CVS Best Practices

Vivek Venugopalan

<vivekv at yahoo dot com>

Revision History

Revision 0.7	2005−10−15	Revised by: vv
A bunch of minor fixes as suggested by readers.		
Revision 0.6	2002−09−10	Revised by: vv
Added content related to tagging and daily builds. Changed Linuxdoc URLs to tldp. Fixed stale links and added other corrections suggested by readers.		
Revision 0.5	2002−08−25	Revised by: vv
Fixed some more errors in the document and added references to other CVS sources and some server side scripting		
Revision 0.4	2002−03−10	Revised by: vv
Added new email address, Added an example flow to show how the practices help		
Revision 0.3	2001−12−06	Revised by: vv
Grammatical errors cleanup		
Revision 0.2	2001−11−27	Revised by: vv
Incorporated first round of feedback and some minor fixes		
Revision 0.1	2001−11−20	Revised by: vv
Created		

Table of Contents

Table of Contents

1. Introduction

> Men have become the tools of their tools.
>
> −−Henry David Thoreau (1817−1862)

This article outlines some of the best practices that can be adopted when Concurrent Versions System is used as the configuration management tool in your software project.

Concurrent Versions System (CVS) is an Open Source configuration management tool that is now being looked at seriously by many commercial organizations as a viable alternative to other commercial Software configuration management tools.

This spotlight on CVS has led to the inevitable question of best practices for deploying CVS as the backbone SCM tool for large software development projects. Having answered this question many times verbally as a bunch of "gotchas" on CVS, it was time to put down on paper some of the best practices that will work well for CVS based projects.

This paper assumes that the reader is familiar with the fundamentals of software version control. Including features like branching, merging, tagging (labelling) etc., offered by modern version control tools such as CVS

Further, This paper is not an introduction to CVS and its usage. There are excellent articles available on the net for the same. This paper assumes that the reader is familiar with CVS commands and is looking at deploying CVS in his or her organization. Some of the popular CVS related links that can provide CVS education are.

1. The Concurrent Versions System site where current informaton about CVS is available. Including the CVS manual.
2. Karl Fogel's book, Open Source Development with CVS is available online.

1.1. Copyright Information

This document is Copyright © 2001 Vivek Venugopalan. Permission is granted to copy, distribute and/or modify this document under the terms of the *GNU Free Documentation License*, Version 1.1 or any later version published by the Free Software Foundation with no Invariant Sections, no Front−Cover Texts, and no Back−Cover Texts. A copy of the license can be found in Appendix A.

This document may be reproduced and distributed in whole or in part, in any medium physical or electronic, as long as this copyright notice is retained on all copies. Commercial redistribution is allowed and encouraged; however, the author would like to be notified of any such distributions.

All translations, derivative works, or aggregate works incorporating this document must be covered under this copyright notice. That is, you may not produce a derivative work from this document and impose additional restrictions on its distribution. Exceptions to these rules may be granted under certain conditions; please contact the author at the address given below.

In short, we wish to promote dissemination of this information through as many channels as possible. However, we do wish to retain copyright on the document, and would like to be notified of any plans to redistribute the same.

1.2. Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and other content at your own risk. As this is a new edition of this document, there may be errors and inaccuracies that may of course be damaging to your system. Proceed with caution, and although this is highly unlikely, the author(s) do not take any responsibility whatsoever.

All copyrights are held by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals.

1.3. New Versions

This document is Version : 0.7.

The latest version of this document can be obtained from (In the order of latest version availability)

1. My website
2. The linux documentation project

1.4. Credits

The list of people who have provided information and correction for this paper in no particular order are.

1. Jens−Uwe Mager
2. Jorgen Grahn
3. Thomas S. Urban
4. Cam Mayor
5. Sally Miller
6. Niels Jakob Darger

1.5. Feedback

Feedback is most certainly welcome for this document. Without your submissions and input, this document wouldn't exist. Please send your additions, comments and criticisms to the following email address : `<vivekv at yahoo dot com>`.

