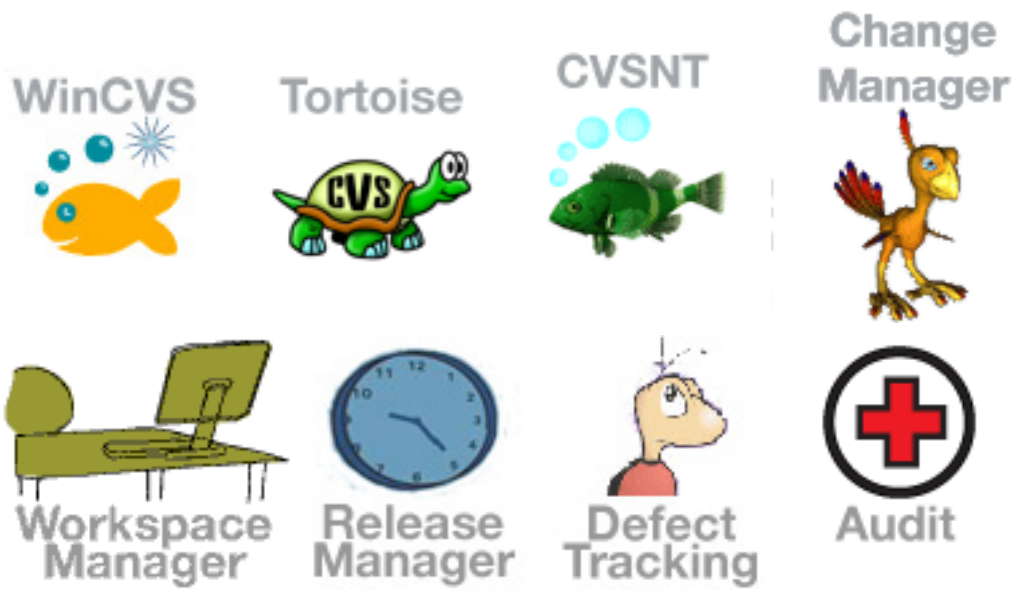
Red Hat Enterprise Linux and CVS Suite

INSTALL GUIDE **2009** Build 9329 **July 2025**



Optimized for Microsoft Visual Studio

March Hare Software Llc

Legal Notices

There are various product or company names used herein that are the trademarks, service marks, or trade names of their respective owners, and March Hare Software Limited makes no claim of ownership to, nor intends to imply an endorsement of, such products or companies by their usage.

This document and all information contained herein are the property of March Hare Software Limited, and may not be reproduced, disclosed, revealed, or used in any way without prior written consent of March Hare Software Limited.

This document and the information contained herein are subject to confidentiality agreement, violation of which will subject the violator to all remedies and penalties provided by the law.

LIMITED WARRANTY.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, March Hare Software Limited AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THIS DOCUMENT, AND ANY ADVICE OR RECOMMENDATION CONTAINED IN THIS DOCUMENT.

NO OTHER WARRANTIES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL March Hare Software Limited OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE FOLLOWING DOCUMENTATION INCLUDING ANY RECOMMENDATION OR ADVICE THERIN, EVEN IF March Hare Software Limtied HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, March Hare Software Limited's ENTIRE LIABILITY UNDER ANY PROVISION OF THIS DOCUMENT INCLUDING ANY RECOMMENDATION OR ADVICE THERIN SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE DOCUMENT OR £5.00; PROVIDED.

march-hare.com

sales@march-hare.com

Acknowledgements

March Hare Software Limited thank the many contributors to the Open Source CVS, CVSNT, CVSWEB, CVSWEBNT, WINCVS, TORTOISECVS and BUGZILLA projects for their tremendous effort and support.

In particular Tony Hoyle has been instrumental in the development of CVSNT, its reference manual and this training manual.

Articles by Bo Berglund, Terris Linebach and Byron Jones have also been incorporated into this document.

Special thanks to Jerzy Kaczorowski for his work with WinCVS, and Torsten Martinsen for his work with TortoiseCVS.

Who Should Read This Book

Software developers, web developers and anyone who has ever wanted to find yesterdays (or last weeks) copy of a file should read this book.

If you have a legislative or contractual requirement to track by whom or when changes are made to computer based files then this course will give you a detailed understanding of various approaches to this goal.

Anyone who wants to implement / install CVS should read this book.

About This Edition

This edition was originally released with INSTALL GUIDE 2009 Build 9329 in July 2025.

The following technical issues have been identified in this release:

- *Refer to separate document Windows installation guide which explains the new installer and configuration wizard*
- *Limited information is provided on CM Suite 2008*
- *Limited information is provided on migrating from CVS Suite to CM Suite*
- *To configure multi-site repositories contact your technical account manager for a quote for on site installation – detailed instructions are NOT provided in this guide*
- *Solaris version of login command echos the password to the terminal*
- *CVS Suite for HPUX 11.23 Itanium is an experimental build*
- *Limited information is provided on upgrading Audit schemas*
- *Limited information is provided regarding the integration with Jira and Mantis*
- *Installing database driver(s) on Linux does not install Audit SQL create table scripts*

This edition contains items from the CVSNT Reference (build 2680) and march-hare.com FAQ (as at May 27th, 2006).

Quick Install Guide

Getting the most out of this software requires that you are familiar with the theory as well as the practical. We the vendors strongly recommend that you first read the *Fundamentals* section of the eBook *All About CVS* or the *Install Guide* before proceeding.

Also see the *Quick Configuration Guide* and the *Quick User Guide* below.

If you are upgrading an existing CVS or CVSNT server to CVS Suite 2009 read *Upgrading CVS Suite Server on Linux*.

Determine and test how the server will authenticate users, eg: to a windows domain with usernames and passwords, with Kerberos tickets or via ssh keys. Test using SSH.

Ensure Red Hat Enterprise software is updated using yum.

Run the CVS Suite 2009 installer as a root.

Regster a new repository2009 read *Register a repository on Linux*.

Support e-mail:
cvsasuite.support@march-hare.com

Quick Install Guide Red Hat 9 using RPM

Please also see above section *Quick Install Guide*.

Install prerequisites using yum:

```
yum install chkconfig
yum install initscripts
yum install glibc.i686
yum install libstdc++.i686
yum install libxcrypt.i686
yum install zlib.i686
```

Extract the CVS Suite RPMs

```
tar xzf cvs-suite-2009-9329-rhel9-rpm.tar.gz
```

Install the CVS Suite RPMs

```
rpm -i cvsnt-2.8.01.9329-1.i686.rpm \
cvs-suite-triggers-2.8.01.9329-1.i686.rpm
```

Configure the server, audit plugin, bugzilla plugin etc. from the *.example files

```
/etc/cvsnt/PServer (copy from PServer.example)
/etc/cvsnt/Plugins (copy from Plugins.example)
/etc/cvsnt/server (copy from server.example)
```

Configure the server to start and auto start on reboot:

```
chkconfig --add cvslockd
chkconfig --add cvsmanager
/etc/init.d/cvslockd start
/etc/init.d/cvsmanager start
```

Test a client can connect and checkout

Support e-mail:
cvsasuite.support@march-hare.com

Quick Configuration Guide

Please also see above section *Quick Install Guide*.

Install using the RPM installer.

Create a repository

```
cvs -d :local:/usr/repo/cvsrepo -a /myrepo
```

Configure the server, audit plugin, bugzilla plugin etc. from the example files

```
/etc/cvsnt/PServer
/etc/cvsnt/Plugins
/etc/cvsnt/Bugs
/etc/cvsnt/Make
/etc/cvsnt/server
```

Import files into the repository, or copy old repository archives into new registered repository

Test a client can connect and checkout

Support e-mail:
cvsasuite.support@march-hare.com

Table of Contents

Part I – Theory

Fundamentals

What is CVSNT and CM Server all about

This software helps computer users keep track of changes to files.

All of the things you create on your computer: Documents, Program Source Code, Web Pages, Pictures, Spreadsheets can be managed using CVS Suite or CM Server.

In addition to tracking the changes, the software also can provide assistance with publishing, reviewing, securing and managing those files and the ability for different computer users to make changes at different stages of the documents life.

CVS Suite was originally designed for tracking changes to files and documents written by computer programmers: computer source code. This is still the primary use of CVS Suite and the focus of this book, though the same procedures can be used for managing any type of file.

CM Server is a more advanced edition of the software and addresses the requirements of larger teams and organisations.

Can I Install the Software and Read this Later

Effectively managing your computer files and the changes to them largely depends on how you work and what your priorities for management are.

If you attempt to use the software without understanding the theory first then you will almost certainly find it is not optimal for your purposes.

Therefore you are encouraged to read the theory before attempting the practical.

I Don't Like Version Control

In this book we use the term *Effective Configuration Management* over and over again. As an organisation it took us at March Hare Software LLC a long time to discover that there is a difference between Configuration Management and Effective Configuration Management.

Most people who do not like version control feel that way because they have been exposed to it in an ineffective environment. Spending time doing things that are ineffective will lead to an enormous level of frustration.

If you don't like version control, please read the theory section in this book and look for a process that you would be happy to use to effectively manage your work.

I prefer to use some other tool

CM Suite is the ideal server software for people who prefer to use other tools as it is client agnostic. Client agnostic CM Server is ideal for a heterogeneous CM environment, allowing each person to choose the tool most effective for them whilst not limiting the choices of other people and retaining the ability to comply with audit and management objectives. CM Suite supports most popular Version Control and SCCM client tools.

The benefits of Software Change and Configuration Management usually comes from the process, so we recommend that customers focus on identifyng the process that directly contributes to resolving the business objectives first, then secondly look for what tools can support that process.

This ensures that you implement a system that provides the benefits all stakeholders are expecting and also ensures that the benefits can be measured to determine the overall cost benefit during post implementation review.

Right versus Wrong

There is no universally right way to implement version control and/or configuration management.

The purpose of this book is not to convince you that one methodology is the correct one to the exclusion of all others.

Each methodology has strengths and weaknesses. Each decision you make while implementing your solution has consequences – most notably in the procedures and management structures that interface with this system

It is very likely a single methodology is correct for you. The purpose of this book is to help you find that and assist you with implementing either CVS Suite or CM Suite to work that way. If you believe some teams may require a different methodology then they should work through this book and make their own choices based on the provided information and questions.

Some Place to Start

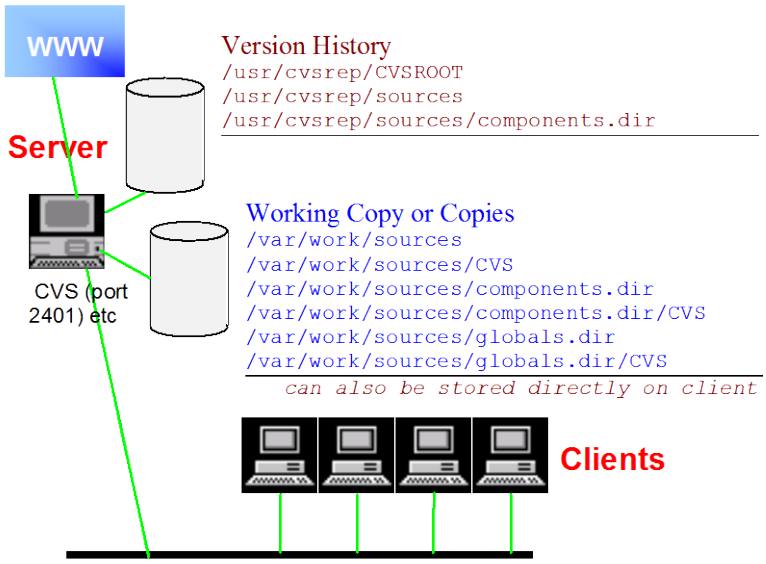
Since there is no correct way to implement version control or Software Change and Configuration Management some people reach an implementation paralysis due to the number of choices. To assist in resolving this we have provided some basic samples of how to use the Suite software tools in the client section of this book.

CVS Architecture

CVS Suite Server and Client are designed to work together and in conjunction with Configuration Management processes can improve developer productivity. Setting up CVS Suite Server and Clients is NOT as simple as installing the software. This section *CVS Architecture* and the next sections *Setting up CVS Suite Server* and *Server Administration* provide the necessary information to install and configure the CVS Suite system. If you do not follow these guides then severe performance problems can occur.

Client / Server Architecture

When using CVS to version control your documents and other objects you always will use the client program (with or without a GUI) with files in a *sandbox*. The CVS Server will always operate on the repository files.



The CVS Server and the CVS Client both use the same executable program *cvsnt.exe*. The server also requires additional components to configure it and start the servers when clients connect. See the section *When does the server start* for more information. The *cvs.exe* is a proxy that will silently start *cvsnt.exe* if required or proxy with *ssh* on a unix server.

There are various CVS *Front Ends* otherwise known as *GUIs*. These all call the CVS executable to perform their functions and to communicate with the server.

The CVS Server may also optionally run a web service (IIS on windows) with CVSWEB or ViewCVS to allow Clients to browse the CVS history using a web browser.

Clients connect to the server via one of several available authentication protocols. March Hare Software recommends that organizations choose their authentication protocols based in part on the security recommendations (see *Security Architecture* section below).

Communication between the clients and server can be encrypted using ssh, gserver or sserver protocols (and sspi on windows only), and some protocols such as sspi and pserver support compression. Compression is performed before encryption so both may be enabled at the same time.

The point at which the encryption starts depends on the client and server software in use. On a CVS client/server (including 3rd party CVS clients like Eclipse) the encryption starts as part of the initial setup, after the root request. CVSNT client and server supports 'rootless encryption' which means that it encrypts immediately after the initial handshake. This is required for working in a secure environment where the root is to be concealed from any network snooping.

When does the server start

CVSNT server does not have a monolithic architecture. What that means is that all clients do not connect to a single process running on the server.

The CVSNT server consists of three processes:

- CVSNT Deadlock Server (EVS combines this with EVSMANAGER)
- a Server Service, either:
  - ⇒ CVSMANAGER High Performance Server Service, or
  - ⇒ EVSMANAGER Server Service (EVS only)
- a Server, either:
  - ⇒ CVSNT Server, or
  - ⇒ EVS Server

The deadlock server runs constantly and serves all locking requests (read locks and write locks), and also publishes server information using Rendevous / Bonjour. The EVSMANAGER service provides similar functions for CM Server.

The CVSMANAGER High Performance Server Service starts and maintains a pool of several CVSNT Server processes. Each waiting process has already performed all initialisation including trigger initialisation and is ready to authenticate and serve a request from a client. If the client is using ssh then the CVSPROXY (*cvs.exe* or */usr/bin/cvs*) will connect to a waiting CVSNT server.

The EVS Server or older 2.5.x CVSNT Servers are started only when a CVS or CVSNT client performs an operation that requires the server to respond. The client connects to the server using port 2401 (or port 22 in the case of SSH) and the operating system (or the EVSMANAGER or CVSNT Server Service in the case or Windows) starts a unique CVSNT Server process for that client.

Web folder or web browser clients and SVN clients do not create additional server processes (the EVSMANAGER service handles all these requests).

Advantages of a Non-Monolithic model

Since each client begins a new process on the server, it is easy to establish information about how many users are currently accessing the server, and if necessary terminate or monitor them.

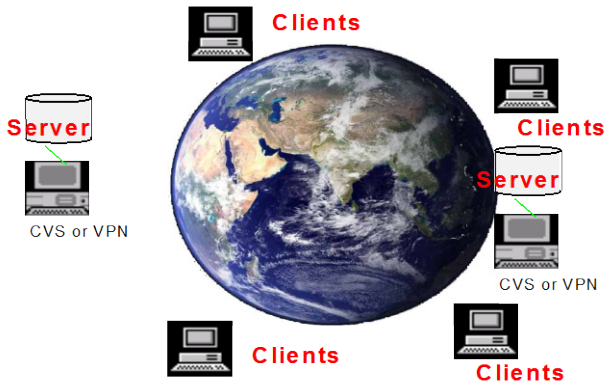
Advantages of a Monolithic model

The primary advantage of using a monolithic model is when the host operating system has a large overhead when creating a new process. CVSNT high performance server overcomes this problem without the added complexity of a monolithic (multithreaded) server.

If your environment will have many hundreds of developers performing *edit* commands on single files at the same time then the overhead of process creation could become high, and the CVSNT Server should be upgraded to 2.8.01 or higher and a suitable pool size set in the server control panel.

Multi Site, Repository Replication and WAN Performance

CVSNT is designed to work well when users are distributed worldwide:



The purpose of this section is to explain in theory how to use CVS well in a distributed environment, and what should be avoided.

Overview

Before starting a technical discussion about Multi Site version control with CVSNT there are some introductory messages that we must clearly advise you of:

- If you have multiple sites relying on CVSNT then you should be be using the latest version of the software with Silver level support or higher, and definitely not using a 10 year old community edition
- Only about 4% of CVS Suite is non-GPL code and the repository replication solutions we offer are all based on open source technologies – no large monolithic proprietary solutions are required
- Define the problems you are trying to solve with metrics

Support of CVSNT Server and your Technical Account Manager

Ensuring the reliability, resilience and integrity of your repository are important when implementing a SCM versioning server. If you are unhappy with the performance or reliability of an existing CVS or CVSNT server then the first step is to ensure that you have support from the people who wrote it, and speak to your Technical Account Manager regarding the specific problems you are experiencing – March Hare Software write CVSNT and can assist with resolving problems through CVS Professional Support Services.

Software Solutions

There are unscrupulous individuals and even some organisations that attempt to convince CVS and CVSNT users that they are experts on Multi Site Version Control and based on that expertise they recommend purchase of a proprietary software solution.

No author of CVS or CVSNT has ever written or said that additional proprietary software is required to resolve any multi-site, performance or reliability issue with the CVS or CVSNT server.

Some of the proprietary software that is advertised for enabling Multi Site CVS or CVSNT also claims to provide added security benefits, reports and controls. CVSNT contains sophisticated and reliable security controls that are better integrated and more reliable than any proprietary add-on package, including:

- Access Control Lists
- Authentication
- Deny/Allow clients

No proprietary software is required to resolve multi-site, performance security or reliability issues.

Define the Problem

It is our experience that people ask about Multi Site CVS for the following reasons:

- They are not currently using CVSNT and they expect problems due to previous experience with other SCM tools like ClearCase, Visual SourceSafe or Continuus/CM Synergy.
- They are not currently using CVSNT and they have been approached by a vendor of a proprietary Multi-Site product or have read a review of such a product.
- They are currently using CVS 1.x not CVSNT, CVS Suite or CM Suite.
- They are using CVSNT and do not have support and have not received professional advice by the authors on how to configure and use the system, eg: setting up 'stand by' servers. The eBook *All About CVS* (a part of CVS Professional or CVS Suite) describes how to set up a standby server and also a suite of all the necessary software that has been tested to work together as well as e-mail based support).
- They are using CVSNT and have a specific performance problem (eg: a 'build' process takes too long or a 'tag' process takes too long).

If the last point applies to you then you should document the specific problem you are experiencing and contact your March Hare Software Technical Account Manager.

For all other problems described above you should purchase CVS Suite or CM Suite with Software Maintenance and Professional Support from March Hare Software, and optionally the on-site training and installation services. See the section *Learning more about CVSNT* above and specifically the headings *Professional Support* and *Contacing Us*.

As with any IT project – you should define the expected outcomes before you begin the project so that after the project is completed then you can establish the relative success of it. If you require the tag to complete within 10 minutes then write that down, if you require the build to complete in 1 hour then write that down – and then give that information to your Technical Account Manager or pre-sales technical support. Ensure that all metrics are not arbitrary, but are based on a business requirement that can be explained easily to our team.

If there is no proprietary software needed why do I need to pay for support?

By encouraging those who require more assistance to pay for support we can ensure that the software continues to be developed and provide the best levels of service to those users who rely on CVSNT for their productivity at work.

Why do March Hare require Silver or higher support for Multi Site?

March Hare Software base our charges on the effort required to resolve problems. It is our experience that if a company has users at multiple sites that their problems are more difficult to resolve. Additional (open source) software may also be provided for customers with multiple sites designed to resolve particular technical problems that only occur in multi-site installations.

Silver level support is also a pre-requisite for on site services – we strongly recommend that one of our team come to your site(s) when installing and configuring Multi Site solutions. Customers with silver support also have telephone access to our team which is very likely to be needed in the first year after installation of a multi-site product.

Multi Site authentication

When working with multiple sites it is important that the authentication scheme is carefully designed. If users are being authenticated by a remote authentication server (eg: a domain controller) then CVSNT client operations may be delayed while that authentication takes place for each command. These types of authentication issues are often not noticeable with operations such as logging in to a PC or connecting network drives, however since CVSNT client operations can be performed many times a minute (especially if using SCCI/Visual Studio Integration) the lag for authentication becomes critical. Also see the sections *Unix and Linux Server Synchronisation to a Reliable Time Source* and *Windows Server Synchronisation to a Reliable Time Source* for other reasons why client authentication may be slow in a multi site installation.

A second multi site issue is that users who are not registered on the central authentication system (such as a domain controller) may require access to your CVSNT server. See the section *CVS only users (no system user with that username)* for details of how to overcome this problem.

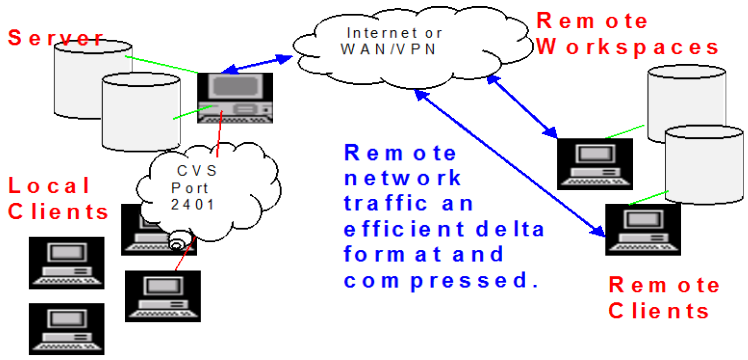
If performance should be good without multi-site software, why is my performance poor?

