare the latest version of any file (web page, cgi script, *.htaccess*, etc.) to previous versions of the same file.

Reports and Invoices Non–technical folks can use *cvsweb* to call up log messages that contain human–readable change notes from the developers, including the exact date and time that each site update was performed.

Which One? Developers can use *cvsweb* to figure out exactly which *version*, *branch*, or *tag* of a project they want to work with. Some of our projects have sprouted multiple *branches*, each with similar files but personalized, per–customer content that must be checked in and out via its own tagged "branch."

Work Confirmation Delivery of subcontracted work can be reviewed via *cvsweb* as soon as the developer *commits* their work to the repository. If you are active in a particular site or project, you should receive email each time a change is committed.

8 Publishing Completed Work to the Web

For the time being, our published (live) web sites are themselves checked–out copies of the site repository hosted by Inter@ctivate Consulting Group. At the expense of filling the customer's site with extra folders called *CVS*, we add some insurance in case a developer forgets what they're doing and edits the web site directly (*never* edit a web site directly on the customer's server if you can avoid it).

[11] We currently handle ten web–related projects with CVS: seven customer sites, our own site, and two software development projects.

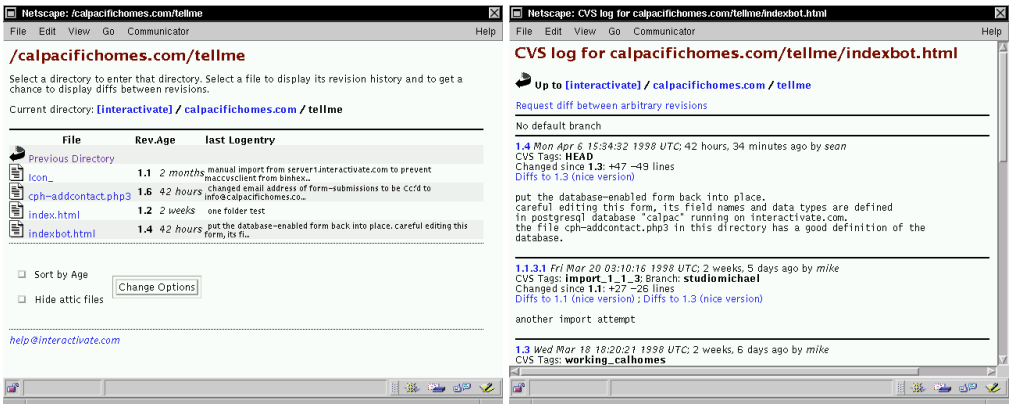

Figure 4: *cvsweb* lets non–technical users access human–readable log messages

When you're ready to bring a customer's "live" site up to date with any recent work in their site repository, *ssh* or *telnet* into their server (or *cd* into their server folder if they live on our ICG machine) and issue the `cvs update` command to publish any changed files.

If you make a mistake and edit files directly on the customer's "live" server, promptly issue the appropriate `cvs commit` instruction to register your changes with the site repository. In such a case, we'll lose some of the author information, so please include your name in the log message. It is possible to automate web–publishing via the `cvs export` feature, we'll experiment with this once everyone is consistently working with CVS version control.

9 Rolling Back Web Site Changes and Versions

Have you ever wanted to pull up last year's version of a web site to show yourself, your development team, the client, or a trade show audience the evolution of a site over time? A site maintained under CVS version control makes this easy to do. Just as a library often maintains older editions of a book for readers to check out, a CVS version control repository tracks every variation of each web site and allows you to make a working copy of the site as it appeared at any point in history! Here is a sample CVS command to retrieve a copy of a web site as it appeared one year ago today:

```
$ cvs checkout -D '1 year ago' provenwinners.com
```

A site under CVS version control can quickly and conveniently revert to an earlier stage of its life. If the lawyers suddenly get around to reading the new bulletin board area and determine it must be taken offline immediately, CVS can be used to quickly restore the previous preferred version of the site—even if the developers who created the bulletin board module are not available to put things back. For example, a site might be running with full legal approval at version 1.4. Then an outside consultant develops a bulletin board application, tests it, and checks it into the site repository, automatically updating the site repository to version 1.5.