2. Focus Areas

The focus areas for best practice are

1. GUI Tools
   - Use GUI CVS client
2. Developer Sandbox
   - Keep System clocks in Sync
   - Do not share the sandbox
   - Stay in sync with the repository
   - Do not work outside the sandbox
   - Cleanup after Completion
   - Check−in Often
3. CVS Server Configuration
   - CVS access control
   - Server side scripting
   - Server Notification
4. Branching and Merging
   - Assign ownership to Trunk and Branches
   - Tag each release
   - Create a branch after each release
   - Make bug fixes to branches only
   - Make patch releases from branches only
5. Change Propagation
   - Merge branch with the trunk after release
6. Software Builds
   - Build Early and Build Often
   - Automate build Process completely
   - All necessary files must be checked−in before build
7. Institutionalize CVS in the Organization
   - Implement Change Management Process
   - Make CVS Usage part of Objectives
   - Collect metrics on CVS usage

---

3. Using GUI Tools

The traditional interface available for CVS is the command–line client. There has also been a slew of GUI client applications that can "talk" to a CVS server. These GUI clients provide a "point and click" interface to the CVS repository.

3.1. Use GUI CVS client

This paper recommends using such GUI clients during the initial deployment of CVS in an organization.

Developers typically use integrated development environments that have the CM tools integrated into them. These tools minimize the learning for the developers about the intricacies of CVS usage and instead allow them to be productive from day one. Developers who are accustomed to other CM tools will find the CVS command–line interface daunting. The adoption and usage of CVS can be improved by using GUI tools for CVS clients.

GUI tools for CVS are available at http://cvsgui.sourceforge.net/. GUI interfaces are available for most of the popular platforms (Windows, Mac and Linux). In addition, on the Windows platform there is an SCC extension that allows integration of CVS as the configuration control tool with popular IDE.

4. Developer Sandbox

The developer "sandbox" is where each developer keeps his or her working copy of the code base. In CVS this is referred to as the working directory. This is where they build, test and debug the modules that they are working on. A sandbox can also be the area where the staging build or the production build is done. Changes made in the work area are checked into the CVS repository. In addition, changes made in the repository by others have to be updated in the sandbox on a regular basis.

The best practices related to developers sandbox are:

4.1. Keep System clocks in Sync

CVS tracks change to source files by using the timestamp on the file. If each client system date and time is not in sync, there is a definite possibility of CVS getting confused. Thus system clocks must be kept in sync by use of a central time server or similar mechanism.

CVS is designed from ground up to handle multiple timezones. As long as the host operating system has been setup and configured correctly, CVS will be able to track changes correctly.

4.2. Do not share the sandbox

Sandboxes have to be unique for each developer or purpose. They should not be used for multiple things at the same time. A sandbox can be a working area for a developer or the build area for the final release. If such sandboxes are shared, then the owner of the sandbox will not be aware of the changes made to the files resulting in confusion.

In CVS, the sandbox is created automatically when a working copy is checked out for a CVS project using the **cvs checkout {project−name}** command.

In very large projects, it does not make sense for the developers to check−out the entire source into the local sandbox. In such cases, they can take the binaries generated by the build team on a regular basis for all those components of the application that is not changed by them and only check−out the parts that are built by the developer.

For example, in a Java project, the build team can keep the results of their last successful build in a standard location in the form of JAR files on the network file servers. Individual developers will use a standard classpath setup that has the network drives mounted on standard paths. Thus, the developers will automatically get the latest version of the files as required by them.

4.3. Stay in sync with the repository

To gain the benefits of working within a sandbox as mentioned above, the developer must keep his or her sandbox in sync with the main repository. A regular **cvs update** with the appropriate tag or branch name will ensure that the sandboxes are kept up to date.

4.4. Do not work outside the sandbox

The sandbox can be thought of as a controlled area within which CVS can track for changes made to the various source files. Files belonging to other developers will be automatically updated by CVS in the developer's sandbox. Thus the developer who lives within the sandbox will stand to gain a lot of benefits of concurrent development.

4.5. Cleanup after Completion

Make sure that the sandbox is cleaned up after completion of work on the files. Cleanup can be done in CVS by using the **cvs release** command. This ensures that no old version of the files exists in the development sandbox. As explained previously, pre–built binaries from the build team can be used to ensure that all the parts of the application are available to the developer without the need for a complete compilation in the sandbox.

4.6. Check–in Often

To help other developers keep their code in sync with your code, you must check–in your code often into the CVS repository. The best practice would be to check–in soon as a piece of code is completed, reviewed and tested, check–in the changes with **cvs commit** to ensure that your changes are committed to the CVS repository.

CVS promotes concurrent development. Concurrent development is possible only if all the other developers are aware of the ongoing changes on a regular basis. This awareness can be termed as "situation awareness"

One of the "bad" practices that commonly occur is the sharing of files between developers by email. This works against most of the best practices mentioned above. To share updates between two developers, CVS must be used as the communication medium. This will ensure that CVS is "aware" of the changes and can track them. Thus, audit trail can be established if necessary.