The most common reasons why we see poor performance are:

- The existing tools are not being used in conjunction with CM and CVSNT best practice.
- There are existing procedures that are not required and are inefficient. They may have been introduced to resolve a problem that there were insufficient skills to resolve any other way or a problem that has been resolved in later releases of the software.
- The expectations are unrealistic (defined as performance expectations that could not be matched with a local workgroup repository). Eg: if people using the software are trying to highlight the advantages of an alternative version control system where the repository is not stored or managed centrally and stored only on their own computer.
- The performance has not been measured and the performance problem overstated. This is occasionally seen with 'outsource' suppliers who are looking for a scapegoat to blame for missed deadlines.
- The performance problems are a direct result of insufficient training of the people who are using the software (the people experiencing the performance problem). This is not common, however can be occasionally seen with 'outsource' suppliers who are using insufficiently trained staff and do not want to bear the cost of training.

WAN Optimised Workspaces and Procedures

Most people who need to use CVSNT at multiple sites or geographically distributed locations will use this solution. This solution provides high performance for remote sites as well as insulation from network connection failures.



The local workspaces provide insulation from network outages. People working remotely continue uninterrupted when there is no connection to the server.

When connections to the server are available the Remote workspaces can be re-synchronised to the server using efficient compressed changesets.

When implementing WAN Optimised Workspaces it is important that the software and the procedures are correctly configured to reduce the network transmission overheads.

For example: it is typical to ensure that binary objects (eg: compiled objects such as `.jar`, `.obj`, `.exe` files) are not regularly checked out and updated using CVSNT, but instead that a 'build server' is located at each site that can create these for the users at the remote site. Implementing a more complex strategy for binary compiled objects is counter-productive since they still need to be regularly synchronised between the servers and results in more administration of conflicts.

Another example is to ensure that developers do not delete workspaces. If a developer was working on Trunk but now needs to work on branch-rel-1 then the trunk workspace should be kept and the developer switches to working on the branch-rel-1 workspace.

At the remote site it is typical to maintain several sandboxes (at least one per branch). These can be set to automatically synchronise to the server whenever the server is available, and most importantly set to synchornise before the work day for the remote site commences. This sandbox can be copied to another persons PC at the same remote location if they need a 'clean' copy of the latest sandbox.

Often remote developers are fearful of making changes when there is no connection to the server because they fear that their workspace may be out of date. Configuring the CVSNT server with automatic e-mail notification will advise users at all sites (via a group e-mail address) of what is changing on the server. A more complex repository replication strategy would not provide a more reliable solution since if the link is down the cache will be out of date.

Multiple Repositories

Multiple servers (repository replication) may be considered for the following reasons:

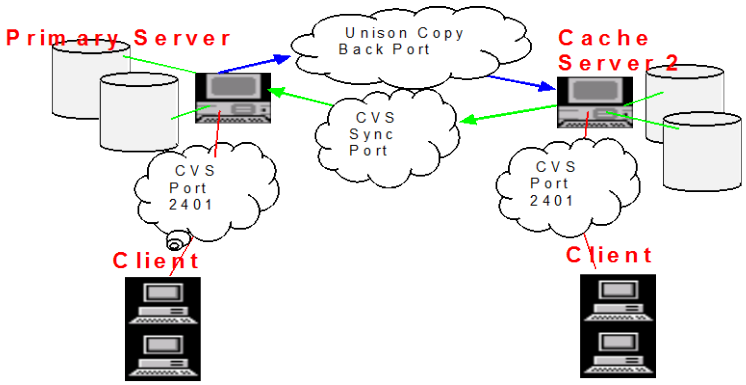
- Large Dual Site Development (Performance / Repository Replication Caching)
  Typically there are two sites in different timezones so that work can proceed 24 hours a day on a project. It is common for each site to house more than 500 users.
- Repository Mirrors for Disaster Recovery or Ownership
  The majority of work is perfomed at one location (eg: India) however the work is owned by an organisation elsewhere (eg: North America)
- Multi Site parallel Development - Offshoring
  Where distinct work is carried out at different sites, eg: In India version 2 is developed but England maintains version 1 and tests version 2.

Large Dual Site Development - Repository Replication Cache

CVSNT client server communication was designed for the kind of networks that were available over twenty years ago. The networks typically in use today exceed the performance and bandwidth of those many times over. Therefore it is not necessary to use Repository Replication to obtain good performance from your CVSNT based CM system.

Implementing replication to improve performance generally does not work: if there is a performance problem it is most likely in the underlying infrastructure or CM methodology. If you have less than 50 users at each location then the best solution for you is described in the section *WAN Optimised Workspaces and Procedures*.

In a few rare cases it may be beneficial to have a local (or regional) cache if there are hundreds of users at each site or if policy prevents the introduction of more efficient workflows and procedures.



All ***read*** requests from the client are processed from the *Cache Server 2* with no need to communicate via the link to the *Primary Server*.

Write requests, such as tag, and commit are redirected by the *Cache Server 2* to the *Primary Server*.

This technology is available to CVS Suite and CVS Suite x64 customers with Silver (or higher) level support who purchase additional on site consulting for the installation and certification of the replication software. Configuring repository replication requires:

- CVS Suite and Silver (or higher) support for *each* person using the software at *each* site
- CM Design and CVS Administration Training Course
- Installation Consulting for *each* site
- Professional Services (usually 1-3 days per site)

Prices and descriptions for each of these are available on the web store, including a sample package for two sites with up to a total of 100 users:

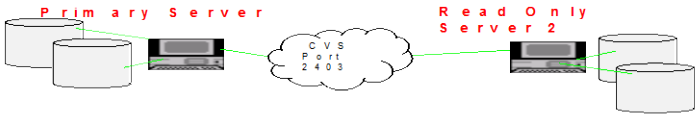
http://store.march-hare.com/s.nl/sc.2/category.12/.f

- 100 x CVS Suite (32 bit edition) with Silver Support [may vary please contact sales]
- 1 x CVS Design and Administration Training Course
- 2 x CVS Installation Consulting for *each* site
- 3 x days on site CVS Professional Services [may vary please contact sales]

Read Only Mirrors - Repository Replication Disaster Recovery or Ownership

Your business may require copies of the repository to be stored at multiple physical locations in case of a critical failure in infrastructure at the primary location, or to ensure that the ownership of the property is centralised.

Included in CVS Suite is a repository replicator named Unison designed for this purpose.

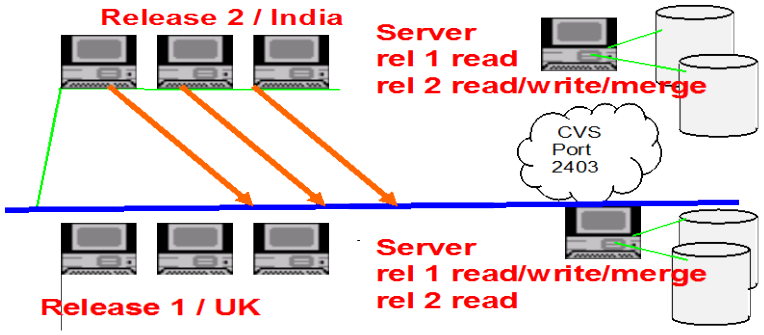


Replication may be triggered at each commit to the primary server, or at a time based interval. See the section on *Unison Repository Replication*.

If the primary server becomes unavailable, a typical disaster recovery procedure would be to use the DR DNS server to assign the *Read Only Server 2* the same name as the original *Primary Server* therefore allowing users to recommence working with the least disruption.

Multi Site Parallel Development

This is required when some of the development of a project has been outsourced to a remote site. In this example each stream of development is clearly performed at a different physical location – though users at all locations need to see the work in progress or initiate merges.



This functionality is only available in CM Suite Multi Site – please discuss your requirements with your Technical Account Manager.

Other Multi Site Techniques

Active-Active Repository Replication (or Quorum based Multi Site)

This solution is not recommended – do not use the services of any consultant advertising these services. March Hare Software regularly receives support requests from companies that have implemented this software and experience problems that the vendor cannot resolve. Our recommendation is to uninstall this software and purchase professional support from the people who wrote CVS and CVSNT.

A quorum based multi site solution uses a large number of servers (usually a minimum of 5 or 8) which all regularly synchronise with each other. Based on the number of servers that can be reached via the network at any time (the quorum) the 'authoratative' server(s) are established and changes sent to servers that are not a part of the quorum are excluded.

This solution is typically offered based on a perception that it improves performance on unreliable networks (eg: India and China), however it does not achieve the desired result. As soon as the nodes in other continents cannot be reached they are excluded from the quorum and any changes there become non-authoratative.

Technically this proprietary software is implemented as a proxy that attempts to intercept the protocol between the client and server – since the authors of the proxy are not members of the active open source CVS and CVSNT communities and the CVSNT clients and servers are written by March Hare Software and open source community members around the world it appears that the proxy intervention regularly causes simple operations to fail.

Due to the large number of servers required there are also very high procurement costs and administratative overhead to this solution.

CVS Server Clustering

This solution is not recommended – typically only organisations hosting projects for many geographically distributed companies require clustering, eg: SourceForge or gnu.org. Some enterprise customers with more than 500 users at a single location may consider this option once initial workflow and procedural issues have been resolved on a phased deployment of smaller stand alone servers.

CVSNT supports clustering 'out of the box' – the deadlock service started on only one node of the cluster co-ordinates the activities of cvs server processes on all nodes of the cluster. The repository is stored on a suitable cluster filesystem[1] on a SAN.

This cluster software is typically combined with a network load balancing switch such as the Barracuda Load Balancer[2] or Cisco Catalyst 4500/4000 Series Switch[3] or with Cluster load balancing software (built into the operating system, eg: Windows Server 2003 Advanced Edition or Red Hat Linux AS 4).

A cluster provides both redundancy and high performance.

As you can see from those references – clusters are older technology that whilst tried and tested and very reliable, are not much in fashion today. Clusters are still supported by Windows Server 2019 and 2016 and you can find about them via Microsoft[4]

SSH versus CVS Network Protocols (SSPI, Sserver, etc.)

Network protocols allow the CVSNT client and server to connect. There are many to choose from. One protocol or another is typically chosen because of the features it supports.

We encourage customers to choose the protocol which is the most secure whilst providing the easiest use (lowest friction) for the person using CVSNT – eg: SSPI on Windows or SSH on Linux. The relationship between compression and encryption, and the point that encryption starts is explained above in the section *Client / Server Architecture*.

There are two network servers (or classes of network protocols) that can be used to connect the CVSNT client and the CVSNT Server.

---

[1] Eg: SAN Filesystem from DataPlow, or
IBM Tivoli SANergy: http://www-306.ibm.com/software/tivoli/products/sanergy/

[2] http://www.barracudanetworks.com/ns/products/balancer_overview.php

[3] https://www.cisco.com/c/en/us/support/docs/lan-switching/etherchannel/12023-4.html#cat4k

[4] https://docs.microsoft.com/en-us/windows-server/failover-clustering/failover-clustering-overview

CVSNT 'class 1' network protocols (SSPI, SServer, etc):

The first and most common class is the ones that CVS includes its own network server (network protocol) which will directly start the CVSNT Server. Because this protocol is native to CVSNT and directly starts the server, the protocol can only be used for accessing CVSNT.

CVSNT 'class 2' network protocols (SSH):

The second class is an external protocol, most commonly on Unix/Linux if you already have SSH server (SSHD) running on the machine which will host the repository, you can use it as the connection between CVSNT client and CVSNT server.

Because this is a generic server, it can be used to login to the Unix/Linux command line or to start other services between windows and Unix/Linux. If you want to use SSH protocol but want to limit it to only CVSNT then you will need your Unix/Linux administrator to configure the user account in such a way as to prevent any other use. This is a Unix/Linux administration issue and will vary depending on your Unix/Linux operating system and version and your policies and is therefore not covered here.

It is not recommended to use an SSH server if your repository will be stored on a Windows server, use the inbuilt CVSNT SSPI protocol instead. See the section *SSPI Protocol* for more information.

CVSNT Client connected directly to CVSNT Server

The most straightforward connection is between a CVSNT client and the CVSNT Server using the included network protocol which operates over port 2401.

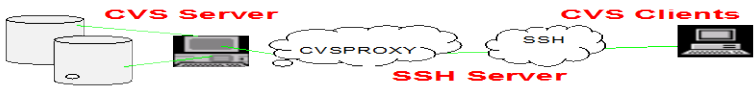
The CVSNT network protocol consists of several elements that can be chosen:

- Authentication Protocol (pserver, SSPI, sserver etc).
- Encryption
- Compression

Some authentication protocols are more secure than others, please read the *authentication* and *SSPI Protocol* sections.

CVSNT Client connected directly to SSH Server

This is the most common technique for connecting a CVSNT windows client to a CVSNT server on Linux, Unix or Mac OS X.



The CVSNT client includes a copy of the Putty SSH client. This avoids the need to set up, install and maintain a separate SSH client to CVSNT. The Eclipse CVS client also includes an SSH client called EXTSSH.

CVSNT Client connected to an external SSH client connected to SSH Server

This is the most common technique for connecting a CVSNT Unix, Linux or Mac OS X client to a CVSNT server on Linux, Unix or Mac OS X. It is no more secure than using the inbuilt ssh client in CVSNT, however Unix, Linux and Mac OS X administrators may feel this is easier to maintain. We do not recommend using this on Windows (eg: using TortoisePlink) – the included SSH client in CVSNT on Windows is newer and contains more bug fixes than the last version of TortoisePlink.

To use this technique requires:

- Use of the :ext: protocol.
- Setting of CVS_RSH environment variable with the ssh command
- Optional setting of CVS_SERVER environment variable

CVSNT Client connected to the external EXTNT client connected to SSH Server

This is a common technique for connecting an Eclipse Windows client to a CVSNT server on Windows. This allows the Eclipse client to use SSPI *via* the ext protocol of Eclipse.

To use this technique requires:

- Use of the :ext: protocol in Eclipse, TortoiseCVS and CVSNT client.
- Setting of CVS_RSH environment variable with the CVSNT client
- Setting of TortoiseCVS ext helper

Why CVSWEB Cannot Edit

CVSWEB directly accesses the CVS repository files, not a *sandbox*. The distinction between the sandbox copies of your documents and objects and the repository copies is important to keep in mind at all times.

When a person wants to change a document or an object either they or someone else must decide:

- *what* copy they will use (from what branch, trunk and repository)
- *where* their copy of the modified file will exist
- *who* else will have access to the modified file (a team via a share or only that developer by placing the file locally)

These decisions are simple to make when using a CVS client on a PC or in a Unix account – however they are almost impossible to make via a stateless session in a web browser.

Once your organization has a mature CM process, it may be relatively simple to gain access to your defined infrastructure in a web browser via ASP.NET or Perl, however a generic tool like CVSWEB cannot.

CVSROOT versus CVSROOT

There are two different meanings of the work *CVSROOT* in CVSNT.

- Repository CVSROOT.
- CVSROOT client connection string

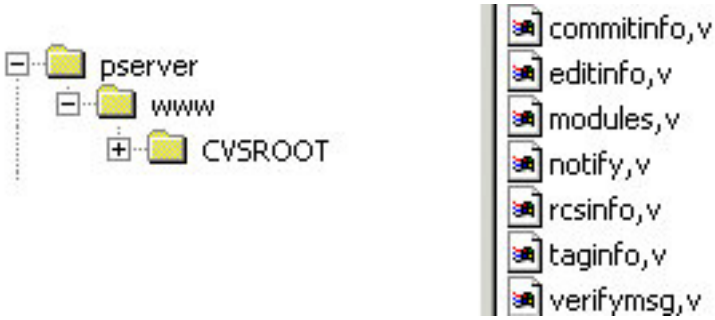
What is the Repository and the Repository CVSROOT

The CVS repository stores CVS configuration information plus all of the versions of each of your documents and objects and also meta data about each of those versions.

The repository is based entirely on RCS format files. These are very easy to understand and there are many hundreds of utilities that can read and write them. However you should only ever allow CVS to make changes to these files except instructed to do so by March Hare technical support.

The repository should not be named CVS or CVSROOT since these names have specific meaning to CVS. It is also not possible to version control a directory or file named CVS, and files and directories named AUX, LPT, CON, and other "Microsoft DOS" reserved words should be avoided.

When a new repository is initialised in consists of only one folder:



The Repository CVSROOT

The CVSROOT folder has files with a *,v* suffix and also files without a suffix. The files without a suffix are the CVS Repository configuration files and should never be edited from this directory.

See the section on *CVS Administrative files* for more information.

The CVSROOT Client Connection String

The CVSROOT client connection string is the way that a CVSNT client connects to a CVSNT server. The CVSROOT client connection string specifies:

- Connection Protocol (secure or insecure);
- Optional Protocol Options;
- Optional Username;
- Server name;
- Repository Name.

Eg: `:sspi:user@host.domain.org:/repository-name`

Directory access

The following CVSNT directories require all CVSNT users to access (unless you use RunAsUser or pserver aliases):

Example Location	*Description*
Write access	
/export/home/cvsnt/repository	Repository
/export/home/cvsnt/reference	Checkout reference area
Read and Execute access	
/usr/bin	CVSNT Program
/usr/lib/cvsnt	CVSNT Shared Libraries
Read access	
/etc/cvsnt/	Configuration files

Workspaces / Sandboxes on Network Shares or Netowrk Drives (SAN or NAS)

March Hare Software provide support for commercial support customers to use network shares (either a windows share, Samba share or NFS share), or a network drive (SAN or NAS) as the desitination for a `cvs checkout` command (this is called a workspace or a sandbox).

Under no circumstances should a compressed drive, network share or network attached storage (NAS) be used to store a repository, see the section *Creating a Repository* below for more information.

If a network share or network drive is used to store a sandbox / workspace then the following limitations apply:

- The workspace should only be used for a single architecture client, ie: do not use the same workspace from both Linux and Windows.
- The client computer and the network share or drive must use synchronised time within 500milliseconds (half a second). Note: some Samba shares and NAS devices by default have a time inaccuracy of 3 seconds which makes them unsuitable for use as a workspace for CVS.

How CVS works with Network Shares and Drives

CVS has a complex interaction with Network Shares and Drives during the *checkout* and *update* tasks.

Network Shares and Drives during Checkout

During the *checkout* task the CVS client requests a version of a file from the CVS server and writes it to a workspace / sandbox. If the workspace / sandbox is located on a network share or network drive then the resulting interaction can be complex.

The CVS Server informs the client of the date/time that the file was last committed (checked in). The CVS Client will attempt to write the file to the workspace / sandbox filesystem using that date/time (expressed in UTC).

The network share host or the processor running the network drive will interpret the CVS client request to set the file date/time based on its own rules and limitations. Some file systems such as the OS/400 may not permit the date/time to be set and will always set it to the date and time the file is written (the currect date/time), other file systems may have a low resolution time (eg: Some Samba and network drives) and will *round up* or *round down* this time to a three second interval.

The CVS Client will inspect the date / time of the file once it is written and closed and store that in the CVS client hidden data (CVS/Entries).

Network Shares and Drives during Update

During the *update* task the CVS client command compares the date / time of each file with the date and time recorded in the CVS client hidden data (CVS/Entries) and if they are different then the entire file is read and sent to the CVS Server for comparison with the original version.

If the file sent is different from the file stored in the CVS Server then the client is notified and the CVS Server sends back the differences.

The CVS Client handles a special case where the CVS server reports no differences. In this case the CVS Client updates the CVS client hidden data with the date on the file in the sandbox / workspace filesystem.

Therefore if the dates on the files in the sandbox / workspace filesystem become out of sync with the hidden data stored by the CVS Client a single *update* command will cause them to be resynchronised.

Designing a solution

Server Architecture

CVSNT Server runs equally well on Unix, Linux, Mac OS X and Windows. Deciding which server architecture to use may depend on:

- security requirements
- performance and hardware requirements
- memory requirements
- disk requirements
- locally available administrative skills

Security Requirements

These requirements may be based solely on your authentication needs – but may also incorporate the relative security of the underlying system architecture.

Security requirements are also discussed in Part II.

chroot jail

CVSNT on Unix, Linux and Mac OS X may be set up in a *chroot jail* which prevents any hacker gaining access via the cvs server to the remainder of your system.

Authentication

CVSNT Server on Windows, Unix or Mac OS X can authenticate:

- to native operating system usernames/passwords
- to windows active directory usernames/passwords
  (on unix/mac uses the *winbind* daemon: *Samba* plus *winbind* must be installed and operational)
- to CVSNT's own internal database of usersnames and passwords

To use CVSNT's own internal database of usernames and passwords you must use the pserver or sserver protocols

See the section *Security Architecture* below, particularly the headings *What are the drawbacks to using server authentication* and *Why are security protocols/authentication mechanisms important* for more information.

Security Architecture

By choosing to implement CVSNT you have chosen the most secure Open Source Version Control solution available today. Most version control solutions provide very little security choices in their protocols. When making decisions about your CVS Security you will need to decide about the security of the network traffic as well as the authentication.

Security requirements are also discussed in Part II under the heading *Configuring Network Access*.

Security of the CVS repository

Once a person gains direct access to the CVS Repository or access to the CVS "admin" commands then it is possible for them to corrupt or remove audits, including changes made to your source code and documents.

The security of this information is therefore the lowest common denominator when setting up a secure CVS environment. Administrators should be chosen carefully and should be the only ones allowed direct access to the files that comprise the CVS repository.

Security of the CVS audit logs

If you are creating any user defined audit logs using the audit plugin or using *notifyinfo* or other trigger scripts then the location that you store the audit file and the users who have access to delete or amend it will provide the parameters for its integrity.

If a security log can be deleted or amended by any user then the value of that log is reduced. This is important for text log files as well as logs stored within a database.

Proper backup of these log files also need to be taken into consideration when planning your CVS security.

Why are security protocols/authentication mechanisms important

The authentication mechanism is separate to the other functions of the CVSNT Server. Until you successfully log on you cannot use any of the more sophisticated hacking techniques to damage files and folders.

Authentication can be further removed to the operating system of the server itself. For maximum security March Hare recommend allowing the server operating system to handle security: SSPI (on windows), and ssh (using a Unix server and high strength RSA keys). See the SSPI authentication section for details about the relative security of Kerberos and NTLM when using SSPI.