When the legal team determines that the bulletin board application is a liability, the site can quickly be restored to the legally compliant version, without the need to track down and re–hire the outside consultants who developed the bulletin board. Here is a sample CVS command to restore a particular version of a web

site to active duty:

```
$ cvs checkout -r1.4 provenwinners.com
```

There are many subtle controls you have at your disposal when labeling (tagging) existing project states or calling up older versions of a site or project to work with. The two examples here should be enough to get you started and solve the two most common needs. For more information see the cvs documentation, available on the UN⋆X server by typing `man cvs` or on the web at this URL address: http://interactivate.com/cvswebsites/cvs˙toc.html

10 After a Project

When you make a working copy of a site, the CVS system makes a note that you're working on it. When you're done contributing to a project, it's polite to tell the CVS site repository that you are not using your working copy of the site anymore. This is accomplished via the `cvs release` command, which has the added benefit of alerting you to any changes you might have made to your *working copy* but forgotten about before you could *commit* them to the site repository. Here's an example of running the `cvs release` command and learning that you may have some additional new content to upload:


```
$ cvs release provenwinners.com
M index.shtml
You have [1] altered files in this repository.
Are you sure you want to release directory 'provenwinners.com': n
** 'release' aborted by user choice.
```

You may then take additional steps to commit your altered file to the site repository before releasing it once and for all (probably by just typing `cvs commit` on the spot). Here's the same *cvs release* situation with no hassle, no new files from your working copy to add to the site repository:

```
$ cvs release provenwinners.com
You have [0] altered files in this repository.
Are you sure you want to release directory 'provenwinners.com': y
```

It is now safe (and wise) to delete your working copy of the site, until you need to work with it again (which you can easily do so via the `cvs checkout` process).

11 CVS Administration Notes

Here are a few quick notes to help web and CVS administrators set up the services that are described throughout this document.

11.1 CVS Server Setup

Crash course in setting up client–server CVS on a Linux UN⋆X server. This will authenticate users via the */etc/passwd* file.

1. Install CVS, should compile out of the box on Linux and produce no errors when you run `make check`.
2. As root, edit */etc/inetd.conf* to add the following (all on ONE line):

```
cvspserver stream tcp nowait root /usr/local/bin/cvs cvs
          --allow-root=/usr/local/cvs-repository pserver
```

3. As root, edit */etc/services* to confirm or add the following:

```
cvspserver     2401/tcp     # CVS remote server function
```

4. As root, issue the command `killall -HUP inetd`
5. Log on to some other machine and test the access using the `cvs login` command.

11.2 Handling Binary Files

CVS regards the files you put into it as plain–text source code. That works well for *.html* files, but causes more than a little trouble when you're trying to *import* and manage files that must be maintained in a binary format, such as the *.gif* and *.jpg* image files commonly found in web sites. You can configure the CVS server to recognize certain file extensions as binary and check them in to and out of the repository with extra care. Here's how we set up ours:

1. Make yourself a working copy of the *CVSROOT* configuration files. Change into a working directory and issue the command:

```
$ cvs co CVSROOT
```

2. Edit the file *CVSROOT/cvswrappers* to tell CVS how to handle each file extension. Here's the *cvswrappers* file we use:

```
*.avi   -k 'b' -m 'COPY'
*.doc   -k 'b' -m 'COPY'
*.exe   -k 'b' -m 'COPY'
*.gif   -k 'b' -m 'COPY'
*.gz    -k 'b' -m 'COPY'
*.hqx   -k 'b' -m 'COPY'
*.jar   -k 'b' -m 'COPY'
*.jpeg  -k 'b' -m 'COPY'
*.jpg   -k 'b' -m 'COPY'
*.mov   -k 'b' -m 'COPY'
*.mpg   -k 'b' -m 'COPY'
```

```
*.pdf   -k 'b' -m 'COPY'
*.png   -k 'b' -m 'COPY'
*.ppt   -k 'b' -m 'COPY'
*.sit   -k 'b' -m 'COPY'
*.swf   -k 'b' -m 'COPY'
*.tar   -k 'b' -m 'COPY'
*.tgz   -k 'b' -m 'COPY'
*.tif   -k 'b' -m 'COPY'
*.tiff  -k 'b' -m 'COPY'
*.xbm   -k 'b' -m 'COPY'
*.xls   -k 'b' -m 'COPY'
*.zip   -k 'b' -m 'COPY'
```

3. Commit your changes to the CVS repository:

```
$ cvs commit -m"added common binary files"
cvs commit: Examining .
Checking in cvswrappers;
/usr/local/cvs-repository/CVSROOT/cvswrappers,v  <--  cvswrappers
new revision: 1.4; previous revision: 1.3
done
cvs server: Rebuilding administrative file database
```

11.3 Automatic Email Notification of Changes

To receive a copy of the log message via email with every change to a project, specify an email command in the *CVSROOT/loginfo* file. We use a dedicated email list for sites where changes are fast and furious, and send all other commit email to a general listserv.