5. CVS Server Configuration

This section deals with best practices for CVS server side setup and configuration.

5.1. CVS access control

One of the important questions that I have been asked time and again is the ability to have access control for files/folders/branches etc., within the CVS repository for various users. Unfortunately CVS does not come with a built in Access control capability but it does support a rudimentary form of access control through the readers/writers files in the CVSROOT repository. I have put together a set of scripts that use the readers/writers files to provide a slightly useable version of access control. This is available at http://cvspermissions.sarovar.org as an Open Source project. Feel free to use it and let me know how it works for you.

5.2. Server side scripting

Server side scripting refers to the ability to make CVS server execute certain scripts when an event occurs. A common script that helps is to verify that all cvs commits contain acomment entered by the developer. The process involves setting up the `CVSROOT/verifymsg` file to run a script when a file is checked−in.

```
------CVSROOT/verifymsg---------

#Set the verifymsg file to fire a script
DEFAULT /usr/local/bin/validate-cvs-log.sh


------/usr/local/bin/validate-cvs-log.sh ---------

#!/bin/sh
#
# validate-cvs-log.sh logfile

# test that log message has some characters in it
if [ `cat $1 | wc -c ` -lt 10 ] ; then
echo "log message too short; please enter a description for the changes"
    exit 1
else
    exit 0
fi
```

5.3. Server Notification

The CVS server can be configured to notify through e−mails in case of a commit happening. This can be used to verify whether commits are occurring during the course of a daily/release build. If such commits occur, based on the project policy, the commits can be ignored or the entire build automatically restarted.

6. Branching and Merging

Branching in CVS splits a project's development into separate, parallel histories. Changes made on one branch do not affect the other branches. Branching can be used extensively to maintain multiple versions of a product for providing support and new features.

Merging converges the branches back to the main trunk. In a merge, CVS calculates the changes made on the branch between the point where it diverged from the trunk and the branch's tip (its most recent state), then applies those differences to the project at the tip of the trunk.

6.1. Assign ownership to Trunk and Branches

The main trunk of the source tree and the various branches should have a owner assigned who will be responsible for.

1. Keep the list of configurable items for the branch or trunk.

   The owner will be the maintainer of the contents list for the branch or trunk. This list should contain the item name and a brief description about the item. This list is essential since new artifacts are always added to or removed from the repository on an ongoing basis. This list will be able to track the new additions/deletions to the repository for the respective branch.
2. Establish a working policy for the branch or trunk.

   The owner will establish policies for check–in and check–out. The policy will define when the code can be checked in (after coding or after review etc.,). Who is responsible to merge changes on the same file and resolve conflicts (the author or the person who recently changed the file).
3. Identify and document policy deviations

   Policies once established tend to have exceptions. The owner will be responsible for identifying the workaround and tracking/documenting the same for future use.
4. Responsible for merge with the trunk

   The branch owner will be responsible for ensuring that the changes in the branch can be successfully merged with the main trunk at a reasonable point in time.

6.2. Tag each release

As part of the release process, the entire code base must be tagged with an identifier that can help in uniquely identifying the release. A tag gives a label to the collection of revisions represented by one developer's working copy (usually, that working copy is completely up to date so the tag name is attached to the "latest and greatest" revisions in the repository).

The identifier for the tag should provide enough information to identify the release at any point in time in the future. One suggested tag identifier is of the form.

`release_`{major version #}_{minor version #}

As one reader pointed out to me, a good practice here is to tag the release first. Checkout the entire codebase using the tag, and then proceed to go through a build / deploy / test process before making the actual release. This will absolutely ensure that what "leaves the door " is a verified and tested codebase.

6.3. Create a branch after each release

After each software release, once the CVS repository is tagged, a branch has to be immediately created. This branch will serve as the bug fix baseline for that release. This branch is created only if the release is not a bug fix or patch release in the first place. Patches that have to be made for this release at any point in time in the future will be developed on this branch. The main trunk will be used for ongoing product development.

With this arrangement, the changes in the code for the ongoing development will be on the main trunk and the branch will provide a separate partition for hot fixes and bug fix releases.

The identifier for the branch name can be of the form.