What are the drawbacks to using server authentication

Some administrators can be concerned that allowing the server operating system to authenticate users can pose a security threat in itself. The expectation being that if a hacker can bypass the operating system security then they have "free reign".

This can be mitigated by setting expiration times on the security tickets (eg: Microsoft Active Directory has a kerberos ticket lifetime that can be locked down so it doesn't automatically renew), and the CVSNT passwd file can be used to restrict authenticated users to a subset of the possible users on the server.

Native usernames versus CVS PASSWD usernames

CVSNT Server on Windows, Unix or Mac OS X can authenticate:

- to native operating system usernames/passwords
- to windows active directory usernames/passwords
  (on unix/mac uses the winbind daemon: Samba plus winbind must be installed and operational)
- to CVSNT's own internal database of usersnames and passwords.

To use CVSNT's own internal database of usernames and passwords you must use the pserver or sserver protocols.

Two factor server authentication (RSA SecurID)

Using a product such as RSA SecurID it is possible to set up two factor authentication for your operating system. If you have installed two factor authentication then CVSNT will automatically use it. See the section *Configuring Network Access* for more information.

What are the recommended security protocols/authentication

We recommend that pserver is disabled (CVSNT allows protocols to be disabled without the need to re-compile). For specific client and server protocols please see the tables below:

Server Platform Security Recommendation	***Protocol***
Server	
Windows	sspi,sserver[1]
Mac OS X	ssh[1]
Unix / Linux	ssh[1]

Note 1: See the SSPI authentication section for details about the relative security of Kerberos and NTLM when using SSPI

What is a "chroot jail" and how do I set one up

A "chroot jail" is a security mechanism for Unix based operating environments such as SuSE Linux, Red Hat Linux, Solaris, HPUX and Mac OS X. It is a place to install CVSNT so that no other files can be accessed either accidentally or by a hacker.

There is no equivalent mechanism for the Windows Operating System.

To create it you must set the Chroot variable in /etc/cvsnt/PServer and the CVSNT Server will *chroot* after doing the authentication - you no longer need to put any libraries in the chroot which is much safer (it just needs a */tmp* to put the temporary files in).

How do I disable insecure authentication protocols

Once you have decided which authentication protocol you will use based on your security requirements – disable all other protocols by deleting the DLL's (Windows) or Shared Libraries (Unix).

On Unix, Mac OS X and Linux the protocols are stored in the directory */usr/lib/cvsnt* by default. On Windows they are stored by default in *c:\Program Files\CVSNT*.

File and Directory Architecture

The physical format of the files that you version will affect the CM process.

- File types (Exports; Binary / Unicode / ASCII)
- File naming
- Include Files
- Size / Contents and the relationship to project activity

File Types

CM systems and CM theory largely require the actual file (or thing) being modified to be versioned (managed) by the tools. The most common scenario for a CM system to version something other than the actual file involved is in a database application. The tables themselves (or the meta file) are not versioned, but a script containing the SQL instructions to create that meta data.

If a copy of the thing (or file) is versioned then additional manual procedures must ensure:

- The *actual file* is updated when the CM system is updated
- The CM system is updated when the *actual file* is updated
- If the *actual file* is modified outside of the system and the CM system attempts to overwrite a newer version with an older one that some system prevents this
- If the *actual file* is being modified that a lock or notification is placed on the CM system so other users wanting to (or are) working on that file are informed.

File Naming

Names of file, or its location in a directory tree of files should not contain important information about the nature or function of the contents. The correct place for that information is in the CM system.

For example: `2005/June/common.h` is not a good name for a file since it is likely to need to change often. Instead the file `common/fin.h` can have many branches or many tags to differentiate different financial periods.

Poor filenames can lead to unnecessary refactoring of the repository (ie: name changes).

Include Files and Common Files

Files that are used by several other files (eg: a document template or an include file) should reside in a separate directory.

Some configuration management systems encourage the users to put the common file in the same directory as the file that uses it and have several links to this file.

This can lead to confusion as to the true nature of the file (ie: used by many – not specific to this document or project). Instead store common files in a separate module (or directory) and use branches for different variations used by different projects or users.

Often these common files will also require different security measures – eg: regular developers can change projects, but only administrators can change the `fin.h`. include file. CVSNT has ACL's that allow repository administrators to set different levels of access for users and groups. These ACL's are designed to be applied to directories instead of individual files.

Size and contents and the relationship to project activity

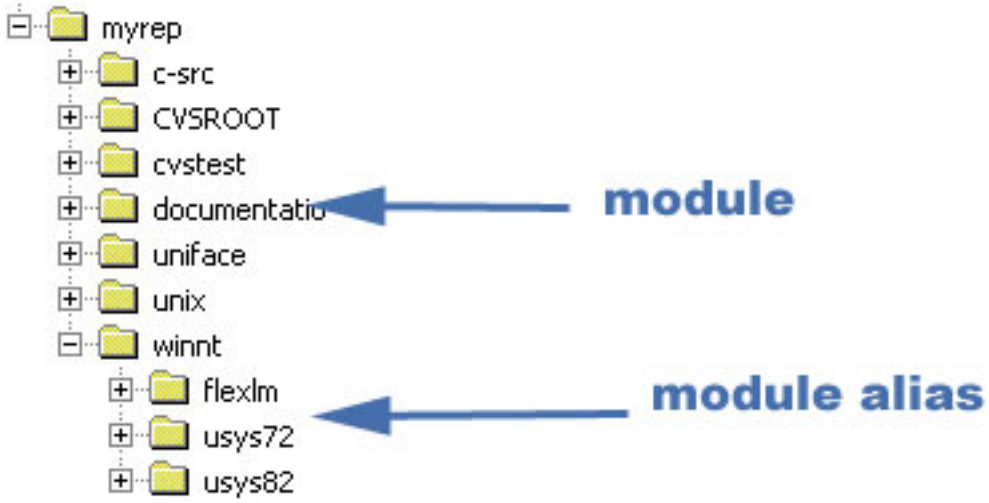
Large text or Unicode files in a well designed CM system are rare. If you have a large document or file check that it cannot be better tailored to the project structures you use. These guidelines can help you find if the document or file would be better divided:

- Different sections of the file / document are regularly maintained or modified by different people. Eg: Joe always makes modifications to section A, and Bill always makes changes to the Glossary. In this case the Glossary and each section are most likely candidates for separate versioning. Then Bill and Joe can easily make their changes at the same time as each other.
- The document or file is frequently modified or frequently the subject of contention (several people or projects wanting to modify it at the same time). If the file has smaller divisions this can reduce contention.
- The document or file is frequently modified and then unmodified by the same two or three teams or people. Eg: Team A set the `fin.h` file to `#define YEAR 2005`, then team B change `fin.h` to `#define YEAR 2006` then Team A set the `fin.h` to `#define YEAR 2005`. This indicates that Team A and Team B should be using different branches of the same file (branch "teamA" and branch "teamB") then the conflicts will not occur.

Organising your projects (modules, directories)

Also see the section above titled *File and Directory Architecture*.

Your CVS repository is organised in to projects – also known as modules. Each "top level" directory in CVS has a special meaning – these directories constitute modules. Modules may also exist at levels below the root and these are defined using module aliases.



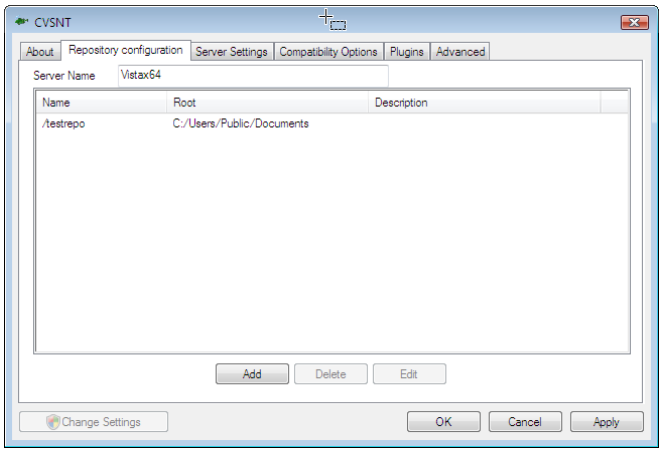
Multiple Repositories

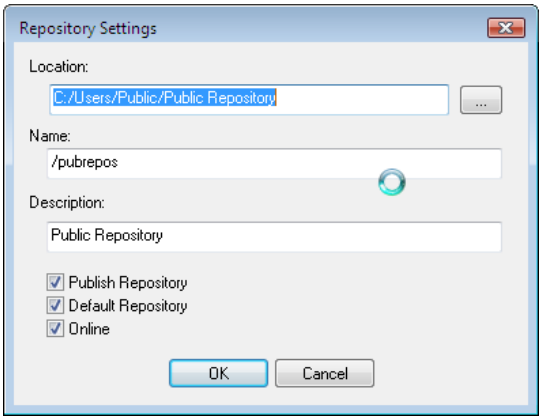
A single server may define multiple repositories. On windows every repository should be created on an NTFS disk.

There is no performance benefit or advantage to setting up multiple repositories however it may be required if:

- a lack of space on a single partition of a disk.
- setting up a server to host repositories for several companies
- The administrative files will be very different for each repository

The location of each repository is defined using the `/etc/cvsnt/PServer` configuration file on Unix or using the Windows Control Panel:





Email notification

CVSNT contains a trigger library which is capable of sending notification emails on commit, tag or notify. It allows you to put any contents in the emails, but the output format is fairly simple - it is no substitute for a purpose designed notification program.

Email sending is disabled by default. To configure it for use you must follow the following sections:

- Configure the commit support files
- Write the template
- Configure the server

Configure the commit support files

Each repository is created with a special administrative directory named `CVSROOT`. This directry contains several files that govern the behaviour of the email trigger:

CVSROOT/

- commit_email
- tag_email
- notify_email
- users

Template files

The commit support files `commit_email`, `tag_email` and `notify_email` contain the names of the template files to use for commit, tag and notify respectively. Each line in these files is a regular expression followed by a filename. The filename is always relative to the CVSROOT directory and may not be an absolute path for security reasons.

The first matching line for each directory committed is used. If there is no match the DEFAULT line is used.

The mapping file

The `users` file is used to lookup the username -> email mapping. This file is a list of colon separated username/email pairs. If this file does not exist or the username is not listed the default domain name set in the global configuration is used.

The checkoutlist

The names of each email configuration file should also be listed in the checkoutlist file. This ensures that the templates and mapping are available for the trigger to use during commit, tag and other operations.

Write the template

The template file is a simple text file listing the exact text of the email to send including headers. The To:, From:, Cc: and Bcc: lines are used by the sending software to determing the addresses to use. After the Subject: line there should be an empty line and there should be no empty lines before the Subject: line.

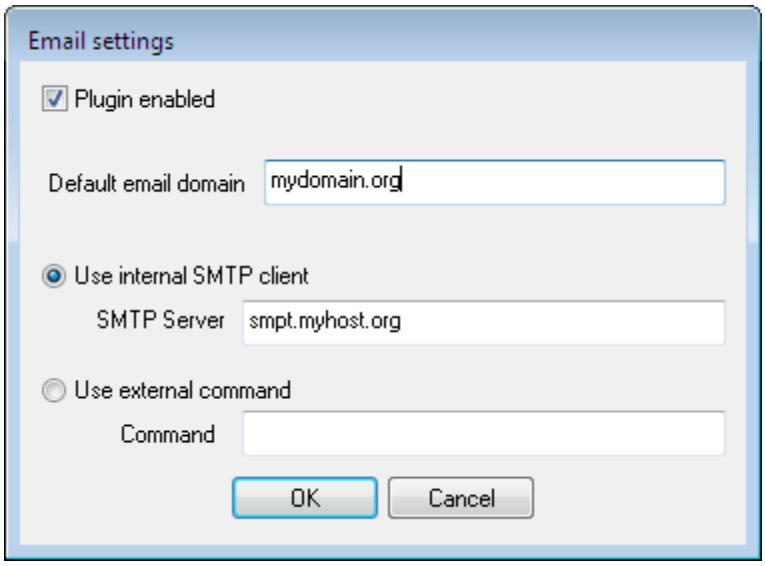
An example commit template is:

```
From: [email]
To: cvsnt_users@mycompany.com
Subject: Commit to [module]

CVSROOT:     [repository]
Module name: [module]
Changes by:  [email]           [date]
On host:     [hostname]

[begin_directory]
Directory: [directory]

[begin_file]
[change_type] [filename]   [tag]   [old_revision] -> [new_revision]   [bugid]
[end_file]
[end_directory]

Log message:
[message]
```

A number of replacements are done on the file to format it for final sending. This differs for each file, and is listed below.

Configure and enable the server plugin

For the CVSNT server to send emails the “Email notification extension” plugin must be enabled. Use the CVS Server Control Panel (on Windows) or the `/etc/cvsnt/Plugins` file (unix) to enable the plugin.



There are two ways that CVSNT can send email. The simplest is to set the SMTP Server and default domain in the global configuration (Control Panel in win32 the */etc/cvs/Pserver* configuration file in Unix) and let the internal SMTP client send the emails.

This will not work in the case where authenticaton is required or the server is not capable of SMTP. In these cases you instead set the Email Command. This command should take a list of 'to' addresses as parameters, and a raw RFC822 email as its standard input.

A suitable configuration for Unix systems is
`/usr/sbin/sendmail -i`

Similar programs exist for Windows from 3rd parties.

Keywords used in template files

The following are the global keywords used in the email template files.

Global Keyword	*Global Keyword Description*
Global keywords used in the email template files	
[hostname]	Client hostname, it known
[repository]	Repository name
[commitid] or [sessionid]	Session identifier
[server_hostname]	Local hostname of server
[user]	User who is performing the action
[email]	Email address, as looked up from `CVSROOT/users`
[date]	Date/time of action
[message]	Message associated with action, if any
[module]	Module associated with action
[begin_directory]	Lines between these tags (which must be on their own on the line) are repeated for each directory referenced by the operation.
[end_directory]	
[begin_file]	Lines between these tags (which must be on their own on the line) are repeated for each file referenced by the operation. These tags can only exist inside begin/end directory tags.
[end_file]	
[directory]	Current directory
[filename]	Curent file

Each type of template also has its own keywords that are used:

- Commit emails
- Notify emails
- Tag emails

commit emails

Commit Keyword	*Commit Keyword Description*
Commit keywords used in the commit email template file ONLY	
[old_revision]	Revision number of previous revision
[new_revision]	Revision number of new revision.
[tag]	Tag for file.
[change_type]	Code for change made by this commit.. 'M', 'A', etc.

tag emails

Tag Keyword	*Tag Keyword Description*
Tag keywords used in the Tag email template file ONLY	
[tag_type]	Type of tag operation.
[action]	What is being done with the tag.
[tag]	Tag for file.
[revision]	File revision that is being tagged.

notify emails

Notify Keyword	*Notify Keyword Description*
Tag keywords used in the Tag email template file ONLY	
[bugid]	Bug identifier(s) associated with this notification.
[tag]	Tag name of file.
[to_user]	Mapped user/e-mail to notify from CVSROOT/users file
[notify_type]	Notification type

Keeping Checked Out Copies

You may configure CVSNT to keep checked out (read only) copies of one or more braches of development. This is usually required:

- When using Uniface – so the IDF can "see" the model level objects when you are modifying a component.
- To keep a web site (or staging server) up to date
- For a build system to operate on

Three methods: shadow, postcommit and Make plugin

There are three ways to configure CVSNT to keep a checked out copy on the server:

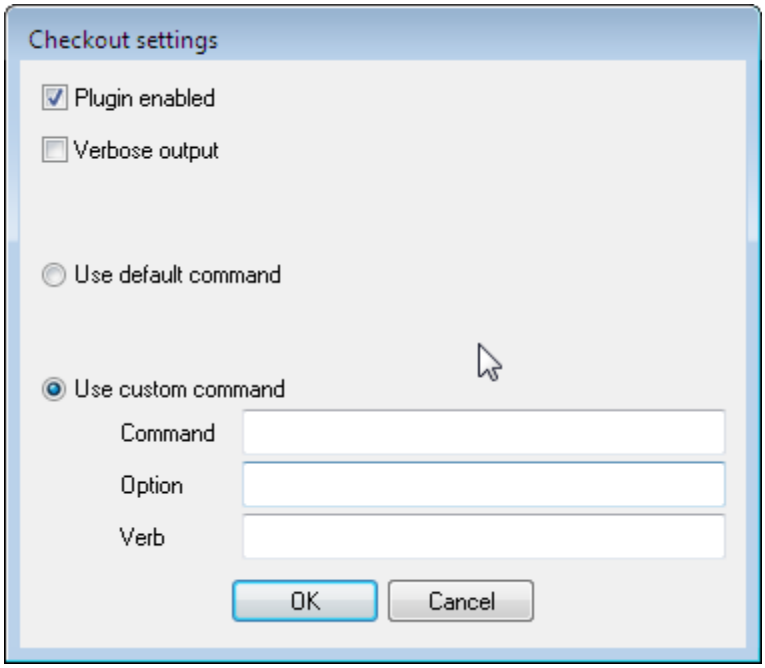
- Shadow
- Postcommit (script)
- Make plugin (script)

The older methods invoke a script to perform the "checkout" to keep the checked out copy up to date, wheras the newer is simply done by specifying the location of the shadow area.

The shadow file

The CVSROOT/shadow file is used by the checkout plugin to specify directories that will be automatically updated on checkout or tag.

In order for the shadow file to have any effect the "Automatic checkout extension" plugin must be enabled in the CVSNT control panel (Windows) or the `/etc/cvsnt/Plugins` configuration file on Unix/Linux/Mac OS X.



Here is an example (this should all be on one line):

```
^cyclic-pages HEAD /u/www/local-docs
```

This will cause checkins to repository module **cyclic-pages** (and all sub directories) to update the checked out tree in **/u/www/local-docs**.

Note that if the shadow copy does not exist already it will be created by the execution of the shadow command. Likewise if a shadow copy exists and new directories have been added to the module then these directories and files will also be checked out into the shadow copy. So it will always be a true representation of the current state of the module.

The shadow file works only on the physical file system level (inside the repository). This means that a module specified in the regular expression must match a physical module name in order to be recognized.

For example if you have created virtual modules inside the CVSROOT/modules file or CVSROOT/modules2 file you cannot specify such a module name in the shadow file.

Scripted methods

The differences in the scriptable options are:

Postcommit	*Make Plugin*
Advantages and Disadvantages of the two techniques	
Script to perform the checkout operation is called from the postcommit trigger.	Script to perform the checkout operation must be named build_make.bat (build_make.sh on unix) and is called from the make plugin.
Script is passed the standard parameters, including the module name. There is no standard parameter for the branch name.	Script is passed the module name, first directory name and the branch name
One script per repository	One script per server

Postcommit method

The way to do this is by having *postcommit* invoke *cvs update*. Here is an example for unix (this should all be on one line):

```
^my-pages            (date; cat; (sleep 2; cd /u/www/local-docs;
cvs -q update -t -d) &) >> $CVSROOT/CVSROOT/updatelog 2>&1
```

This will cause checkins to repository directories starting with my-pages to update the checked out tree in `/u/www/local-docs`.

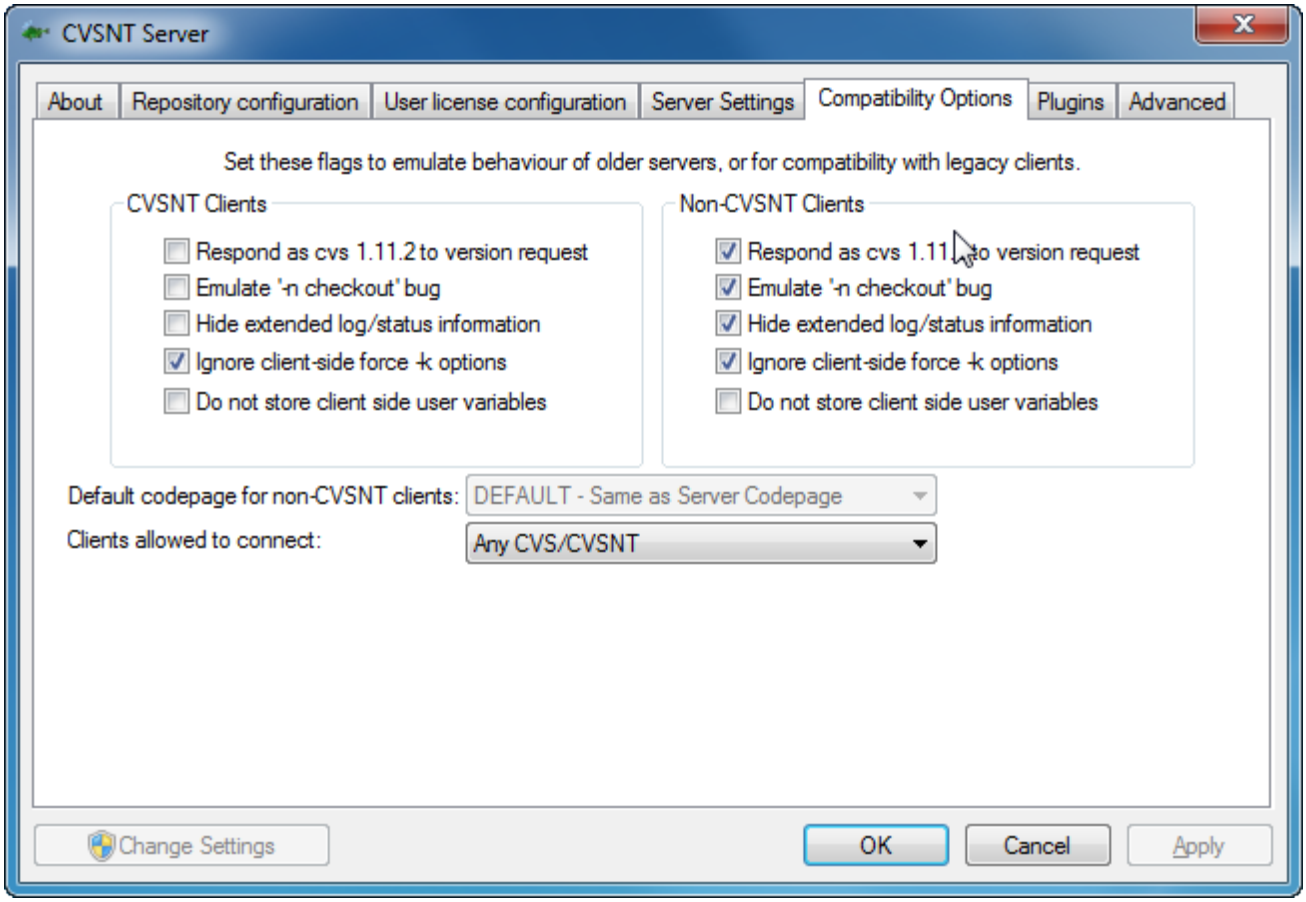
To achieve the same result on Windows you will need to write an MS-DOS Batch File.