1. Make yourself a working copy of the *CVSROOT* configuration files. Change into a working directory and issue the command:

```
$ cvs co CVSROOT
```

2. Edit the file *CVSROOT/loginfo* and indicate which module names or regular expressions should generate an email when an update is committed to the CVS repository, and the appropriate *Mail* command. Here's the *loginfo* file we've got in place:

```
^californiamar  /usr/bin/Mail -s "%{} update now available" interact
^calpacifichom  /usr/bin/Mail -s "%{} update now available" calpac
^calstrawberry  /usr/bin/Mail -s "%{} update now available" calstraw
^interactivate  /usr/bin/Mail -s "%{} update now available" interact
^padres         /usr/bin/Mail -s "%{} update now available" padres
^workforce      /usr/bin/Mail -s "%{} update now available" interact
^imagebase      /usr/bin/Mail -s "%{} update now available" imagebase
### Anything that doesn't get noticed above, send to the CVS-Admin to check
DEFAULT         /usr/bin/Mail -s "%{} unknown CVS commit" sean@interactivate.com
```

3. Commit your changes to the CVS repository:

```
$ cvs commit -m"updated email-on-commit"
cvs commit: Examining .
Checking in loginfo;
```

```
/usr/local/cvs-repository/CVSROOT/loginfo,v  <--  loginfo
new revision: 1.13; previous revision: 1.12
done
cvs server: Rebuilding administrative file database
```

11.4 Getting *cvsweb*

The *cvsweb* software touted in this document is a hacked up version of the original. *cvsweb* was created by Bill Fenner (fenner@freebsd.org) and the Color–diff and other improvements come from Henner Zeller (zeller@think.de). The improved *cvsweb* consists of a Perl CGI script and a little Perl library, available from:
http://lemming.stud.fh-heilbronn.de/~zeller/download/cvsweb.tar.gz
cvsweb will run with minimal reconfiguration, you just tell it where to find your CVS repository and away you go.

12 Quick Reference Sheet: CVS For Web Projects

CVS PROCESS	WHAT IT DOES	COMMAND LINE

LOGIN	Sign on to the CVS server so you have permission to check in and check out web sites. Run `cvs login` from your *development* folder.	`cvs login`
		
		`cd ~/development/`
		`cvs login`

IMPORT	Send a whole new site up to the server. `cvs import` is **not used very often**. Run `cvs import` from inside the root *htdocs* folder on the server/site to be imported.	`cvs import -m'message' clientdomain vendorname start`
		
		`telnet www.sandiegozoo.org`
		`cd htdocs/; cvs login`
		`cvs import -m"import the zoo" sandiegozoo.org interactivate start`

CHECKOUT	Grab a working copy of a site (or any part of a site) and put it onto your desktop computer. `cvs checkout` gets the most recent version of the site by default, but can also be used to checkout previous versions of a site or file. Run `cvs checkout` from your *development* folder. A subfolder will be created to contain each site that you checkout.	`cvs checkout clientdomain`
		
		`cd ~/development/`
		`cvs checkout kawanhee.org/index.html`

UPDATE	Freshen your working copy of a site. Adds any new files from the CVS server to your working copy, tries to automatically merge changes. You can *update* your working copy to turn it into any previous version of the site. `cvs update` defaults to the latest version. Run `cvs update` from inside the folder you would like to *update*.	`cvs update`
		
		`cd ~/development/imagebase/`
		`cvs -q update -rdwiggins -dP`

ADD	Schedule a list of newly created files that exist in your working copy to be added to the CVS site server the next time you *commit*.	`cvs add file1 file2...`
		
		`cd ~/development/calstrawberry.com/`
		`cvs add new1.html olddir/new2.html newdir`

REMOVE	Schedule files to be removed from the CVS server the next time you *commit* this site. `cvs remove` can be run from any location inside your working copy of a site.	`cvs remove file1 file2...`
		
		`cd ~/development/padres.com/`
		`cvs remove -f old1.html dir/old2.html olddir`

COMMIT	Send changes made to files in your *working copy* up to the CVS server. Run `cvs commit` from your *working copy* folder.	`cvs commit file1 file2...`
		
		`cd ~/development/californiamart.com/`
		`cvs commit`

RELEASE	Sign out with the CVS server, make sure there are no loose ends hidden within your working copy. Run this command from your development folder.	`cvs release clientdomain`
		
		`cd ~/development/`
		`cvs release workforce.org/`

CVSWEB	Web–based tool to browse our CVS site server. You can view everyone's changes and improvements on a file–by–file basis.	http://interactivate.com/apps/cvs/ http://interactivate.com/cvswebsites/