`release_`{major version #}`_`{minor version #}`_patches`

6.4. Make bug fixes to branches only

This practice extends from the previous practice of creating a separate branch after a major release. The branch will serve as the code base for all bug fixes and patch release that have to be made. Thus, there is a separate repository "sandbox" where the hot fixes and patches can be developed apart from the mainstream development.

This practice also ensures that bug fixes done to previous releases do not mysteriously affect the mainstream version. In addition, new features added to the mainstream version do not creep into the patch release accidentally.

6.5. Make patch releases from branches only

Since all the bug fixes for a given release are done on its corresponding branch, the patch releases are made from the branch. This ensures that there is no confusion on the feature set that is released as part of the patch release.

After the patch release is made, the branch has to be tagged using the release tagging practice (see Tag each release).

7. Change Propagation

Change propagation practices explore how changes made to one version of the application are migrated to other living versions of the application.

7.1. Merge branch with the trunk after release

After each release from a branch, the changes made to the branch should be merged with the trunk. This ensures that all the bug fixes made to the patch release are properly incorporated into future releases of the application.

This merge could potentially be time consuming depending on the amount of changes made to the trunk and the branch being merged. In fact, it will probably result in a lot of conflicts in CVS resulting in manual merges. After the merge, the trunk code base must be tested to verify that the application is in proper working order. This must be kept in mind while preparing the project schedule.

In the case of changes occurring on branches for a long period, these changes can be merged to the main branch on a regular basis even before the release is made. The frequency of merge is done based on certain logical points in the branch's evolution. To ensure that duplicate merging does not occur, the following practice can be adopted.

In addition to the branch tag, a tag called {branch_name}_MERGED should be created. This is initially at the same level as the last release tag for the branch. This tag is then "moved" after each intermediate merge by using the **–F** option. This eliminates duplicate merging issues during intermediate merges.

8. Software Builds

This section deals with the best practices for software builds. Build is the process of creating the application binaries for a software release. They are done in a periodic manner by the build teams to provide baseline binaries for daily work.

8.1. Build Early and Build Often (BEBO)

A variation of this adage has been around in the Open Source community called "Release Early and Release Often" for quite some time albeit for a different reason. BEBO helps a development team identify issues that can arise from checking in the wrong files. BEBO will address integration issues at the application level that might have slipped passed individual developer builds. It will also improve the team morale when they see a working version of the application.

Builds must be done on a regular basis. There should be a dedicated resource(s) assigned to do the same. The entire project team must be trained to view the daily build as an important activity and not as a chore. Builds must be completed without any failures on a regular basis. Build failures must be a rare event and should be treated with utmost seriousness. The project team should ensure that successful builds are top priority on their agenda. The seriousness can be emphasised by setting up a penalty for breaking the build.

Each build can be tagged in CVS using a standard naming convention. This can help developers checkout a working version of the entire system from daily builds for local development.

8.2. Automate build Process completely

Another key practice for software builds is to automate the build process completely. The automation process must also include automatic retrieval of the right source files from the CVS repository. This ensures that the build process is completely repeatable and consistent. In addition, the chances of a build with the wrong version of the application source files are reduced to a large degree.

By automating the build process, the task of building often becomes less burdensome.

8.3. All necessary files must be checked−in before build

This adage sounds trivial at first but this problem is very common even with experienced development teams due to oversight. The problem of oversight cannot be easily addressed since the onus is on the individual developer to ensure that his or her file has been checked in. This practice should be drummed into the team in the form of training and pre−build announcements to ensure that the right version of source code is available in the repository.

Automated build process as explained above will help in catching this problem to a certain degree since they will automatically take the source code from the CVS repository and perform the software build. Any missed items will surface during the build process itself (makefiles etc.,) or during the regression testing of the product (older version of the file checked in).

A penalty based system can be setup to handle wrong check−in. Having a kitty for a post project party to which each person who makes a wrong check−in will contribute a fixed amount will act a good penalty system.

9. Institutionalize CVS in the Organization

Here we will look at the best practices for institutionalizing CVS usage in the organization.

9.1. Implement Change Management Process

All organizations must implement a good Change management process (CMP). A good CMP will define how changes are received, recorded, tracked, executed and delivered. CVS provides version control for your project. Change management addresses the "bigger picture" of how enhancements and bugs are received, tracked and closed. CVS will play a smaller but a very important part in this entire picture. With a formal change management process in place in the organization, tools such as CVS will be looked at as aiding this process instead of acting as a general development overhead.

Change management is quite a vast topic that cannot be done justice here. Please look up other sources of information on change management.