Make plugin method

See the section *Integration with Make (Build Post Commt Trigger)* for further information.

Compatibility Options

CVS Suite Server has some options to enhance the compatibility with non-CVSNT clients that you may wish to configure using the `/etc/cvsnt/PServer` configuration file or the CVSNT Control Panel.



Respond as cvs 1.11.2 to version request (Compat<n>_OldVersion)

This option can be disabled or enabled for CVSNT clients, non-CVSNT clients or both CVSNT and non-CVSNT clients. This option is needed by some older versions of Eclipse and perhaps other 'smart' clients that have broken behaviour when they identify a server as being non-CVS.

Emulate '-n checkout' bug (Compat<n>_OldCheckout)

This option can be disabled or enabled for CVSNT clients, non-CVSNT clients or both CVSNT and non-CVSNT clients. Older versions of Eclipse may require this.

Hide extended log/status information (Compat<n>_HideStatus)

This option can be disabled or enabled for CVSNT clients, non-CVSNT clients or both CVSNT and non-CVSNT clients. This disables the extended information that the CVSNT server can provide for log and status requests.

Ignore client-side force -k options (Compat<n>_IgnoreWrappers)

This option can be disabled or enabled for CVSNT clients, non-CVSNT clients or both CVSNT and non-CVSNT clients. This disables the clients ability to set the binary/Unicode/text type of a file and forces the server to always determine the file type based on the server settings. See cvswrappers for more about how to set the server defaults.

Do not store client side user variables (Compat<n>_IgnoreWrappers)

Clients can send *user variables* to the server which are automatically stored with the revision. Enabling this option disables the storage of these variables in the repository.

Clients allowed to connect (AllowedClients)

This setting allows you to prevent non-CVSNT clients, or older CVSNT clients from connecting to the server.

Default codepage for non-CVSNT clients (Locale)

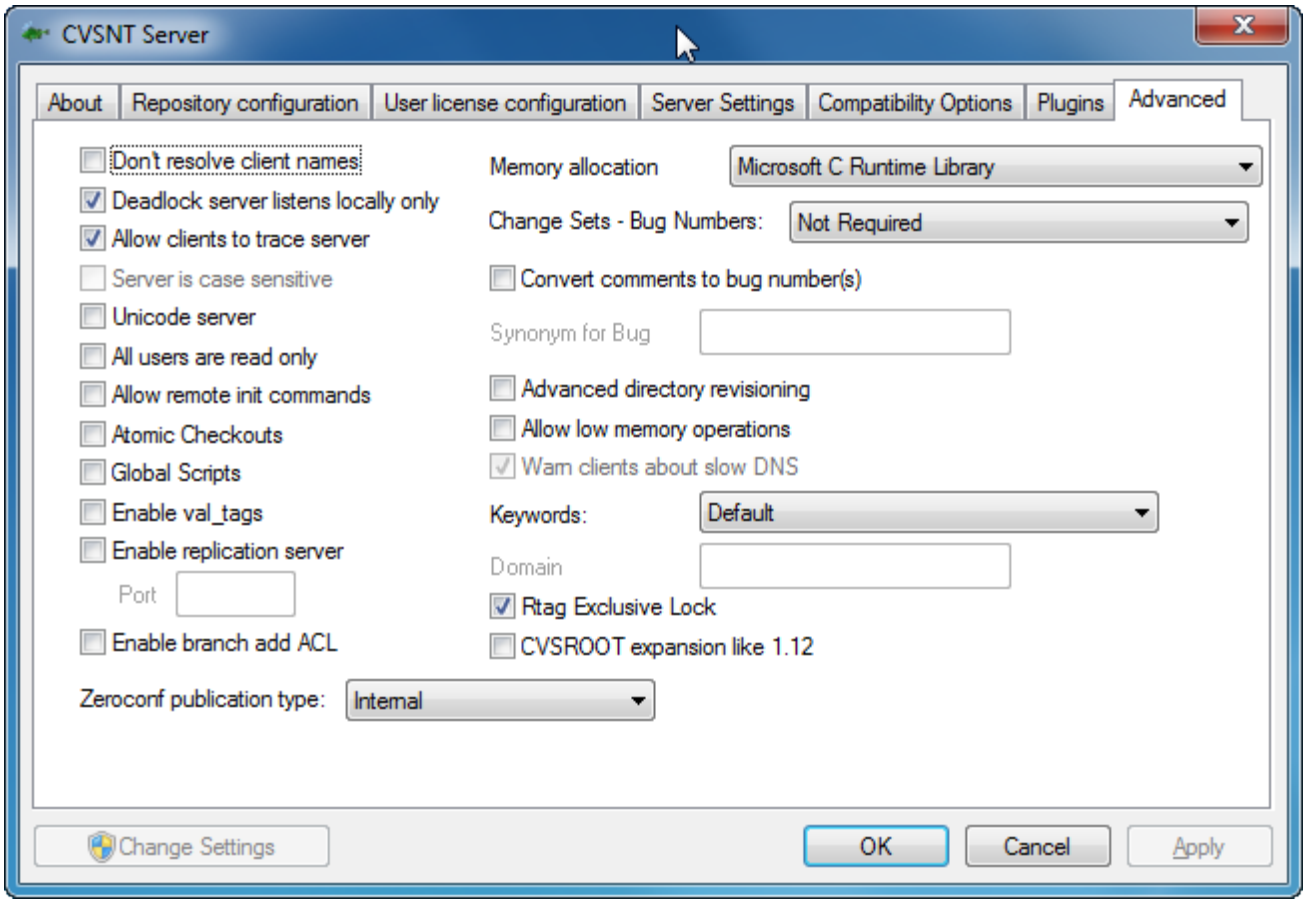
Non-CVSNT clients do not communicate to the server the codepage of the filenames. If the server and the client are using the same codepage this generates no errors, however if the client is using a different codepage to the server (eg: server is windows and the client is Mac OS X) then the filenames can become corrupted. If your non-CVSNT clients (eg: Netbeans or Eclipse) use a different codepage to the server then you should set it here.

Note 1: This conversion only occurs for non-CVSNT clients and does incur a performance penealty.

Note 2: SmartCVS clients are automatically detected and assigned the codepage ISO-8859-1.

Advanced Options

CVS Suite Server has some advanced options that you may wish to configure using the `/etc/cvsnt/PServer` configuration file or the CVSNT Control Panel:



Don't resolve client names (NoReverseDns)

The CVS Suite Server will always attempt to determine the name of a connected client using reverse DNS - this information is recorded in the Audit log. This option is enabled by default. If many clients are connecting to your server that your DNS does not have accurate reverse-dns information for, or your DNS lookup is very slow then you may need to disable this option.

Deadlock server listens locally only (LockServerLocal)

This option is enabled by default and should only be altered under instruction from a March Hare Software Engineer. This option allows a cluster of CVS Suite Servers to use a single CVS Deadlock Server (eg: in a cluster with a SAN).

Allow clients to trace server (AllowTrace)

If you need to send a bug report to March Hare Software you should enable this option and follow the instructions at the beginning of this book on providing traces to March Hare Software support.

Server is case sensitive

This option can only be enabled if the CVS Suite Server for Windows Case Sensitivity Driver is installed and licensed.

Due to limitations of the Microsoft Windows Server system, filenames in a CVS Suite Server for Windows repository must be case insensitive. Therefore if you are using CVS Suite Server you cannot store the linux files “Hello.txt” and “HELLO.txt” in the same directory (they can existin separate directories).

If you need to store case sensitive filenames using CVS Suite Server for Windows then you will require the CVS Suite Server for Windows Case Sensitivity Driver, ask your technical account manager if you require a quote for this product.

Alternatively you can use CVS Suite Server for Linux/Unix or Mac.

Unicode server

This option runs the server in Unicode using UTF-8 across the protocol. Enabling this option will cause scripts that were written assuming the server is running in ANSI mode to cease to function.

It is usually safe to enable this option on CVS Suite Server for Unix/Linux or Mac, and experimentation with CVS Suite Server for Windows may be required.

Activating this option enables the compatibility option *Default codepage for non-CVSNT clients* described in the section above.

All users are read only (ReadOnlyServer)

Prevent changes to the repository by all users – including administators. It may be convenient to enable this option during backup.

Allow remote init commands (RemoteInitRoot)

Allow server administrators to run the “cvs init” command across protocols other than `:local:`.

Atomic checkout

Enabling this option ensures that clients obtain a consistent checkout on a busy server. There is a significant performance penealty for enabling this option.

Enable global scripts

Allow activescripts outside of CVSROOT to be executed from a CVSNT trigger.

Enable val-tags (ValTags)

Check tags used on a client against a ‘valid tag list’ stored in the CVSROOT.

Enable replication server

See the section *Unison Repository Replication* for information on this option.

Enable branch add ACL (BranchAddUseSticky)

Please contact the support team for more information about this option.

Zeroconf publication type (ResponderType)

Do not modify this option unless instructed by a March Hare Software engineer.

Memory allocation type:

This option is available on Windows platforms only.

Change sets – Bug numbers (ResponderType)

There are four options:

- required on edit and commit
- required on commit
- required on edit
- Not required

Convert comments to bug numbers (MessageToBugs)

For non-CVS Suite or CVSNT clients, the user defined change set (ie: bug number) can be taken from the message, so a message string like "Fixed bug 123" will store the number 123 as the change set.

Synonym for Bug (BugSynonym)

This option is used in conjunction with *Convert comments to bug nunmbers* option when your organisation uses an alternative word to describe a bug. If you set the *synonym for Bug* to "SCR" and then enter a message like "Fixed scr 101" then CVS Suite Server will store the number 101 as the change set.

Advanced Directory Versioning

Do not set this option unless advised to by March Hare Software technical support. This option is used to alter how CVSNT server selects different revisions of a renamed file.

Allow low memory operations

When working with large files the server will check the amount of available memory against a guess of how much memory will be required for the commit operation. If the memory guessed is less than what is available then the operation will fail. This saves the client and the server the time of uploading and attempting to commit files which the server will then be unable to process.

Warn clients about slow DNS

This is an internal option only – it is not available for the CVSNT Administator to control.

RTAG – Exclusive Lock

When performing an *RTAG* the server should exclusively lock files sooner to help avoid deadlocks.

CVSROOT Expansion like 1.12

Enable expansion in administrative files (like *loginfo*) following the CVS 1.12 rules, eg:
`${CVSROOT}1234`

Backups

Since the CVS repository consists of individual RCS files for each document and object under version control – backup is simply a matter of backing up these files.

To ensure that you back up a consistent repository then:

- Stop the server from creating new server processes and wait until the existing processes have completed.
- Copy the repository to a new location
- Allow the server to create new processes again

If you are using a database such as MySQL for the Audit then the procedure is:

- Stop the server from creating new server processes and wait until the existing processes have completed.
- Stop the database
- Copy the repository to a new location
- Copy the database to a new location
- Start the database
- Allow the server to create new processes again

Your backup software should then backup the copied repository – not the live one.

Windows Backup

Here is an example for windows:

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>net stop cvsmanager
C:\>wait4cvs -q
C:\>net stop mysql
C:\>xcopy c:\myrepo \\server\cvsbackup /Q /S /C /H /R /O /Y
C:\>xcopy c:\mysql\data\audit \\server\cvsbackup /Q /S /C /H /R /O /Y
C:\>net start mysql
C:\>net start cvsmanager
```

On Windows you can use the MS-DOS command *at* to run a script like this at a certain time every day.

In the event of a need to restore the entire repository on new hardware you will also need a copy of the original installer for CVS Suite Server (and perhaps CVS Suite Client). March Hare Software recommends keeping copies of these installers in the same physical location as the backup files.

Unix Backup

Here is a sample backup script for linux/unix:

```
#!/bin/sh
/usr/local/mysql/bin/mysqladmin -u root --password=password --
       shutdown-timeout=0 shutdown
#this stop command is for linux
/etc/init.d/cvsmanager stop
# this stop command is for hpux
#/sbin/init.d/cvsmanager stop
ps -ef | grep -q c[v]smanager$
if [ $? -eq 1 ]; then
  telnet localhost 2401 < /dev/null | grep -q "Connected to
       localhost"
  while [ $? -eq 0 ]
  do
    echo "still more processes to finish"
    sleep 2
    telnet localhost 2401 < /dev/null | grep -q "Connected to
       localhost"
  done
  echo "all finished!"
else
  echo "You must first stop CVSMANAGER before continuing"
  exit 1
fi
cd /export/home/cvsnt/repository
ps -ef | grep -q "cvs[n]t preload"
while [ "$?" -eq 0 ]
do
   echo "cvs users still active, try in 30 seconds"
   sleep 30
   ps -ef | grep c[v]s$
done
tar cf /export/home/cvsnt/backup/backup.tar .
cd /usr/local/mysql/data
tar rf /Apps/cvs/backup/backup.tar audit mysql *.err
cd ..
#this start command is for linux
/etc/init.d/cvsmanager start
# this start command is for hpux
#/sbin/init.d/cvsmanager start
./bin/mysqld &
```

In the event of a need to restore the entire repository on new hardware you will also need a copy of the original installer for CVS Suite Server (and perhaps CVS Suite Client for your windows and mac clients). March Hare Software recommends keeping copies of these installers in the same physical location as the backup files.

Restore from Backup

In the event of server hardware failure or other catastrophic incident and you need to create a server from a backup follow this procedure:

- Install CVS Suite Server.
- Copy the repository to a new location
- Allow the server to create new processes again

After a restore of the server, the server may be missing some files or revisions to files that were committed from the clients to the server after the last backup. To resolve this we recommend that you backup and archive all your client sandboxes/workspaces after a restore of the server, and perform a clean checkout and clean build.

If some work is 'missing' then it can be copy/pasted from the saved/archived files.

DO NOT COPY DIRECTORIES FROM THE ARCHIVED WORKSPACES TO THE NEWLY CHECKED OUT SANDBOXES (ONLY COPY INDIVIDUAL FILES).

Build Systems

Build systems can be integrated with CVS so that at certain intervals the entire source code of the application can be checked for the need to re-compile. / re-create the objects (eg: compile the application).

The Build system can also run automated tests if you have any automated testing tools installed on the build server.

Build system configuration file

A configuration file defines for the build system the dependencies. This is often referred to as the *Makefile*. The configuration file defines:

- Resume.pdf is a result of Resume.doc.
- Hello.exe is a result of Hello.c and Hello.h
- Hello.testresults.txt is a result of Hello.exe
- Addcust.frm is a result of components.dir/addcust.xml and includeprocs.dir/mylib.xml

The build system configuration file also defines which tool to use for each object, eg: *gcc* or *idf*.

However to re-create your objects the build system will need a checked out copy of the source. This can either be a copy created (or updated from the previous days build) at a particular time or the *checked out copy* described above.

Unlike some other version control systems it is not necessary to modify your *Makefile* to check that each file is up to date. CVS will automatically recurse through your directory tree to ensure each file is up to date before the build begins.

When you check out or update the sandpit for the build you must instruct CVS to stamp the files with the date time that they were last checked in using the *-t* option on the *checkout* or *update* commands.

Identify which source code is responsible for which build

Once the build is complete you may want to *tag* each revision so it is immediately clear which files created that executable. For example if you created build 2020 then you can tag the versions with the command *cvs tag build2020*.

Reporting

CVS includes several commands for reporting on what has occurred in the repository, and in addition to these you can use the CVSNT triggers to define your own logging (eg: to a defect management system or to an SQL database).

Defining your own logging

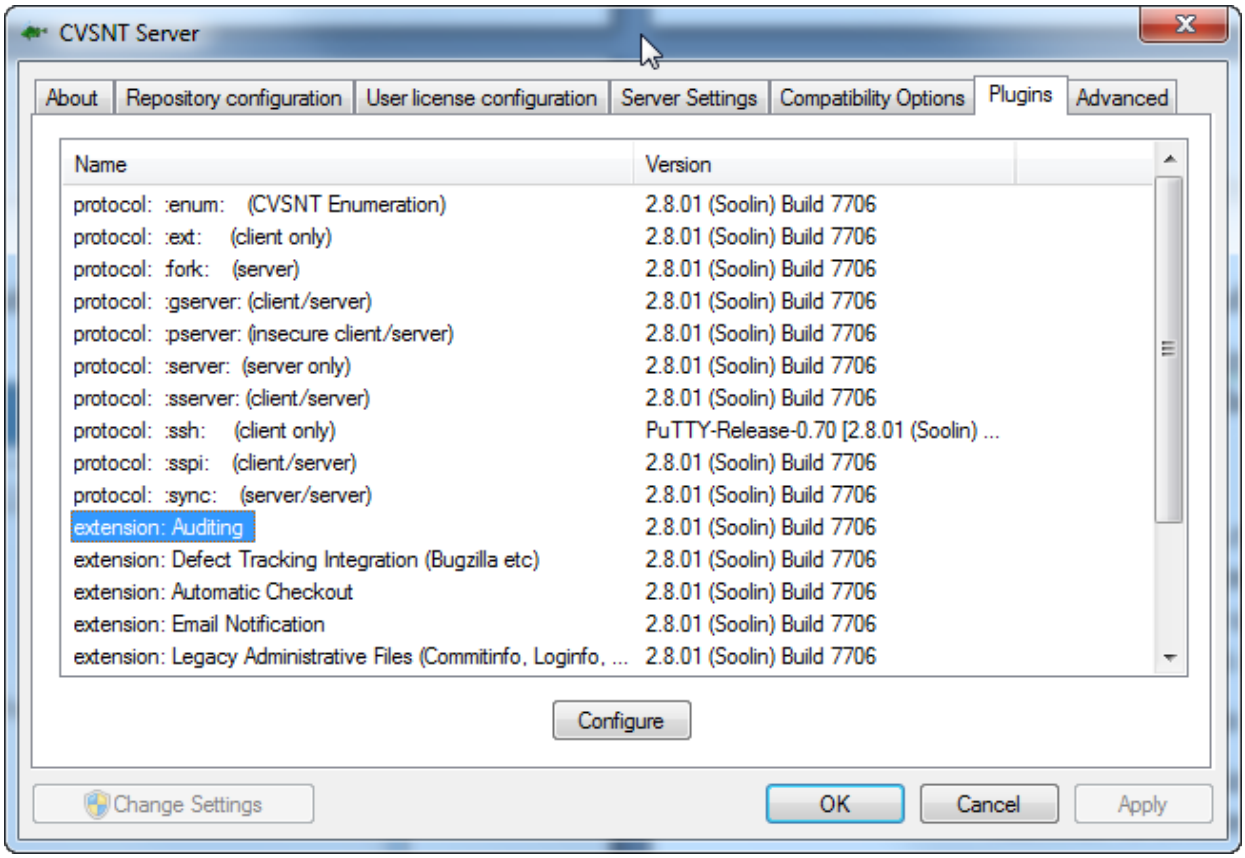
To define your own logging use the CVSNT triggers. Triggers of particular interest are:

- *notify* – log information about when files are locked/unlocked.
- *loginfo* – log checkin text and revision numbers

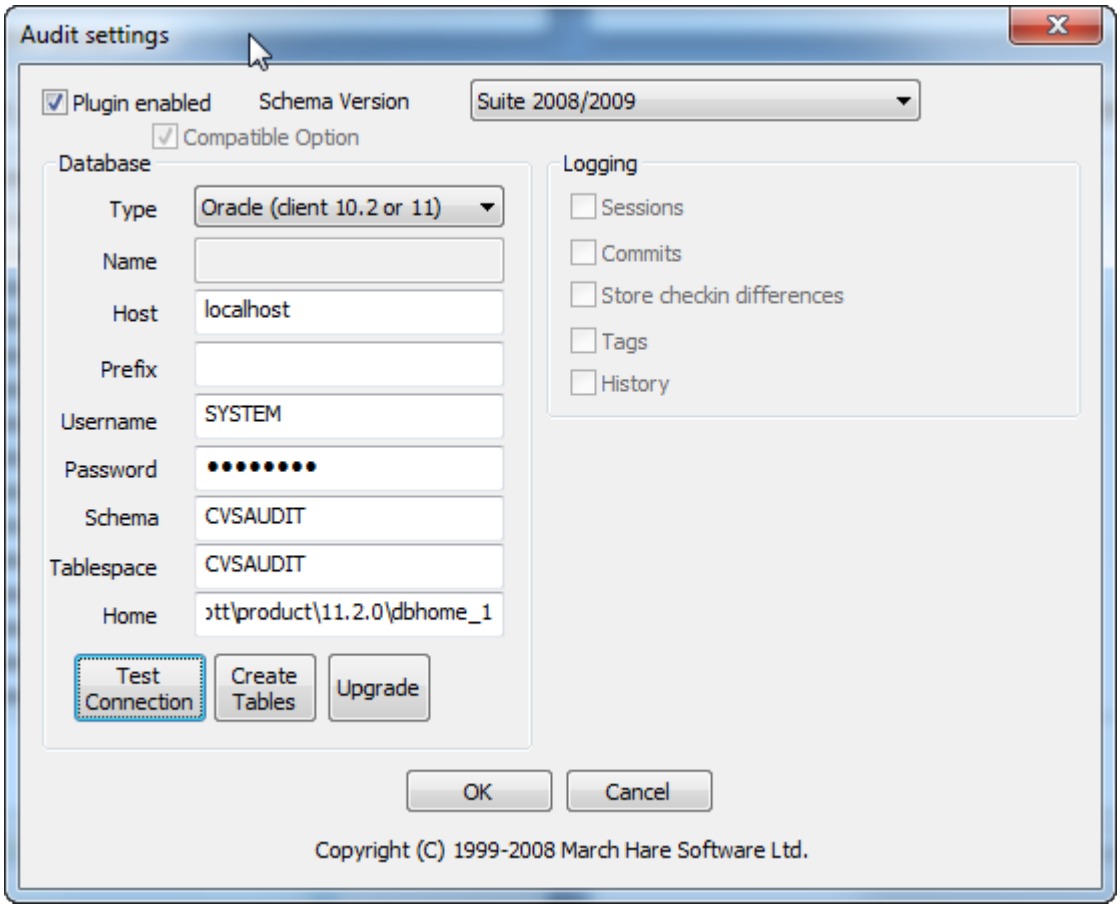
See the section on *triggers* below for more information,

Audit Plugin

March Hare Software LLC provide an audit plugin for support customers. This plugin stores audit information in a database (eg: MySQL or Oracle). On Windows it is configured using the CVSNT Server Control Panel on the server, on Linux it is configured using the `/etc/cvsnt/PServer` and `/etc/cvsnt/Plugins` files. See the section *Audit (to database)* for information on how to enable, configure and produce reports from the Audit Plugin.



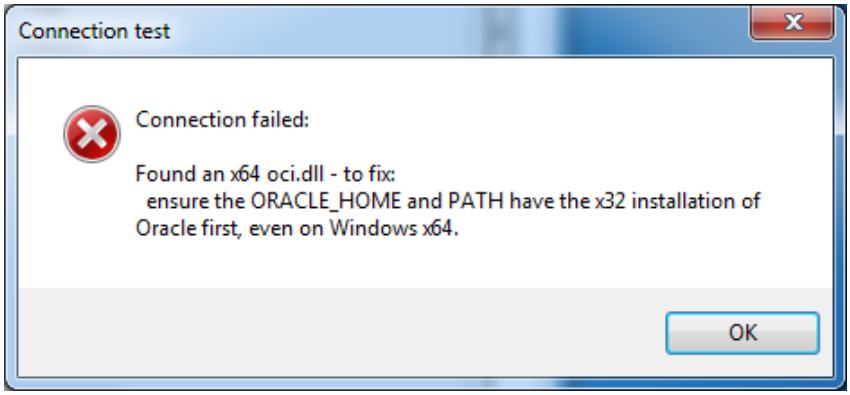
On windows Navigate to the Plugins page, select the Repository Auditing Extension and press the Configure button:



NOTE: The plugin will not create the database for you, but the "Create Tables" button can be used to add tables to an existing database.

Oracle / Postgres / DB2 Audit Plugin Database Client

The Oracle client software must be installed and be compatible with your CVS Suite Server release. Note: CVS Suite 2009R2 on Windows is an x32 application (even on Windows x64), so you will need the Oracle x32 (32 bit) client not the Oracle 64 bit client software. If you attempt to use an ORACLE_HOME with the incorrect bit size you will see this error:



SQLite Audit Plugin Database Creation

If you are using the SQLite database type then you can create an "empty" (zero byte) file and then use the plugin to create the tables.

Linux / Unix Audit Plugin Database Create Tables Procedure

On Linux or Unix you will need to use the database command line utilities to create the database using the supplied *create_table_xxxxx.sql* files.

Log and Rlog

The *cvs log* and *cvs rlog* commands allow you to print out log information for files. The *log* command requires a checked out copy of your documents (sandpit) and the *rlog* command does not. These commands can be used to selectively find all information about a single file or all files in a module.

This command is normally used in a batch program or script to filter the results.

If there are particular queries that you know you will want to run regularly then it is more efficient to create your own logging.

History

The cvs *history* command uses the *history* database file in the CVSROOT of the CVS repository to find specific information about a file.

The following events are recorded and can be recalled using *cvs history*:

- release
- checkout
- export
- tag
- update (may generate one of: collision, merge, copied, deleted)
- commit/check in (may generate one of: add, modify, remove)

Part II – Installation

Setting up CVS Suite Server

This half of the book is designed implement the theory of what you have learned in Part I. If you want to get a server running quickly and then begin to experiment then these are the steps you need to follow.

This section is divided up into what needs to be done on the server and what needs to be done on the client. They can be the same machine if you prefer, however the procedure is the same.

Note that setting up CVS Suite Server and Clients is NOT as simple as installing the software. The section *CVS Architecture*, and the this section *Setting up CVS Suite Server* and the next section *Server Administration* provide the necessary information to install and configure the CVS Suite system. If you do not follow these guides then severe performance problems can occur.

Support during an upgrade (including on weekends)

March Hare Software typically can ask an engineer to be available for customer technical support during the planned time of the upgrade and provide multiple contact paths to this person (eg: desk/pager/cell/email). You should provide at least 2 weeks notice if you think you may require out of hours support for your upgrade.

Note: for out of hours support (Public Holidays and Weekends), you must have a support contract that includes telephone support. Your support contract does not usually cover out-of hours, but we will try and accommodate your upgrade schedule depending on staff availability.

Note: the purpose of our technical support is to assist with problems directly attributable to the operation of our supplied software, consulting on project management, Windows / Solaris / HPUX / Linux administration or providing user training or CM training are not covered by your support contract.

March Hare Software can provide on site consulting services to assist with your upgrade (minimum 2 consecutive days per person). These services are typically booked out six weeks in advance.

Migrating CVS Suite Server between Linux Servers

If you are moving your CVS Suite or CVSNT repository from one Linux Server to another, eg: from Red Hat Enterprise Linux 6 to Red Hat Enterprise Linux 9 or from SuSE Enterprise Linux 11 to Ubuntu you can follow these generic migration instructions.

Gather Diagnostics

You can generate diagnostics for your existing server with this command:

```
echo ---- ver >> /tmp/cvsdiag-oldserver.txt
cvs ver >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- PServer > /tmp/cvsdiag-oldserver.txt
cat /etc/cvsnt/PServer >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- info -vb >> /tmp/cvsdiag-oldserver.txt
cvs info -vb >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- info -r localhost >> /tmp/cvsdiag-oldserver.txt
cvs info -r localhost >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- uname -a >> /tmp/cvsdiag-oldserver.txt
uname -a >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- redhat-release >> /tmp/cvsdiag-oldserver.txt
cat /etc/redhat-release >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- PAM >> /tmp/cvsdiag-oldserver.txt
cat /etc/pam.d/cvsnt >> /tmp/cvsdiag-oldserver.txt 2>&1
```

You can install CVS Suite on the new server and generate diagnostics with:

```
echo ---- ver >> /tmp/cvsdiag-server.txt
cvs ver >> /tmp/cvsdiag-server.txt 2>&1
echo ---- PServer > /tmp/cvsdiag-server.txt
cat /etc/cvsnt/PServer >> /tmp/cvsdiag-server.txt 2>&1
echo ---- info -vb >> /tmp/cvsdiag-server.txt
cvs info -vb >> /tmp/cvsdiag-server.txt 2>&1
echo ---- info -r localhost >> /tmp/cvsdiag-server.txt
cvs info -r localhost >> /tmp/cvsdiag-server.txt 2>&1
echo ---- uname -a >> /tmp/cvsdiag-server.txt
uname -a >> /tmp/cvsdiag-server.txt 2>&1
echo ---- redhat-release >> /tmp/cvsdiag-server.txt
cat /etc/redhat-release >> /tmp/cvsdiag-server.txt 2>&1
echo ---- PAM >> /tmp/cvsdiag-server.txt
cat /etc/pam.d/cvsnt >> /tmp/cvsdiag-server.txt 2>&1
```

If you send the diagnostics through we can check these for you and highlight any additional recommendations. We can also use this information to tailor requests for more specific diagnostics.

Linux Migration Guide

For this outline, "OLDHOST" is your current (eg: red hat enterprise linux 6) server and "NEWHOST" is your new (eg: Red Hat Enterprise Linux 9)) server. This is assuming a basic configuration, if you are using server-side scripting or want to test other integrations then consult the install guide and send an email to sales@march-hare.com if you need assistance.

Here's how to tell if you're using server-side scripting (and might need additional assistance):

- Check out your "old" repository CVSROOT
- Examine the contents of the files in there. Are there any "*info", "post*", "pre*", or "triggers" files that have scripts in them? Ignore lines that start with "#", they are comments.

If YES, then you may have server-side scripts that you are using. Send them to us if you have questions.

Before you begin:

- Commit any outstanding changes that you want included in the migration (optional, but it's simpler if they are all checked in)
- Remove any outstanding locks (use cvs editors to check)
- Backup your repository.

On NEWHOST:

- Log in as root
- Install CVS Suite, configure services, start services
- create test repository
- adjust function settings in /etc/cvsnt/PServer, /etc/cvsnt/Server and /etc/cvsnt/Plugins to match the settings on the OLDHOST. In particular:
  - compatibility settings
  - repository configuration
  - plugins (audit, bugzilla etc.)
  - advanced settings: don't resolve client names, change sets: required, etc. (try and avoid copying non functional settings like TEMP dir location)

Note: for repository configuration, the alias name you use should be the same on the new server as on the old server, but the physical location can be wherever is most suitable. Do NOT store your repository on a NAS (a SAN is OK). Your repository is like a database, so if you wouldn't store your Oracle database on that drive, it's not the place for your CVS repository.

The server process 'runs as' the client user – eg: the user fred – so that user must have 'full control' over the repository files and directories, eg /opt/repo/myrepo/drawings/pdf/

We recommend that all the users belong to one group (eg: cvsusers) and that group has full control over all the files and directories in the repository.

Note: you can generate diagnostics on OLDHOST and NEWHOST and compare them with diff to look for differences. Not all differences are bad/wrong. Feel free to contact the support team to discuss your results.

Note: it is possible to copy all server settings from the OLDHOST to NEWHOST, but we do NOT recommend this. It is better to start with 'new' default settings and just change the required settings.

On OLDHOST:

- Locate repository.
- COPY selected modules from OLDHOST repository to NEWHOST repository location. Don't copy the CVSROOT (administrative files) module.

On your client:

- Verify operation of the repository of NEWHOST. You should see the modules copied over, and be able to work with them.

On ALL clients, existing sandboxes will still be pointing to OLDHOST. You can update this a couple ways:

A) New checkout (recommended): Check out the files to a new location. Use the Checkout command to get a new working set and duplicate any changes that have not been committed from the old sandbox to the new one. Note that you can copy entire files, but DO NOT copy the "CVS" control folders between sandboxes.

B) Update sandbox control files: Within every folder of every sandbox, there are "CVS" folders that contain control files. You can alter these files to redirect the sandbox to the new server. We recommend using a tool that can find and replace text in multiple files in one operation, such as Ultra Edit (or many others):

   1) Back up the sandbox
   2) Edit EVERY CVS\Root file in your sandbox to represent the new CVSROOT connection string that you will use to connect to NEWHOST. For example, if your old CVSROOT is ":pserver:OLDHOST:/PATH/TO/REPOSITORY" then the new CVSROOT might be ":ssh:NEWHOST:/myrepo"
   3) Test the sandbox by doing a "cvs update -dP". If the result doesn't look correct you can restore your sandbox from the backup and try again. It is best to shut down the old CVS server while trying this to ensure you will get an error if you still have any control files pointing to the old server.

Getting Help with an Upgrade

If you still need help, please provide us with the following:

- Description of what was being attempted and the expected result.
- Screenshot or EXACT wording of the command issued and resulting error message
- CVSROOT “connection string” you are using to connect from your clients to OLDHOST and NEWHOST.
- Diagnostics from OLDHOST. See above example.
- Diagnostics from NEWHOST. See above example.
- Results of the following command. At a command prompt on NEWHOST, type:

```
cvs info -b
cvs info -r localhost
```

- your current repository administrative files from the repository

```
passwd
config
readers
users
admin
group
cvswrappers
checkoutlist
cvsrc
loginfo
modules
modules2
```

(some of these files may not be present)

Regular Updates

It's best to plan to keep CVS Suite Server (CVSNT) updated regularly (at least once a year). We regularly release security updates, as well as bug fixes and enhancements for CVS Suite. Our latest security newsletter is always available via this link:
https://www.march-hare.com/maillist/securitycvs.asp?OPENID=Browser

Our full list of security issues is available here:
https://www.march-hare.com/cvspro/security.htm

Consulting Help with Upgrades

Most customers perform migrations themselves, with technical support from us, but silver support customers are entitled to purchase on site/zoom consulting for migrations - this is typically 2-3 days work. This can be combined with things like Audit or Defect Tracking/Bugzilla install or setting up DR servers or hot standby or other multi-site options. Let the sales team know if you need a quote for this.

Upgrading Linux with CVS Suite

If you are upgrading an existing CVS or CVSNT server or client to a new version of Linx, eg: Red Hat Entperprise Linux 8 to Red Hat Enterprise Linux 9.5 follow these steps before continuing. Also read the section *Upgrading CVS Suite Server on Unix and Linux*

Note: CVSNT Server supports IPv4 and IPv6, but we recommend using IPv4 for all CVS and CVSNT server connections. If you require use of IPv6 please contact support.

Before RH upgrade:

- use 'cvs editors' to find any open locks, remove locks
- Test repository integrity and save log file
- Stop and Uninstall CVS Suite Server

After RH upgrade:

- check support for legacy System V init scripts (initscripts, chkconfig)
- check CVS Dependencies (glibc.i686 zlib.i686 libstdc++.i686 libxcrypt.i686)
- check port 2401 (firewall-cmd --zone=public --add-port=2401/tcp --permanent)
- Install (not upgrade) CVS Suite Server 2009 Build 9329
- Test repository integrity and compare log file using `diff`

Linux Upgrade Commands

Unix	***Syntax***
Check for locks	In a checked out sandbox: **`cvs editors`** `aproj/func/jobhist.cpp  scott  Wed Apr 16 12:37:37 2025 GMT  w11box` **`cvs unedit -u scott aproj/func/jobhist.cpp`** *`look for error messages`*
Check for repository errors	In a checked out sandbox: **`cvs log > /tmp/log-result-before.txt 2>&1`**
Uninstall	**`/etc/init.d/cvsmanager stop`** **`/etc/init.d/cvslockd stop`** **`rpm -e cvsnt cvs-suite-trigger`**
System V	**`yum install chkconfig initscripts`**
Dependencies	**`yum install glibc.i686 zlib.i686 libstdc++.i686 libxcrypt.i686`**
Firewall	**`firewall-cmd --zone=public --add-port=2401/tcp –permanent`**
Install	**`tar xf cvs-suite-2009-9329-rhel9-rpm.tar.gz`** (for RHEL v9) **`rpm --import key-public-cvs-suite-support.asc`** **`rpm -i cvsnt-2.8.01.9329-1.i386.rpm \`** **`  cvs-suite-triggers-2.8.01.9051-1.i386.rpm`** **`chkconfig --add cvslockd`** **`chkconfig --add cvsmanager`** **`/etc/init.d/cvslockd start`** **`/etc/init.d/cvsmanager start`**
Check for repository errors again	In a checked out sandbox: **`cvs log > /tmp/log-result-after.txt 2>&1`** **`diff -c /tmp/log-result-before.txt /tmp/log-result-after.txt`**

Upgrading CVS Suite Server on Unix and Linux

If you are upgrading an existing CVS or CVSNT server to CVSNT **2009** Build 9329 on Unix or Linux then follow these steps before continuing.

Upgrade Planning

March Hare Software expects customers upgrading Linux or Unix (Solaris or HPUX) CVSNT server installations to follow an upgrade plan similar to the following:

1) plan your upgrade and write an upgrade plan (see list of items that should be included in such a plan below)
2) submit your upgrade plan to March Hare Software and other stakeholders at least a week in advance of the planned upgrade for review (usually two weeks or more in advance of the upgrade)
3) review stakeholder comments and modify plan
4) submit your final upgrade plan to March Hare Software and other stakeholders at least a week in advance of the planned upgrade
5) execute plan

Research for your upgrade plan should include:

- eBook for "new" version of CVSNT
- Software Release notes
- Frequently Asked Questions list on web site
- Bugs reported/solved with that version on web site trouble ticket database
- other material as suggested by your technical account manager

The upgrade should be planned by a team that includes people with the following qualifications/skills:

- Linux / Solaris / HPUX administrator
- CVS server administrator
- Administrators for any 3rd party integrations (eg: mysql)

The upgrade plan should include the following at a minimum, but may contain additional information:

A) Current versions of all software including server operating system, server software, server run-time libraries and patches and dependent software versions (eg: MySQL).

B) Hardware diagram and details of how/if this hardware is changing in the upgrade (specific attention paid to disks, memory and network)

C) Changes to Linux / Solaris / HPUX planned as a part of the upgrade (kernel parameters, patches etc)

D) Current backup regime for the Linux / Solaris / HPUX server as well as any specific backup for the CVSNT repository

E) Software versions planned to be installed (this software should be downloaded and copied onto a CDROM before submission of the plan to the stakeholders)

F) Detailed (step by step) command execution for the installation

G) A copy of all modified configuration files (CVSROOT) and any new configuration files (eg: perl scripts called from CVSROOT/loginfo)

H) A copy of the `/etc/cvsnt/PServer` and `/etc/cvsnt/Plugins` configuration files

I) An exhaustive list of the clients that connect to this server including: cvs client, cvs client version, operating system, $CVSROOT

J) If the client will be changing as a part of the upgrade (eg: if some clients are run on the same physical machine as the server)

K) Impact analysis for the server changes, including what measures have been taken to address that impact (should specifically refer to the release notes for the "new" version wherever possible)

L) Impact analysis for the client changes, including what measures have been taken to address that impact (should specifically refer to the release notes for the "new" version wherever possible)

M) Date and Time of the planned installation and names and contact details for the people involved including roles and responsibilities

N) Detailed (step by step) Test set that will be used to verify the "new" installation including performance, functional and integration tests

O) Troubleshooting procedure for each test step should that test fail

P) Acceptance criteria (including performance timing, functional results and integration results)

Q) 2nd level support contact procedure (eg: March Hare Software, Hardware vendor [HP, Sun] etc)

R) Rollback plan should the "new" installation be deemed unacceptable

S) Backup and restore procedures for different scenarios

In addition it is very helpful if the vendors (such as March Hare Software) understand the purpose of the upgrade (eg: we want to upgrade to use new feature xyz), and a little about your environment and how you use the product.

If you are using a client or application that uses CVS in some way on the same Linux, HPUX or Solaris machine as the CVSNT Server the upgrade is more complex and the boundaries between steps (K) and (L) need to be clearly defined in the upgrade plan. Ie: what test do you run on the "server" to ensure that it is working before attempting to use the "client".

Outstanding work

Have all users commit their pending changes on all sandboxes served by the CVS or CVSNT Server you are about to upgrade.

This step is not strictly required - you can skip this step.

Backup the repository directory

Backup the repository by copying all the files to another disk or tape.

If you are upgrading from CVS to CVSNT then you should definitely backup the repository before continuing. CVSNT adds information to the repository that CVS does not understand, so this version will be your last copy of the repository compatible with the older CVS.

If you are upgrading from CVSNT to CVSNT then this step is optional.

Shutdown Services

Before starting the software upgrade you must ensure that the audit database is shutdown and none of these processes are running:

- cvs
- cvslockd

For example to stop the lock deamon on solaris (as root):

```
# ps -ef | grep c[v]slockd$
    root 22333     1  0 16:25:20 ?         0:00 /usr/local/bin/cvslockd
# kill -9 22333
```

To stop the lock deamon on hpux (as root):

```
# /sbin/init.d/cvslock stop
```

To stop the lock deamon on linux:

```
sudo /etc/init.d/cvslockd stop
```

Generate CVSNT Diagnostics on Linux

CVSNT on Linux does not include a *cvsdiag* program, so you can generate CVSNT diagnostics on Linux with:

```
echo ---- ver >> /tmp/cvsdiag-oldserver.txt
cvs ver >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- PServer > /tmp/cvsdiag-oldserver.txt
cat /etc/cvsnt/PServer >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- info -vb >> /tmp/cvsdiag-oldserver.txt
cvs info -vb >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- info -r localhost >> /tmp/cvsdiag-oldserver.txt
cvs info -r localhost >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- uname -a >> /tmp/cvsdiag-oldserver.txt
uname -a >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- redhat-release >> /tmp/cvsdiag-oldserver.txt
cat /etc/redhat-release >> /tmp/cvsdiag-oldserver.txt 2>&1
echo ---- PAM >> /tmp/cvsdiag-oldserver.txt
cat /etc/pam.d/cvsnt >> /tmp/cvsdiag-oldserver.txt 2>&1
```

Uninstall CVS

CVSNT and CVS use will not both install on a system at the same time in the default distributions. If you want to run CVS and CVSNT (or two versions of CVSNT) in parallel on one machine you should contact March Hare Professional Support.

Backup configuration files and certificates

The installer may destroy configuration files and certificates. Back these up manually to ensure they are not lost (e.g.: the `/etc/cvsnt/PServer` configuration file).

Upgrade CVSNT

Install CVSNT into the default directory using the procedure below *Quick guide to installing a new CVSNT server*. Note: Some systems require a different command for upgrading an installation, eg.

Linux Upgrade Commands

Unix	***Syntax***

Red Hat upgrade (all)	CVSNT has many optional components. You can find what you already have installed:
	`rpm -qa | grep cvs`
	`cvsnt-database-mysql-2.8.01.nnnn-1.i686`
	`cvsnt-2.8.01.nnnn -1.i686`
	`cvsnt-protocol-sserver-2.8.01.nnnn-1.i686`
	`cvs-suite-triggers-2.8.01.nnnn -1.i686`
	Unpack the tar/gz file for the version of RHEL you have:
	`tar xf cvs-suite-2009-9329-rh9-rpm.tar.gz` (for RHELv4)
	`tar xf cvs-suite-2009-9329-rhel5-rpm.tar.gz` (for RHELv5/v6)
	`tar xf cvs-suite-2009-9329-rhel7-rpm.tar.gz` (for RHELv7)
	`tar xf cvs-suite-2009-9329-rhel8-rpm.tar.gz` (for RHELv8)
	`tar xf cvs-suite-2009-9329-rhel9-rpm.tar.gz` (for RHELv9)

Red Hat ES v4/v5/v6 *(both x32 and x86_64 systems)*	`rpm -U cvsnt-2.8.01.9329-1.i386.rpm \`
	`cvs-suite-triggers-2.8.01.9329-1.i386.rpm \`
	`cvs-?????-database-2.8.01.9329-1.i386.rpm`
	Note: The Audit and Defect Tracking triggers of CVSNT require a supported database client library. For MySQL the client 3.23 is required not 4.0/4.1/5.x, there are three (3) ways to obtain this:
	a) with the shared client "compatible" support. Eg:
	`rpm -i MySQL-shared-compat-VERSION.i386.rpm`
	b) with the Backlevel MySQL shared libraries (from Red Hat Network) Eg:
	`rpm -i mysqlclient10-3.23.58-4.RHEL4.1.i386.rpm`
	c) with the Backlevel MySQL shared libraries (from up2date) Eg:
	`up2date mysqlclient10`
	If you do not already have cvs-suite-triggers installed, and you are not going to enable the features like auditing, then you can install the Linux CVSNT rpm with the "--nodeps" option:
	`rpm --nodeps -U cvsnt-2.8.01.9329-1.i386.rpm`

Red Hat Enterprise v7/v8/v9	**yum update -y glibc.i686 zlib.i686 libstdc++.i686 libxcrypt.i686** **rpm -U cvsnt-2.8.01.9329-1.i386.rpm ** **cvs-suite-triggers-2.8.01.9329-1.i386.rpm ** **cvsnt-protocol-?????-2.8.01.9329-1.i386.rpm ** **cvs-database-?????-2.8.01.9329-1.i386.rpm** **systemctl daemon-reload** Note: The Audit and Defect Tracking triggers of CVSNT require a supported database client library. For MySQL simply install the MariaDB library support. Eg: **yum update mariadb-libs.i686** NOTE: The Sserver protocol of CVSNT requires OpenSSL client library. Simply install the support. Eg: **yum update openssl-libs.i686** If you do not already have cvs-suite-triggers installed, and you are not going to enable the features like auditing, then you can install the Linux CVSNT rpm with the "--nodeps" option: **rpm --nodeps -U cvsnt-2.8.01.9329-1.i386.rpm**

Linux Upgrade Commands (continued)

SuSE Enterprise v9 x86_64 *contact sales for s390 availability*

```
tar xf cvs-suite-2.8.01.9329-sl9-rpm.tar.gz
rpm -U cvsnt-2.8.01.9329-1.i386.rpm \
     cvs-suite-triggers-2.8.01.9329-1.i386.rpm \
     cvs-?????-database-2.8.01.9329-1.i386.rpm
```

Note: the last item is optional and should replace ????? with the name of the audit database plugin (note: if you are using the Bugzilla plugin then you must install the MySQL database driver.

Ubuntu 14.04 LTS

Ask sales team for username and password then add the repository to /etc/apt/sources.list:

```
deb http://usr:pwd@unifacecm.de/webtools/bo/ubuntu trusty contrib
```

Now you can install the package directly from our repository:

```
sudo apt-get update
gpg --keyserver keyserver.ubuntu.com --recv-keys ECEF7A0A
gpg --export -a ECEF7A0A | sudo apt-key add -
sudo apt-get upgrade cvs-suite
```

Alternatively, download the files from the customer downloads and use:

```
cat key-cvs-suite-ECEF7A0A-text.txt | sudo apt-key add -
sudo dpkg -r cvs-suite
sudo dpkg -i cvs-suite-2.8.01.9329-1_amd64.deb.gz
```

Note: the Ubuntu edition of CVS Suite always ships with the trial SEK, and after installing the software you need to update the license in /etc/cvsnt/License. Therefore during update you will see a message like:

```
(Reading database ... xx files and directories currently installed.)
Preparing to unpack cvs-suite-2.8.01.9329-1_amd64.deb ...
/etc/init.d/cvs-suite: 38: test: 1: unexpected operator
Unpacking cvs-suite (2.8.01.9329) over (2.8.01.nnnn-1) ...
Setting up cvs-suite (2.8.01.9329) ...

Configuration file '/etc/cvsnt/License'
 ==> Modified (by you or by a script) since installation.
 ==> Package distributor has shipped an updated version.
   What would you like to do about it ?  Your options are:
    Y or I  : install the package maintainer's version
    N or O  : keep your currently-installed version
      D     : show the differences between the versions
      Z     : start a shell to examine the situation
 The default action is to keep your current version.
*** License (Y/I/N/O/D/Z) [default=N] ?
```

We recommend you use the default action to keep your current version of the License. Alternateively you can overwrite the license then update the file with your license key.

Debian

```
dpkg -r package-name
dpkg -i package-name
```

Each Unix operating system (HPUX, Solaris, AIX, Tru64) has its own upgrade installation command based on the default for that system.

Unix Upgrade Commands

Unix	***Syntax***

Solaris	`Pkgrm cvs-suite`
	`bunzip2 cvs-suite-2.8.01.9329-sol9-sparc32.bz2`
	`pkgadd -d cvs-suite-2.8.01.9329-sol9-sparc32`
	or
	`pkgrm cvs-suite`
	`bunzip2 cvs-suite-2.8.01.9329-sol9-sparc64.bz2`
	`pkgadd -d cvs-suite-2.8.01.9329-sol9-sparc64`
	Note: Unless you will be committing very large revisions (>1GB) install the sparc32 edition on 64-bit Sparc systems.

HPUX pa-risc requires patches `PHCO 33711`, `PHNE_34135` or later	`compress -d /tmp/cvs-suite-2.8.01.9329-hppa-hpux32.depot.Z`
	`swinstall -s /tmp/cvs-suite-2.8.01.9329-hppa-hpux32.depot cvs-suite`
	or
	`compress -d /tmp/cvs-suite-2.8.01.9329-hppa-hpux64.depot.Z`
	`swinstall -s /tmp/cvs-suite-2.8.01.9329-hppa-hpux64.depot cvs-suite`
	Note: Unless you will be committing very large revisions (>1GB) install the hpux32 edition on 64-bit HPUX 11i systems.

HPUX itanium	`compress -d /tmp/cvs-suite-2.8.01.9329-ia64-hpux64.depot.Z` `swinstall -s /tmp/cvs-suite-2.8.01.9329-ia64-hpux64.depot cvs-suite`
Tru 64	Platform Build – please contact your sales representative
OS400	Platform Build – please contact your sales representative
AIX 5.1	Platform Build – please contact your sales representative

Quick guide to installing a new CVS Suite server

This section provides a quick guide, or a checklist of the actions needed to install a new CVSNT server. For more information read the detailed guide in *Server Administration* below.

Upgrading

If you are upgrading an existing CVS, CVSNT server or CVS Suite Server to CVS Suite 2009 read *Upgrading CVS Suite Server*.

Pre-Requisites

Before Installing you must ensure:

- server and clients are synchronised to the a reliable time source
- if CVS Suite client users will authenticate to a domain the server operatin system software should already be joined to that domain
- you know which protocol you will use to connect the client and server

Ensure Server is Synchronised to a Reliable Time Source

For CVS Authentication to operate correctly, and to maximise performance of the CVS System the Server must be synchronised to a reliable time source. CVS Clients must have their time synchronised to the CVS Server or the same time source as the CVS Server. If the CVS Client will use a network share or disk as a workspace / sandbox then the network share or disk should be synchronised to the same reliable time source as the CVS Server.

Server operating system is joined to a domain (if using domain authentication)

Ensure operating system software is installed and up to date (eg: using Windows Update or `up2date` or `yum`). If the server will authenticate users to a domain, ensure the server operating system is joined to that domain already.

Choose a Protocol

Choose a communications protocol that suits your client connectivity requirements and your security requirements. For most users with a Windows client this will be SSPI. The SSPI protocol can be used on a Unix or Linux server.

Generally as a rule - do not use the *:local:* protocol except when instructed to do so by a support engineer from March Hare Software LLC. The *:local:* protocol is always installed.

Install CVSNT on the server

Linux

Each linux operating system has its own installation command based on the default for that system. On Linux you can delete the unwanted protocols by simply not installing them. The table below shows the commands required to install CVS Suite Server and Command Line Client on the most common Linux systems.

Note: All Red Hat installs are x32 even on x64 systems (including Red Hat 7/8/9). All SuSE and Ubuntu installs are x64 – there is no x32 package available for SuSE or Ubuntu. There is currently no s390 (IBM Z) build available for Linux, eg: SuSE 10, 11, 12 or 15 – please contact our pre-sales technical support team for pricing.

Linux Install Commands

System	***Syntax***

Red Hat ES v4/v5/v6 *(both x32 and x86_64 systems)*	`sudo rpm -e cvs` (erases the older cvs install)
	`tar xf cvs-suite-2009-9329-rh9-rpm.tar.gz` (for RHELv4)
	`tar xf cvs-suite-2009-9329-rhel5-rpm.tar.gz` (for RHELv5/v6)
	`sudo rpm --import key-public-cvs-suite-support.asc`
	`sudo rpm -i cvsnt-2009-9329-1.i386.rpm`
	`sudo rpm -i cvs-suite-triggers-2009-9329-1.i386.rpm`
	
	If you are not going to enable the auditing then you can install the Linux CVSNT rpm with the "-nodeps" option.
	`rpm --nodeps -i cvsnt-2009-9329-1.i386.rpm`

Red Hat Enterprise v7/v8/v9	`sudo rpm -e cvs` (erases the older cvs install)
	`tar xf cvs-suite-2009-9329-rhel7-rpm.tar.gz` (for RHELv7)
	`tar xf cvs-suite-2009-9329-rhel8-rpm.tar.gz` (for RHELv8)
	`tar xf cvs-suite-2009-9329-rhel9-rpm.tar.gz` (for RHELv9)
	`sudo rpm --import key-public-cvs-suite-support.asc`
	`sudo rpm -i cvsnt-2.8.01.9329-1.i686.rpm \`
	`cvsnt-protocol-?????-2.8.01.9329-1.i386.rpm \`
	`cvs-database-?????-2.8.01.9329-1.i386.rpm`
	
	If you are not going to enable the auditing etc. then you can install only cvsnt:
	`sudo rpm -i cvsnt-2.8.01.9329-1.i686.rpm`

SuSE Enterprise v9 x86_64 *contact sales for s390 availability*	`tar xf cvs-suite-2.8.01.9329-sl9-rpm.tar.gz`
	`rpm -i cvsnt-2.8.01.9329-1.i386.rpm`
	`rpm -i cvs-suite-triggers-2.8.01.9329-1.i386.rpm`
	
	Note 1: Use YaST to update the `glibc` package to at least version `2.3.3-98.28`.
	Note 2: The Audit trigger of CVSNT requires a supported database client library. For MySQL the client 3.23 is required not 4.x/5.x, the easiest way to install this is with the "compatible" support. Eg:
	`rpm -i MySQL-shared-compat-VERSION.i386.rpm`
	
	If you are not going to enable the auditing then you can install the Linux CVSNT rpm with the "--nodeps" option.
	`rpm --nodeps -i cvsnt-2.8.01.9329-1.i386.rpm`

Debian	`dpkg -i` *package-name* Note 1: Please contact our sales team for details as to the availability of this package

Ubuntu 14.04 LTS	Ask the sales team for a password then add the repository to /etc/apt/sources.list:
	`deb http://`*usr*`:`*pwd*`@unifacecm.de/webtools/bo/ubuntu trusty contrib`
	
	Now you can install the package directly from our repository:
	`sudo apt-get update`
	`gpg --keyserver keyserver.ubuntu.com --recv-keys ECEF7A0A`
	`gpg --export -a ECEF7A0A | sudo apt-key add -`
	`sudo apt-get upgrade cvs-suite`
	
	Alternatively, download the files from the customer downloads and use:
	`cat key-cvs-suite-ECEF7A0A-text.txt | sudo apt-key add -`
	`sudo dpkg -i cvs-suite-2.8.01.9329-1_amd64.deb.gz`

Linux Core Dependencies

Newer releases of Linux may require core compatibility files to be installed.

Linux Install Commands	
Red Hat v5	Note 1: Some customers report needing the following additional library: `yum install compat-libstdc++-33.i386`
Red Hat v6 x64	`yum install glibc.i686`
Red Hat v7 x64 &	`yum install -y glibc.i686 zlib.i686 libstdc++.i686 libxcrypt.i686`

Red Hat v8 x64 & Red Hat v9 x64	Note 1: Some customers report needing the following additional library:
	`yum install yum-utils`
	`yum-config-manager --enable rhel-7-server-optional-rpms`
	`yum install compat-libstdc++-33.i686`

Linux MySQL or MariaDB Dependencies

If you are using any integration between CVSNT server and MySQL or MariaDB databases, then you will need to install the appropriate connector library. Eg: if using CVSNT Bugzilla integration or Audit to MySQL or Audit to MariaDB.

Linux Install Commands	
System	***Syntax***
Red Hat v5	`rpm -i mysqlclient10-3.23.58-4.RHEL4.1.x86_64.rpm` Note 1: you can download this library from the Red Hat Netowrk

Red Hat v6	NOTE: The Audit trigger of CVSNT requires a supported database client library. For MySQL the client 3.23 is required not 4.0/4.1/5.x, there are three (3) ways to get this
	a) with the shared client "compatible" support. Eg:
	`rpm -i MySQL-shared-compat-VERSION.i386.rpm`
	b) with the Backlevel MySQL shared libraries (from Red Hat Network) Eg:
	`rpm -i mysqlclient10-3.23.58-4.RHEL4.1.i386.rpm`
	c) with the Backlevel MySQL shared libraries (from up2date) Eg:
	`up2date mysqlclient10`

Red Hat v7/v8/v9	`yum install mariadb-libs.i686`

Linux SSL Dependencies

Newer releases of Linux may require SSL compatibility files to be installed.

Linux Install Commands	
System	***Syntax***
Red Hat v6	`rpm -i openssl098e-0.9.8e-17.el6.i686.rpm` Note 1: https://access.redhat.com/solutions/15835
Red Hat v7	`yum install openssl-libs.i686`

Unix

Each operating system (Mac OS X, Windows, Linux, HPUX, Solaris) has its own installation command based on the default for that system. On Unix you can delete the unwanted protocols after installation.

Unix Install Commands	
System	***Syntax***
Mac OS X	The Package Manager is a Graphical Tool.

Solaris	`bunzip2 cvs-suite-2.8.01.9329-sol9-sparc32.bz2`
	`pkgadd -d cvs-suite-2.8.01.9329-sol9-sparc32`
	or
	`bunzip2 cvs-suite-2.8.01.9329-sol9-sparc64.bz2`
	`pkgadd -d cvs-suite-2.8.01.9329-sol9-sparc64`
	Note: Unless you will be committing very large revisions (>1GB) install the sparc32 edition on 64-bit Sparc systems.

HPUX pa-risc requires patches `PHCO 33711`, `PHNE_34135` or later	`compress -d /tmp/cvs-suite-2.8.01.9329-hppa-hpux32.depot.Z`
	`swinstall -s /tmp/cvs-suite-2.8.01.9329-hppa-hpux32.depot cvs-suite`
	or
	`compress -d /tmp/cvs-suite-2.8.01.9329-hppa-hpux64.depot.Z`
	`swinstall -s /tmp/cvs-suite-2.8.01.9329-hppa-hpux64.depot cvs-suite`
	Note: Unless you will be committing very large revisions (>1GB) install the hpux32 edition on 64-bit pa-risc systems.

HPUX itanium	`compress -d /tmp/cvs-suite-2.8.01.9329-ia64-hpux64.depot.Z` `swinstall -s /tmp/cvs-suite-2.8.01.9329-ia64-hpux64.depot cvs-suite`
Tru 64	Platform Build – please contact your sales representative
OS400	Please see *CVS Suite for iSeries installation guide*
AIX 5.1	Platform Build – please contact your sales representative

Unix Core Dependencies

CVSNT on Unix may require GNU iconv and GNU gcc run time libraries to be installed. These may be available in the customer area of the March Hare web site. Here is a brief list of where else to find additional libraries.

Unix Install Commands	
Unix	***Localtion and Minimum Version Required***
Solaris	`http://www.sunfreeware.com` packages *libgcc-3.4.2* and *libiconv-1.8*. optional: `http://www.mysql.com` package *mysql-max-4.1.x*. `http://www.sunfreeware.com` package *sqlite3*.
HPUX	`http://hpux.connect.org.uk` packages *libiconv-1.9.2* and *gcc-4.0.2*. optional: `http://www.mysql.com` package *mysql-max-4.1.x*. March Hare Customer Area package *sqlite3*.

Starting the Servers

If you only intend to use CVSNT on Linux as a client, you can skip this step.

Unix and Linux

You must configure the system to:

- *Start the deadlock server (cvslockd)*
  ⇒ *you can start it manually by:*
  /etc/init.d/cvslockd start
- *Start the high performance server at startup (cvsmanager)*
  ⇒ *you can start it manually by:*
  /etc/init.d/cvsmanager start

You do not need to configure /etc/inetd.conf (or xinetd.conf) with CVS Suite 2009.

Allowing external access via port 2401

If you only intend to use CVSNT on Linux as a client, you can skip this step.

Red Hat Enterprise Linux 7

Red Hat use firewalld by default in Enterprise Linux 7. See this article at how to open a port:

https://access.redhat.com/solutions/1443293

All the in-built CVSNT protocols use port 2401 (Pserver, SServer, Gserver). If you are using SSH to connect then CVSNT SSH protocol will use the SSH port (default: 22).

You can disable firewalld this and use iptables instead[5].

Red Hat Enterprise Linux 6

Here is a sample /etc/sysconfig/iptables (ensure this is added before the COMMIT line):

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 2401 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp --dport 2401 -j ACCEPT
```

Red Hat Enterprise Linux 4 and 5

Here is a sample /etc/sysconfig/iptables (ensure this is added before the COMMIT line):

```
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 2401 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m udp -p udp --dport 2401 -j ACCEPT
```

[5] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-setting_and_controlling_ip_sets_using_iptables

Here is a sample output from the `/sbin/iptables` Red Hat Linux command:

```
[root@rhelv4 repo]# /sbin/iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
RH-Firewall-1-INPUT  all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
RH-Firewall-1-INPUT  all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain RH-Firewall-1-INPUT (2 references)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
ACCEPT     icmp --  anywhere             anywhere            icmp any
ACCEPT     ipv6-crypt--  anywhere            anywhere
ACCEPT     ipv6-auth--  anywhere            anywhere
ACCEPT     udp  --  anywhere             224.0.0.251         udp dpt:5353
ACCEPT     udp  --  anywhere             anywhere            udp dpt:ipp
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:https
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:ftp
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:smtps
ACCEPT     udp  --  anywhere             anywhere            state NEW udp dpt:smtps
ACCEPT     tcp  --  anywhere             anywhere            state NEW tcp dpt:cvspserver
ACCEPT     udp  --  anywhere             anywhere            state NEW udp dpt:cvspserver
REJECT     all  --  anywhere             anywhere            reject-with icmp-host-prohibited
```

Here is a sample `/etc/services`:

```
cvspserver      2401/udp    # cvspserver
cvspserver      2401/tcp    # cvspserver
```

You may also need to configure `/etc/hosts.allow`. If `hosts.allow` or `hosts.deny` contains any lines then ALL client protocols must be specified. The easiest way is to define port 2401 in `/etc/services` (as above) and then hosts.allow like

```
cvspserver : all
```

You do not need to configure `/etc/inetd.conf` (or xinetd.conf) with CVS Suite 2009.

Optional Server Configuration

If you only intend to use CVSNT on Linux as a client, you can skip this step.

Unix and Linux

You can configure the following optional system features:

- *Auto start deadlock server and high performance server service on reboot*
  - ⇒ *CVSNT Deadlock Server*
    chkconfig --add cvslockd
  - ⇒ *CVSNT High Performance Server Service*
    chkconfig --add cvsmanager

- *Configure CVSNT Server using the configuration files*
  - ⇒ *Copy the sample files:*
    cp /etc/cvsnt/PServer.example /etc/cvsnt/PServer
    cp /etc/cvsnt/Plugins.example /etc/cvsnt/Plugins
    cp /etc/cvsnt/server.example /etc/cvsnt/server
    cp /etc/cvsnt/Bugs.example /etc/cvsnt/Bugs
    cp /etc/cvsnt/Make.example /etc/cvsnt/Make
- *PAM*
  - ⇒ *So CVS users can be authenticated by pluggable authentication (usually for Windows Active Directory / Domain authentication)*

Here is a sample `/etc/pam.d/cvsnt`:

```
#%PAM-1.0
auth    sufficient      /lib/security/pam_winbind.so
auth    required        /lib/security/pam_unix.so
account sufficient      /lib/security/pam_winbind.so
account required        /lib/security/pam_unix.so
```

For RHEL5 and later, this is the recommended PAM configuration for CVS Suite that will use the system authentication preferences set up by system-config-authentication:
`/etc/pam.d/cvsnt`

```
#%PAM-1.0
auth       include     system-auth
account    include     system-auth
```

This is the equivalent configuration for systems below RHEL5:

```
#%PAM-1.0
auth       required    pam_stack.so service=system-auth
account    required    pam_stack.so service=system-auth
```

Configure the server

If you only intend to use CVSNT on Linux as a client, you can skip this step.

As a minimum you should configure the following:

- *If you have Eclipse clients:*
  - ⇒ *Switch on all the compatibility options for non-CVSNT clients*
- *Set the Temporary Directory*
- *Configure the Build Integration*
- *Configure the Bugzilla Integration*
- *Configure the Audit Integration*

Mac OS X, Linux, Unix

To configure the server edit the */etc/cvsnt/PServer* file.