9.2. Make CVS Usage part of Objectives

To institutionalize CVS, it can be made as part of the performance objectives for the developer to use CVS in the project. In addition, it can also be made as part of the objective for the project manager to deploy CVS in his or her project.

Compliance of this can then be reviewed as part of the appraisal cycle for the employee.

9.3. Collect metrics on CVS usage

CVS usage metrics can be collected in terms of percentage of deployment in the organization, project size handled etc., This information will spur other line managers and program managers to look at CVS as a tool that will aid them in their daily operations.

10. Best Practices in Action

The best way to explain the need for these best practices is by putting together an example of a real world project scenario and show how exactly will these best practices fit into the "bigger picture". Also, a lot of readers have told me that the sections on Branching and Merging and Change Propagation will require examples for better explanation. Listening to the readers is a Good Thing so I have put together a particular project scenario and then create a series of events to show how the best practices, if followed, would help is making operations smoother.

10.1. Inception

Consider a software project where version 1.0 has just been put into production and everyone is done celebrating. The next step is to start working on the new features of the subsequent release. Also, the users of the system have started to use it full–time and bug reports of various levels have started to come in.

Before jumping into new enhancements or bug fixes, the best practices for Branching and Merging should be followed. Few of the important practices are Tag each release and Create a branch after each release. These practices will effectively established two "development environments", one for regular enhancements and the other for bug fixes and minor enhancements on the last release.

Let us assume that the release was tagged as

```
release_1_0
```

Then the branch was created with the branch name

```
release_1_0_patches
```

10.2. Development and Delivery

Now, we are ready for business. Let us examine the bug fixes and enhancements track. Assume that there are three bugs of which two are of a high priority that should be fixed right away (possibly within a week) and the third can be delivered after some time (say after 4 weeks). In the middle of this schedule there is a regular release scheduled in three weeks. Considering that we have a busy month ahead, let us see how exactly we can use the Best practices to ease the days ahead.

The timeline for the various release in the next month looks like this.

```
             Fix       Enhancement  Fix
     Today   Release 1  Release    Release 2
        |_______|______________|_________|
                  Time -->
```

We have two teams, one working on the bug fix branch and another team working on the features for the next release on the main trunk. These teams must make sure that they start out with the right version in their sandbox.

1. The bug fix team will check out using the command line

cvs checkout −R −r release_1_0_patches {project name}

2. The team that is working on the next release will use the command line

   cvs checkout −R {project name}

As soon as the bug fix team completes the two top priority bugs, they will update, verify a successful build and commit their changes to the bug fix branch using the command line

cvs update −R −r release_1_0_patches {module name}

The team should perform a build at this point to verify that the update did not break any code on the branch. Once the build is successful, the branch should be committed back into the repository.

cvs commit −R −r release_1_0_patches {module name}

Build Early and Build Often : On a daily basis, each developer will check in code to CVS and to ensure sanity of code, daily builds on the bug fixed branch will be undertaken by checking out from CVS on a clean environment and completely rebuilt. These daily builds can be tagged in CVS using the following naming convention

```
build_1_1_yyyymmdd : for the branch
build_2_0_yyyymmdd : for the trunk
```

The regular process of build−test−fix is followed to make a version ready for delivery. The tag will help developers checkout a working copy of the latest build as and when necessary.

When the source code is released to the outside world, two practices have to be followed.

1. Tag each release : This ensures that the bug fix release is tagged correctly and so can be traced out at a later point in time if necessary.
2. Merge branch with the trunk after release : This ensures that the bug fix is merged back into the main trunk ensuring that all future releases is a truly cumulative delivery.

11. Conclusion

These best practices are meant to help software teams get a head start on using CVS for their development. The ideas presented here have to be constantly reviewed and evolved. I would like this to be a growing and evolving document. Please send your comments and ideas to `<vivekv at yahoo dot com>`

A. GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front−matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front−Cover Texts or Back−Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine−readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard−conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine−generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front−Cover Texts on the front cover, and Back−Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine−readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly−accessible computer−network location containing a complete Transparent copy of the Document, free of added material, which the general network−using public has access to download anonymously at no charge using public−standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front−matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties−−for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front−Cover Text, and a passage of up to 25 words as a Back−Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front−Cover Text and one of Back−Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self–contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. Future Revisions of this License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front−Cover Texts being LIST, and with the Back−Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front−Cover Texts, write "no Front−Cover Texts" instead of "Front−Cover Texts being LIST"; likewise for Back−Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.