Create a repository

Mac OS X, Linux, Unix

Use the following command:

```
cvs -d :local:/usr/local/myrepo init -a alias -d description
```

eg:

```
cvs -d :local:/usr/local/myrepo init -a /myrepo -d "my repository"
```

This command will perform the following actions:

- *Create the directory* `/usr/local/myrepo/CVSROOT`
  - ⇒ *This directory contains the CVS Server Administration files*
- *Alter the configuration file* `/etc/cvsnt/PServer`
  - ⇒ *This file must include an entry for each repository*

NOTE: a client accessing this server would use a CVSROOT that uses the alias location of the repository, not the real location, eg:

```
cvs –d :ssh:hostname:/myrepo rls
```

Repository permissions and ownership

If you only intend to use CVSNT on Linux as a client, you can skip this step.

Users and Groups (Linux/Unix)

It is customary to create a user account that will "own" the repository and a group that will have access to the repository that existing users can be assiged to. To create these on Red Hat Linux you can use:

```
# /usr/sbin/groupadd cvsusers
# /usr/sbin/groupadd cvsadmin
# /usr/sbin/useradd -p cvsadminpasswd cvsadmin
# /usr/sbin/usermod -g cvsusers cvsadmin
# /usr/sbin/usermod -a -G cvsadmin cvsadmin
# /usr/sbin/usermod -G cvsusers other-user
```

Best Practice for Users and Groups (Linux/Unix)

Unless you use pserver aliases or the RunAsUser setting, each CVSNT Server process will execute as the user who is logged in. The account designated as the cvsnt administrator should be a member of the same group as the cvsnt users and that the setgid bit should be set on the repository root.

For other users to be able to read and write to the repository you must configure the write permissions on the repository directory to allow read and write access to the files created there, eg:

```
cd /usr/local
chgrp -R cvsusers myrepo
chown -R cvsadmin myrepo
chmod -R u+rw-xs,g+rw-xs,o-rwxs myrepo
find myrepo -type d -exec chmod u+rwx,g+rwxs,o-rwx {} \;
chmod u+rwx,g+rwxs,o-rwx myrepo
```

CVSNT server never updates existing files, but creates new ones and renames them over the top of the originals. Therefore it will not matter if user Fred creates on RCS file and user Mary wants to update it.

Security on "reference copy" (Unix/Linux)

If you are planning to configure your repository to maintain a checked out copy or copies (using *shadow*, *postcommit* or make integration *build_make*) you must configure the write permissions on the reference directory to allow read and write access to the files created there, eg:

```
cd /usr/local
chgrp cvsusers myref
chmod g+rwxs myref
```

Configure the Repository

The repository configuration is stored in text files in the CVSROOT. You edit these by using a text editor (such as vi on Linux) and CVS command line client.

The most common things a new CVSNT administrator will need to do are:

- *Grant an administrator access to the repository using cvs rchacl*
- *Set per repository settings*
  - ⇒ *File extensions that are binary that are not a part of the default list (eg: Microsoft Word files *.doc).*
  - ⇒ *Enable Binary deltas for certain file extensions*
  - ⇒ *Default keyword expansion for certain file extensions*
  - ⇒ *If you wish to use mixed reserved/concurrent editing or exclusively use reserved editing then set the default option for which file types*
  - ⇒ *Add, Modify or Remove Keywords*
- *Set per server setttings*
  - ⇒ *Configure default access control (default – allow, or default – deny)*
  - ⇒ *Configure plugins*

Grant an administrator access to the repository using cvs rchacl

Regardless of whether you will use access control, you should always set one user (the administrator) to have global rights on the repository. Then if the access control is switched on then you will have these rights. Here is a template – customise the *CVSROOT* and *user*:

```
$ cvs -d :pserver:localhost:/myrepo rchacl -u cvsadmin -a all .
```

Set per repository settings

On the server use the following commands:

```
cvs -d:local:/usr/local/myrepo co CVSROOT
cd CVSROOT
```

Use a text editor to edit the file named *cvswrappers*. This file is described in detail in the section below *Administration Files*. A Simple cvswrappers to enable reserved edits in all files and keyword expansion by default in C programs would use this syntax:

```
*.c     -kvx
*.doc   -kbx
*       -kx
```

Use a text editor to edit the file named *config*. This file is described in detail in the section below *Administration Files*. To deny access by default (if you do not set this CVSNT will allow access by default):

```
aclmode = normal
```

When your changes are complete use cvs to commit the changes:

```
cvs commit -m "I have set up my repository"
```

Set per server settings

The directory `/etc` on Linux/Unix contains the per server settings. Copy the `PServer.example` and the `Plugins.example` to `PServer` and `Plugins` respectively. This was also described above in *Optional Server Configuration*.

Disable Virus Scanning on the Repository and CVSNT Temp Directories

Sophos and other Linux kernel based on access Virus scanning should be disabled on the repository directories and also on the temporary directory used by the CVSNT Server. Failing to disable Virus Scanning may result in poor performance and unrecoverable corruption of your version history.

Be sure to read the section *Anti Virus Controls* under *Server Administration* for detailed information on virus scanner support.

Disable or Remove Unused Protocols

If you expect users to connect to your new repository using only sspi, then disable the other protocols (with the exception of the Enum protocol).

For details of how to do this see the *Server Administration* section below under the heading *Disable or Remove Unused Protocols*.

Configure Server for Auto Connection

Smart CVSNT clients such as the CVS Suite Studio can identify that the server exists on the same local network (subnet) as the client and automatically connect the user without the user needing to know the protocol, repository name etc.

For details of how to correctly set this up see the *Server Administration* section below under the heading *Configure Server for Auto Configuration*.

Server Administration

Note that setting up CVS Suite Server and Clients is NOT as simple as installing the software. The sections *CVS Architecture*, *Setting up CVS Suite Server* and the this section *Server Administration* provide the necessary information to install and configure the CVS Suite system. If you do not follow these guides then severe performance problems can occur.

Installing CVS Suite on Unix and Linux

Unix and Linux Server Synchronisation to a Reliable Time Source

For CVS Authentication to operate correctly, and to maximise performance of the CVS System CVS Servers must be synchronised to a reliable time source. This is usually achieved by synchronising the Windows Server to the Primary Domain Controller (PDC) or an external NTP server (eg: `pool.ntp.org`).

Unix Time Synchronisation	
HPUX / Solaris/ Linux	**/usr/sbin/ntpdate -u *servername***
Unix systems must run this command as root and the command can be put in a cron job to run at regular intervals. Alternatively some systems may have ntp client daemons that can achieve this task.	

CVSNT Server

You will need to be familiar with Unix server administration tasks to install CVSNT Server on Unix or Linux, or be assisted by a Unix administrator.

See the guide in the quickstart section above for installing on Unix and Linux.

What is the Deadlock Server (Lockserver)

In all recent versions of CVSNT the *deadlock server* is the primary means of handling file locking. There should be one *deadlock server* per physical server as multiple repositories share it. Once running it should require no maintenance.

The *deadlock server* provides file-level locking for the server, which allows much greater concurrency than previous versions of CVS. It also provides checkout atomicity that ensures that you always get a coherent view of the repository. The previous method of locking using directory locks on the file system is now depreciated and should not be used, as it does not have these advantages.

Setting up the *deadlock server* under Windows is handled by the setup program and happens automatically.

AntiVirus Controls

You can – and it is possible to use CVSNT and some virus scanners on both servers and clients. However March Hare Software does not provide support for problems directly arising from using Anti Virus Software in conjunction with CVSNT. We make this particular exception from our support because:

- Anti Virus software alters the behaviour of processes on the system, and so poorly implemented Anti Virus software can make a well designed and implemented application like CVSNT unstable
- It is important to design your virus scanning strategy together with your CVSNT server implementation
- In high performance environments – no amount of support or application software design can compensate for a poorly implemented Anti Virus system or implementation.

March Hare Software has never had a reported case of Anti Virus Software on a Unix, Linux or Mac OS X client or server affect the CVSNT supported applications. Sophos and some other Linux kernel based on access anti-virus scanning software is becoming more common, and for performance reasosn we strongly recommend that you disable scanning of the CVSNT Temp and repository directories (though scanning the repository CVSROOT directory is OK).

Supported and Unsupported Anti Virus Software

March Hare Software do not recommend any particular Anti Virus Software. To obtain an environment where CVSNT and Anti Virus Software work well together is outlined in the next sections *Configuring Anti Virus Software on Client* and *Configuring Anti Virus Software on Server*.

The Windows Nod32 Anti Virus Software (and in particular the IMON service) has been reported as defective by several customers and March Hare Software do not support or recommend this package in any way.

The Windows BitDefender Anti Malware Softwre (particularly the ‘fast ring’ software which is recommended by BitDefender for non critical systems) has been reported as having occasional false positives against CVS Suite – these are usually quickly rectified by BitDefender themselves.

Most of the laptops that March Hare Software use for demonstrating servers and clients to prospective customers run the Microsoft Security Essentials, however these environments are not high performance. Our Windows and Unix/Linux servers run Sophos endpoint protection. We have previously used McAfee Anti-Virus software successfully on Windows.

If you are running Anti Virus software on a client PC or Server that you require support for from March Hare Software – always advise your support representative. This will NOT preclude you obtaining support. However if the problem is escalated to an engineer and the engineer decides that the Anti Virus may be significantly affecting the behaviour of our application software then March Hare Software reserve the right to ask you to reproduce the support issue on a system without Anti Virus installed. If after using diagnostic procedures on the affected system(s) and on systems without the Anti Virus software March Hare Software are unable to reproduce the problem without the Anti Virus software then your support representative will provide suggestions and support you in the design of an Anti Virus implementation that will not affect the behaviour of the supported application.

An Introduction to Anti Virus Software

Anti Virus software runs mostly on Windows Client and Server PC's. Due to a history of unsecure software running on Windows PC's a range of applications have been developed that exploit this poor software design to allow for additional applications to run without the users explicit consent – these are gererally called viruses or spyware.

Anti Virus software typically links to the Windows operating system in such a way that regardless of how a file (or sometimes a communications port) is opened the Anti Virus software is informed (regardless of the users consent).

The majority of Anti Virus software achieves this by linking in to the Windows Kernel as an IFS driver. These drivers do not appear in the process table – but rather appear as an overhead in every application that uses the file system.

Often problems (such as high memory use, disk activity, performance) that are easily attributed to a program by using the Task Manager is in fact caused by an IFS driver or Anti Virus Package that does not appear as a separate task. This is because "tasks" are user-space programs that can be started and stopped by other user-space programs. By ensuring the Anti Virus software runs at the kernel level the Anti Virus author can be guaranteed that a virus cannot close the anti virus application.

Based on our definition of a virus above - for a virus to be a virus it must execute. Therefore viruses are always held in a file format that can be executed, such as Applications (.exe) Macros (.scr, .xls, .doc, .html) etc.

By default most Anti Virus packages scan memory, and every file on the disk as it is opened or read (a file rename includes an open and read), and some also scan the traffic on the networking ports. Much of this activity is counter productive because the files (or traffic) being monitored can never be executed. To improve the performance anti virus software can usually be configured by an administrator to exclude (not scan) some ports, files or directories.

However since the Anti Virus configuration program runs in the user space and the IFS driver in the Kernel then this configuration change does not do exactly what it is advertised to do. The Anti Virus software cannot ask the kernel to not call it for some files – it is called for all files or not called at all. So the Anti Virus software is still called but depending on its configuration, may perform more or less actions when it is called.

Common Problems with CVSNT and Anti Virus Software

The most common problems that occur with CVSNT and Anti Virus software are due to performance. When a user "updates" thousands of files in a CVSNT project the CVSNT client can download changes and apply them as well as copy (rename) them over the old version very quickly. Some Anti virus software is so poorly designed that when CVSNT attempts to rename a file that is still being scanned it causes CVSNT to be denied access or to crash. This same problem can be easily reproduced with test programs that read, write and rename files quickly.

Configuring Anti Virus Software on the Server

Anti Virus software should NOT scan the CVSNT repository since all these files have a ",v" suffix and are therefore not executable under any operating system, including Windows and Red Hat Linux. Scanning these files is a waste of resources and the Anti Virus software can interfere with CVSNT Servers ability to update the repository files in a timely manner. If there is any .exe or .dll or other "executable" files stored in the repository CVSROOT directory it is acceptable to allow a virus scanner to scan those.

March Hare Software recommends that you specify a temporary directory that is used for ONLY the CVSNT server and set that in the CVSNT Server Control Panel (on windows). That temporary directory should not be scanned – or to be pessimistic – only allow "executable" files in that directory to be scanned.

If the vendor provides different versions of the software for mission critical systems (eg: the BitDefender 'slow ring') then we recommend you use this rather than software which is designed for non-critical systems (like the BitDefender 'fast ring').

Configuring Anti Virus Software on the Client

Since most of the files CVSNT is working with are plain ASCII (or Unicode) text files they do not warrant scanning – particularly the files in the hidden "CVS" directories of a workspace or sandbox.

If possible configure the client Anti Virus to ignore the hidden CVS directories.

Broadly speaking Anti Virus problems occur more frequently on servers than on clients. If an occasional Anti Virus related problem occurs on a client it is usually acceptable by the user to retry the operation and for it to be successful a second time.

Conclusion

If you install commercially supported Anti Virus software on your Windows or Linux/Unix/MacOS based client or server PC's based on industry best practice and the manufacturers recommendation with CVSNT then there is little chance this will cause a functional problem with CVSNT. March Hare Software technical support will assist you in finding a solution in the rare cases when this occurs.

If you require more assistance in determining if CVSNT will operate and can be supported in your environment please do not hesitate to contact pre-sales technical support or technical support. Your Technical Account Manager can also discuss with you options for March Hare Software engineers to visit your site to install or validate the installation of the supported software.

Configuring Network Access

CVSNT Server and CVSNT Clients communicate via the network using one or more *protocols*. These protocols have an impact on security and the performance of your network, the physical server and CVS itself.

In a typical installation people using CVSNT will be authenticated using an operating system token – this means that they do not login to CVSNT – logging into to PC is enough. It is also possible to configure CVSNT Client to require a separate login or require a certificate to authenticate with the server. See the heading Authenticate bypassing Operating System below for more information.

Native usernames versus CVS PASSWD usernames

CVSNT Server on Windows, Unix or Mac OS X can authenticate:

- to native operating system usernames/passwords
- to windows active directory usernames/passwords
  (on unix/mac uses the winbind daemon: Samba plus winbind must be installed and operational)
- to CVSNT's own internal database of usersnames and passwords.

To use CVSNT's own internal database of usernames and passwords you must use the pserver or sserver protocols.

The default way to use CVSNT Server is for the server to authenticate using the operating system usernames and passwords. If this is how your team works then there is no need to perform any additional configuration and no need to use the *cvs passwd* command.

Protocols

CVS client and server may communicate with each other using various supported protocols. When you install CVSNT you are given a choice of installing these:

CVSNT Protocol Description	*Protocol*
Protocol Components Installed	
External command protocol. This includes the EXTNT facility on windows.	:ext:
Password server protocol.	:pserver:
Named pipe protocol.	:ntserver:
Require that clients use authenticated or encrypted packets.	:fork:
GSSAPI for Active Directory.	:gserver:
GSSAPI for MIT Kerberos.	:gserver:
SSPI protocol.	:sspi:
RSH client protocol.	:server:
SSH protocol.	:ssh:
SSL protocol.	:sserver:

March Hare Software recommends that you do not install any protocols that you do not intend to use, or you remove them manually.

The ext protocol and EXTNT on windows

The EXTNT facility on windows allows windows clients that do not have native support for protocols such as SSPI (eg: Eclipse) to use a native protocol via the EXT protocol. See the section *Setting up Eclipse Client on Windows* for more information.

Protocols Ports and Firewalls

CVSNT authentication occurs on port 2401 using TCP (not UDP). In order to use the CVSNT server on one computer from a different computer – network connections to the server on port 2401 must be enabled.

If you are hosting the CVSNT server on a computer with a software firewall such as Red Hat firewalld or iptables then you must ensure that port 2401 is open to allow other computers to use the CVSNT Server.

Protocols Authentication Syntax

Each protocol has its own syntax for authentication. This table describes the key differences.

Protocol Libraries			
Protocol	***Usr***	***Pass***	***Syntax***
Gserver	No	No	`:gserver[;keyword=value...]:host[:port][:]/path`
Sserver	Opt	Opt	`:sserver[;keyword=value...]:[username[:password]@]host[:port][:]/path`
Ext	Opt	No	`:ext[{program}][;keyword=value...]:[user@]host[:]/path`
Pserver	Opt	Opt	`:pserver[;keyword=value...]:[username[:password]@]host[:port][:]/path`
Ssh	Opt	Opt	`:ssh[;keyword=value...]:[username[:password]@]host[:port][:]/path`
Sspi	Opt	Opt	`:sspi[;keyword=value...]:[username[:password]@]host[:port][:]/path`
Server	Opt	Opt	`:server[;keyword=value...]:[username[:password]@]host[:]/path`

Authentication

Authenticate using Operating System

A remote CVSNT Server repository is usually configured so that the authentication requirements are the same as for the operating system using a 'normal' interactive logon (including two factor authentication such as RSA SecurID). This is the usual and default authentication type.

Authenticate bypassing Operating System

Alternatively the administrator can choose to use the CVSNT security system to provide CVSNT accounts without having to set up accounts valid on the underlying operating system and to bypass operating system security. This is not the default and is uncommon, but is supported.

Config

The *config* administration file has a setting *SystemAuth* which defines how logins are authenticated and it also effects how administrator users are defined. More information on how to edit this file is contained in the section *Administrative Files.*

If *SystemAuth = Yes* the user is authenticated by the host operating system and is considered to be an administrator if they are listed in the CVSROOT/admin file or if they are in the 'Administrators' group (NT) or 'cvsadmin' group (Unix).

If *SystemAuth = No* the user is authenticated only by the password in CVSROOT/passwd (see the notes on the sspi protocol below in the section *Limiting user access with sspi* below). Administrative users are defined by the CVSROOT/admin file.

To deny access by default: set *aclmode* = *normal* (if you do not set this CVSNT will allow access by default). Before setting this you must ensure that you have already granted the administrator access using cvs rchacl command, eg:

```
$ cvs -d :pserver:localhost:/cvsrepos rchacl -u cvsadmin -a all .
```

Admin

The *admin* administrative file contains a list of usernames who are designated repository administrators, one per line. This file should *not* be put under CVS control, as that would be a security risk.

Repository administrators are automatically made members of the group 'admin'.

SSH

SSH authentication allows Unix servers to authenticate the logged in user using the standard secure SSH protocol. See the above section *SSH versus CVS Network Protocols (SSPI, Sserver, etc.)* for information on how this is different to the CVSNT in-built protocols.

After your repository is created (cvs init) and published (cvs init -a or by editing /etc/cvsnt/PServer) test that you can connect to your server using ssh from the server:

```
ssh user@localhost
```

From the windows client connect to the server using:

```
cvs -d :ssh:user@host:/cvsrepo ls
```

For a Unix client you will need to set up an :ext: connection.

SSPI

SSPI authentication allows Windows to authenticate the logged in user. The authentication will be at least as secure as the authentication used to log you in the Windows workstation and connect to the network.

SSPI works over TCP/IP so it can more easily traverse firewalls. The SSPI protocol does not need a login, the login you did when you started your workstation is used with this protocol.

NTLM vs Kerberos

Depending on the Domain Controllers in your Windows network the authentication will be handled by NTLMv1, NTLMv2 or Kerberos. Windows domains based on Active Directory will generally use Kerberos (which is the most secure) unless specifically configured not to.

To force CVSNT to connect using a specific authentication protocol (*kerberos* or *ntlm*) when connected using SSPI use the *force* keyword. Eg:

:sspi;force=kerberos:*hostname*:/*repository*

CVS only users (no system user with that username)

You may have external users who need to access the CVSNT server who do not have a system login. You can define those users in the CVSROOT/passwd file (or using the *cvs passwd -a* 'adduser' command) and specifying a generic user (which is a valid system user) for the process to with the credentials of.

Example the CVSROOT/passwd file:

```
melissa:tGX1fS8sun6rY:pubcvs
qproj:XR4EZcEs0szik:pubcvs
```

The user *melissa* will be authenticated by CVSNT server only, and any processes initiated by that user will run on the server with the credentials of pubcvs. The CVSNT Access Control (ACL lists etc) however will all be restricted based on the username *melissa* not the user *pubcvs*.

Limiting user access with sspi

When used 'normally': SSPI will accept connections from all system users that authenticate against the system (local or domain). Often this is not really what is wanted; instead we want to use the same mechanism as is used with the PSERVER protocol. Here the *passwd* file limits the logins accepted by CVSNT to those mentioned in the file.

If you set the parameter *SystemAuth = No* in the config administration file then only users listed in passwd will be authorized to connect to the CVS Suite Server. You can add a user to the passwd file with the following command run as a CVS Suite Administrator:

cvs passwd -a -d DOMAIN newuser

If the newuser is a member of a different domain than the default domain for the CVS Suite Server, then use this syntax:

cvs passwd -a -d DOMAIN DOMAIN\newuser

For example, if your CVS Suite Server is located on a server in the ALPHA domain but your user account is in the BETA domain, you would use this command to add the user to the passwd database:

cvs passwd -a -D BETA BETA\username

Note: the username and the domain name are case sensitive. On Windows, Domain names are typically all UPPERCASE, but usernames on both Windows and Unix/Linux are however the administrator entered them. You can see the exact case information that CVS Suite is receiving if you type the following command:

cvs -d :sspi:host:/repo login

The username will be shown in the line above the password prompt, press CTRL-C to cancel the login request.

You can specify either an authenticating domain or a password in the passwd file (press enter twice to tell CVSNT that no password is used). Failure to include one or the other will allow anyone to connect as that user without a password and is NOT recommended.

Note: Be sure to add an administration user to the passwd file before setting SystemAuth = No, or otherwise you will be locked out.

If you are not going to be using other protocols like pserver or sserver, we recommend that you disable them using the CVS Suite Server control panel.

Requiring Encrypted and/or Authenticated Network Traffic

CVS client and server may communicate with each other using various supported protocols, some of these are encrypted and/or authenticated, and with some of the protocols there is a choice.

What is Authenticated Network Traffic

Authenticated network traffic contains additional information to guarantee that each individual packet was sent by the specified client and has not been impersonated in some way.

What is Encrypted Network Traffic

Encrypted network traffic has been scrambled so that its contents are meaningless without the "keys" known only to the server. Encrypted network traffic is also authenticated (as defined above). The level and type of encryption used is dependent on the protocol. CVSNT does not use one type of encryption for all protocols. Many of the protocols will have options for stronger or weaker encryption, eg: SSPI will allow NTLM (weaker) or Kerberos (Stronger).

NOTE: It is not possible to use an encrypted traffic with an insecure protocol such as pserver does and this would not result in secure transmission. Defining a secure environment is not as simple as ticking one checkbox.

How do you specify what level of encryption and authentication to use?

The authentication and encryption choices at the server are:

CVSNT Control Panel Description	***Setting***
Encryption in the Server Settings Tab	
Allow clients to connect without encryption.	Optional (Default)
Request that clients use authenticated packets if available.	Request Authentication
Request that clients use encrypted packets if available.	Request Encryption
Require that clients use authenticated or encrypted packets.	Require Authentication
Require that clients use encrypted packets.	Require Encryption

Unix/Linux

On Red Hat Linux and other Unix/Linux systems you can set the EncryptionLevel and CompressionLevel using the `/etc/cvsnt/PServer` configuration file:

```
#
# Encryption  0=Don't force encryption,
#           1=Request authentication,
#           2=Request encryption,
#           3=Require authentication,
#           4=Require encryption
#
#EncryptionLevel=0

#
# Compression 0=Don't force compression
#           1=Request compression
#           2=Require compression
#
#CompressionLevel=0
```

What protocols can be encrypted?

CVSNT does not use one encryption library for all protocols, instead each protocol defines its own encryption. SSPI will use NTLM or Kerberos encryption (the default is Negotiate), GSERVER will use Kerberos encryption.

If the server option 'require encryption' is enabled then SSPI is NOT allowed as a connection protocol. This is is a deliberate decision since the 'default' negotiated protocol on standalone windows systems uses NTLM unless the user specifically alters the CVSNT SSPI protocol plugin to only use Kerberos, or joins the system to a Domain/Active Directory.

Requiring Compression

CVS client and server may optionally compress the network communication before it is encrypted. This may improve CVS performance on a congested network or a network where bandwidth is limited (such as a WAN or Public Internet). You may use the CVSNT Control Panel to set CVSNT Server to require a particular level of encryption. The encryption choices are:

CVSNT Control Panel Description	*Setting*
Compression in the Advanced Tab	
Allow clients to use compressed packets.	Optional (Default)
Request that clients use compression when available.	Request Compression
Require that clients use compression.	Require Compression

Disable or Remove Unused Protocols

If you expect users to connect to your new repository using only sspi, then disable the other protocols (with the exception of the Enum protocol).

Disable protocols on Mac OS X / Unix / Linux

On Unix and Linux edit the `/etc/cvsnt/Plugins` configuration file to disable any protocols that should not be used to connect to this server. Example:

```
EnumProtocol=1          # Remote enumeration (info -r)
GserverProtocol=0       # GSSAPI
PserverProtocol=1       # Original pserver
SserverProtocol=0       # SSL encrypted pserver
SspiProtocol=0          # SSPI (via Winbind, if configured)
```

Configure Server for Auto Configuration

Smart CVSNT client such as the CVS Suite Studio can identify that the server exists on the same local network (subnet) as the client and automatically connect the user without the user needing to know the protocol, repository name etc.

This will work for 99% of windows CVSNT server "out of the box", however unix, linux and Mac OS X servers will require additional steps. These steps are also good for Windows CVSNT server administrators to consider.

Steps to allow server to allow advanced clients to automatically connect:

The following should be configured to allow advanced clients such as CVS Suite Studio to connect automatically to the new CVSNT Server:

- Protocols that have not been configured have been disabled, eg: sserver. See above section *Disable or Remove Unused Protocols* for more information.
- Set an anonymous username and protocol that is most often required for Unix servers to allow non-unix users to browse the module list. The username specified can have the access restricted using an ACL. (See /etc/cvsnt/PServer)
- Set the repository to Publish and the choose one repository as the Default. (See /etc/cvsnt/PServer). Windows users can configure this in the CVSNT Server control panel by editing the repository in the Repository Configuration tab.
- Enable Zeroconf publication in the Advanced tab of the CVSNT Server control panel or in the /etc/cvsnt/PServer configuration file.

Requiring Secure SSPI connections

Requiring secure SSPI connections requires these settings on the server:

- Encryption: Optional or Request Clients use Encryption
- Protocol plugins: disable (or remove) all protocols except SSPI and ENUM
- Protocol plugin SSPI: enable protocol, disable NTLM & Negotiate, enable Kerberos

The client can use a connection string CVSROOT like this to force the encryption:

```
:sspi;force=Kerberos:host:/repo
```

Using Kerberos requires that the CVSNT server is joined to a domain, and when you authenticate with a domain user (not a local user) then the domain policy will enforce Kerberos authentication and encryption[6]. If the server is not on a domain, the client connection will fail when force=Kerberos.

It is not possible to have a securely encrypted SSPI connection to CVSNT Server when the server computer is not joined to a domain because then NTLM authentication is used – this is a limitation of windows. Also Linux clients are unable to use Kerberos encryption with SSPI connectsions – this is a limitation of Linux. CVSNT Linux clients can use GSERVER to Windows CVSNT Servers. GSERVER is a secure Kerberos protocol.

Ensuring SSPI connections are secure (single sign on):

The weakest point of any encrypted authentication is the password. The encryption is entirely dependent on the protocol, which for SSPI is dependant on the authentication – evidenced by the fact that the same CVSNT protocol 'SSPI' is 'secure' if it uses Kerberos authentication and encryption but insecure if it uses NTLM. So if the password is stored poorly in the client, the value of the encryption is negligible. If the user enters their Windows domain password into CVSNT, WinCVS, TortoiseCVS or CVS Suite Studio, then it will be stored with trivial encryption (or plain text) in the Windows Registry.

Therefore it is important that the users use 'single sign on' and not enter passwords manually. Single Sign on uses the authenticated token that Windows holds from their desktop login is passed to the server for authentication. To enable 'single sign on' do NOT specify a username or password with SSPI and do not use 'cvs login'.

Here is an example of a SSPI secure client connection:

```
cvs :sspi;force=Kerberos:host:/repo checkout mymodule
```

Here is an example of an insecure SSPI secure client connection:

```
cvs :sspi:user@host:/repo login
password: *****
cvs :sspi:user@host:/repo checkout mymodule
```

Requiring Secure SSH connections

Requiring secure SSH connections requires these settings on the server:

- Encryption: Require Clients use Encryption
- Enable your SSH server on the CVSNT server
- Protocol plugins: disable (or remove) all protocols except ENUM

SSH Authentication:

There are three ways customers typically authenticate SSH connections via CVSNT:.

[6] See Microsoft NTLM documentation:
https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-ntlm

- password
- keys
- GSSAPI

Ensuring SSH connections are secure (private and public keys):

The weakest point of any encrypted authentication is the password, or frequently in the case of SSH logins: the private keys. If the password or keys are stored poorly in the client, the value of the encryption is negligible. If the user enters their SSH password into CVSNT, WinCVS, TortoiseCVS or CVS Suite Studio, then it will be stored with trivial encryption (or plain text) in the Windows Registry. If the private key is stored in a file and passed to the SSH client then it is also insecure.

Customers typically use either of these methods to secure their SSH authentication:

- Storing the keys in the Active Directory
- Using a secure key agent, eg: putty pageant and loading the keys from a secure place, eg: an encrypted key file, or secure key storage device

Here is an example of an insecure SSH secure client connection:

```
cvs :ssh;privatekey=d:\me.ppk:user@host:/repo checkout mymodule
```

Integration dependencies and requirements

Whilst the CVSNT Server integrations do not have any direct dependencies, you may need to install dependent software for the integrated system, eg: the Bugzilla integration requires Bugzilla to be installed – and Bugzilla requires Perl and MySQL. Detailed instructions are available in the section *Configuring MySQL for CVSNT*.

Windows XP/Vista and Windows 7 Professional can generally be used as a server provided that simple networking is switched off and the number of users will not exceed the pre-defined limits of that edition of windows. March Hare Software strongly recommends running Windows Server 2003 or 2008 rather than Windows XP/Vista or Windows 7.

Integration with Make (Build Post Commt Trigger)

When a user commits a file to the CVSNT server it is possible to automatically begin a build process. The process begins after the command has completed on the client.

The make_trigger.dll is included in the CVS Suite Server install available from the customer area of the march-hare.com web site.

Default Behaviour

The build post commit trigger plugin will be used if enabled and has the following behaviour:

- On completion of a commit the file `CVSROOT\build_make.bat` (`build_make.sh` on unix) will be executed. The *build_make* will be checkout out and executed from a temporary directory – it does not have to appear in the administrative file list.

- The build file receives the 1st and 2nd level directory (module name and directory name) followed by the branch name (if available), followed by the bug number (if available). Eg build_make.bat:

  ```
  @echo Build: 0=%0,1=%1,2=%2,3=%3,4=%4 >> c:\temp\build_make.log
  ```

  Returns this result:

  ```
  Build: 0=CVSROOT\build_make.bat,1=TestModule,2=,3=,4=
  Build: 0=CVSROOT\build_make.bat,1=TestModule,2=,3=TestBranch,4=
  Build: 0=CVSROOT\build_make.bat,1=TestModule,2=afolder,3=,4=
  Build:
  0=CVSROOT\build_make.bat,1=TestModule,2=afolder,3=TestBranch,4=1234
  ```

Note 1:
The "branch name" passed is a simple mechanism and is designed primarily for a sandbox which is either on the Trunk or on a Branch. Situations where a single directory contains files from several branches or with different tags are not supported by the make integration DLL.

Note 2:
The "2nd level directory" passed is a simple mechanism and is designed primarily for when a sandbox begins at level 2: ie when checked out with:

```
cvs -d :sspi:localhost:/myrepo co -r TestbRANCH TestModule/afolder
```

which is either on the Trunk or on a Branch. Situations where a single directory contains files from several branches or with different tags are not supported by the make integration DLL.

Configuring on Unix

The build integration is enabled in the file `/etc/cvsnt/Plugins` .

Audit (to database)

Using the Audit integration it is possible to generate detailed audits of CVSNT repository activity including activity based reporting for a time period such as:

- most frequently changed file
- user with most changes
- most modified file by number of lines
- user with largest changes by number of lines

Audits client activity, not server administration

The purpose of the CVS Suite Server audit function is to independently audit changes made to the repository using the CVS Suite Client software or 3rd party client software which connects to the CVS Suite Server using the published API.

The CVS Suite Server audit function does not:

- audit changes to server configuration files
- audit intrusions into your server
- audit changes to the system configuration files (or windows registry)
- track changes made by an administrator on the server

Your operating system may have tools to perform that type of auditing, and there are also 3rd party products available for performing those auditing functions. Whether you require that type of auditing should be considered carefully. March Hare Software do not make any specific recommendations regarding server system auditing.

On Red Hat Enterprise Linux v5, the **auditd daemon** is supplied by the vendor which may be of some use for system level auditing.

On Microsoft Windows you may be able to configure the Group Policy to automatically audit changes to the registry.

Default Behaviour

The trigger DLL will be used if the triggers administrative file is configured to activate it. The trigger is pre programmed with the following behaviour:

- On all user interaction log the details, options include:
  - ⇒ Sessions (this should always be enabled if any other option is enabled since all tables are linked using the session id)
  - ⇒ Commits (including number of lines changed)
  - ⇒ Store checking differences
  - ⇒ Tags
  - ⇒ History

Limitations

The audit subsystem has the following limitations:

- The SessionLog Hostname field will always contain the name of the server when the client connects via the ssh protocol
- The client command *login* is not audited and does not appear in the SessionLog
- The Date and Time fields in the Audit database always are expressed in GMT/UTC.

Creating the Database

CVSNT Audit Integration supports MySQL Server 4.1, SQLlite Version 3.2 and Oracle 9i and 10g as well as MS SQL on Windows. You will need to install the database server either on the same machine as the CVSNT server or another machine, create a database, username and create the database from the supplied SQL scripts (on windows there is a button on the Audit plugin control panel for creating the database).

Creating the Database using MYSQL

After installing MySQL ensure that you have the services running and the database configured (eg: /etc/my.cnf), then create a database for the audit tables.

```
mysql --user=root --password=xxx mysql

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.1.7-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database audit;
Query OK, 1 row affected (0.00 sec)

mysql> grant all privileges on audit.* to 'audit'@'localhost' identified by
'';
Query OK, 0 rows affected (0.00 sec)

mysql> grant all privileges on audit.* to 'audit' identified by '';
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> quit
Bye
```

Configuring the Database using MySQL

After the database is installed and created, create the database from the supplied SQL scripts.

Example for MySQL:

```
mysql -u user -D database < create-script.sql
```

If you want to set a password for the audit account then you will also need to use the "set password" command to ensure the password is encoded in the 3.23 format. If you are using MySQL 3.23 server:

```
set password for audit@'%' = password('password');
```

If you are using MySQL 4.1 or 5.x server (or later):

```
set password for audit@'%' = old_password('password');
```

If you do not use this setting then you will get the following error message:

```
Client does not support authentication protocol requested by
server; consider upgrading MySQL client
```

Configuring on Unix

The Audit integration is configured in the file `/etc/cvsnt/PServer` and is enabled using the file `/etc/cvsnt/Plugins` .

Audit Data Model

*Table **SessionLog***

Id (automatic primary key)
Command char(32)
StartTime Timestamp,
EndTime Timestamp,
Hostname char(256)
Username char(256)
SessionId char(32)
VirtRepos char(256)
PhysRepos char(256)
Client char(64)
FinalReturnCode integer

*Table **AuditReport***

ReportName char(32)
ReportDesc char(256)
ReportResults Integer
ReportSQL text

*Table **CommitLog***

Id (automatic primary key)
SessionId Integer
Directory char(256)
Message text
Type char(1)
Filename char(255)
Author char(255)
Tag char(64)
BugId char(64)
OldRev char(64)
NewRev char(64)
Merge1 char(64)
Merge2 char(64)
Added Integer
Removed Integer
Diff text

*Table **HistoryLog***

Id (automatic primary key)
SessionId Integer
Type char(1)
WorkDir char(256)
Revs char(64)
Name char(256)
BugId char(64)
Message text

*Table **TagLog***

Id (automatic primary key)
SessionId Integer
Directory char(256)
Filename char(256)
Tag char(64)
Revision char(64)
Message text
Action char(32)
Type char(1)

Audit Database Queries and Reports

Custom Reports

March Hare Software provides on site consulting services to customers with CVS Suite Annual Software Maintenance and Support level Silver, Gold or Platinum (or per-incident for 50+ licenses) at competitive rates. Using these consulting services we can deliver a variety of reports for your Audit requirements tailored to your Database, platform and CVS Suite Server versions. We also offer on site CM Design and CVSNT Administration training.

Self Support and Bronze level usually do not qualify for on site services because there is often a need for follow up questions & answers which are best done over the phone, and Self Support and Bronze support customers do not have telephone access to our support team.

If you have a requirement for any consulting services please contact your Technical Account Manager they will be able to assist you with the best quotes.

Sample Reports

The following sections contain some sample queries that are designed for use with MySQL and CVS Suite 2.5.03. These queries can be used directly in an SQL client, or with a report writer such as Microsoft Query Tool or Crystal Reports.

Users by lines of code changed

A basic query to extract the users by lines of code changed compatible with MySQL is:

```
select sum(added)+sum(removed),Username
from commitlog, sessionlog
where commitlog.sessionid = sessionlog.id
group by Username
order by 1 desc;
```

Find changes by text and date

To find all the instnaces of "<input type='text'" that have been modified in the last 7 days it is necessary to have the *store checkin differences* option selected in the Audit plugin which this is not enabled by default. Depending on how CVS Suite Server is used, enabling the *store checkin differences* option can result in the CommitLog table requiring a large database.

```
select Diff,Username
from commitlog, sessionlog
where commitlog.sessionid = sessionlog.id
and DATE_SUB(CURDATE(),INTERVAL 7 DAY) <= Date
and Diff like "%<input type='text'%"
order by 2 desc;
```

Alternatively if the Audit database has not been configured to store the checkin differences then you can use this query to return the files modified in the last 7 days and the revisions. This list of files and revisions can then be used with the *cvs rdiff* command to extract the checkin differences and the result parsed.

```
select CONCAT('cvs rdiff -r ', OldRev, ' -r ', NewRev, ' "', Directory, '
/', Filename, '"')
from commitlog, sessionlog
where commitlog.sessionid = sessionlog.id
and DATE_SUB(CURDATE(),INTERVAL 7 DAY) <= Date
order by 3 desc;
```

Integration with Email

When a user commits, tags or begins work on a file (with cvs edit) it is possible to automatically send an e-mail to people who are “watching” that file.

More complex Email integration is possible using CVSMailer a separate (free) product.

Default Behaviour

When a user commits, tags or begins work on a file (with cvs edit) all users . It allows you to put any contents in the emails, but the output format is fairly simple - it is no substitute for a purpose designed notification program.

Email sending is disabled by default. To configure it for use you must do the following.

Configure the commit support files

commit_email, tag_email and notify_email

The commit support files *commit_email*, *tag_email* and *notify_email* contain the names of the template files to use for commit, tag and edit respectively. Each line in these files is a regular expression followed by a filename. The filename is always relative to the CVSROOT directory and may not be an absolute path for security reasons.

The first matching line for each directory committed is used. If there is no match the DEFAULT line is used.

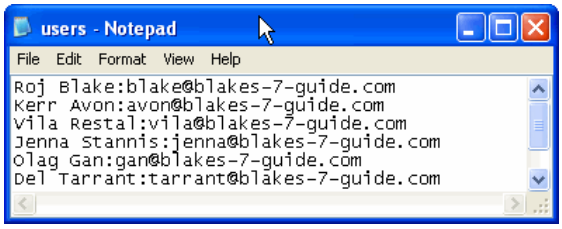
template and checkoutlist

The name of the template file should also be listed in the *checkoutlist* administration file so that it is available for the script to use.

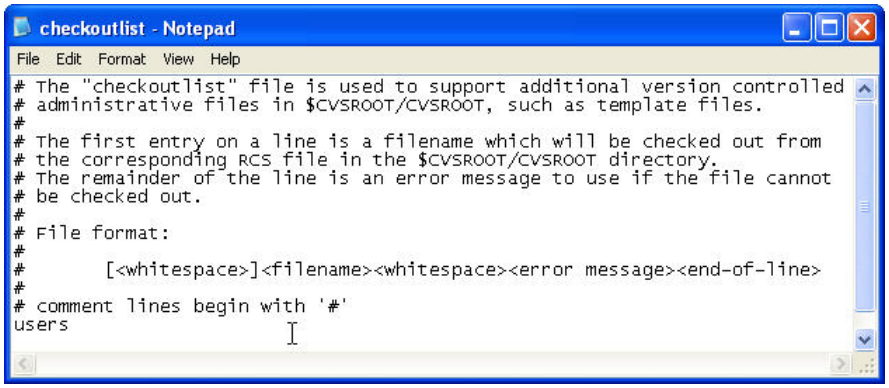
users and checkoutlist

The administration file *users* is used to lookup the username -> email mapping. This file is a list of colon separated username/email pairs. If this file does not exist or the username is not listed the default domain name set in the global configuration is used.

Create a file named users (no file name extension) that will map between users login names (ie: Active Directory names) and their e-mail addresses:



Add the users file to the checkoutlist file:



Write the template

The template file is a text file listing the exact text of the email to send including headers. The To:, From:, Cc: and Bcc: lines are used by the sending software to determine the addresses to use. An example commit template is:

```
From: [email]
To: cvsnt_users@mycompany.com

Subject: Commit to [module]

CVSROOT:     [repository]
Module name: [module]
Changes by:  [email]           [date]
On host:     [hostname]

[begin_directory]
Directory: [directory]

[begin_file]
[change_type] [filename]   [tag]   [old_revision] -> [new_revision]
[bugid]
[end_file]
[end_directory]

Log message:
[message]
```

A number of replacements are done on the file to format it for final sending. This differs for each file, and is listed below.

Configure the server

There are two ways that CVSNT can send email. The simplest is to set the SMTP Server and default domain in the global configuration (CVSNT Server Control Panel in Windows, `/etc/cvsnt/PServer` in Unix) and let the internal SMTP client send the emails.

This will not work in the case where authentication is required or the server is not capable of SMTP. In these cases you should set the Email Command. This command should take a list of 'to' addresses as parameters, and a raw email as its standard input.

A suitable configuration for Unix systems is

```
/usr/sbin/sendmail -i
```

Similar programs exist for Windows.

A quick practical introduction to command line CVSNT

This section is intended to give the administrators a quick overview as to how CVS works on a day to day basis. Administrators should become familiar with how to use the command line version of CVSNT. Graphical clients such as CVS Suite Studio, the Visual Studio .NET integration, TortoiseCVS and WinCVS do run these identical commands – and you can see them displayed in their output windows.

The first two sections give an introduction to the syntax and environment of the command line CVSNT, then each section describes a single command. For more detailed information on each command refer to the reference in the appendix.

Syntax

Each CVS command consists of:

- Executable CVS
- Global Options (optional)
- CVS Command (verb - required)
- Command Options (optional)
- Command Parameters (optional)

This is the generic layout of a CVSNT command:

```
cvs global-options cvscommand options parameters
```

CVSROOT

The CVSROOT is needed by every CVSNT command except *info*. CVSROOT defines:

- Authentication protocol
- Protocol Options
- Repository Server
- Repository Name
- Username / Credentials

The CVSROOT can be defined on the command line between the name of the executable CVS and the cvs command (verb) or it can be set in an environment variable. All of the examples in this chapter specify the CVSROOT on the command line, eg:

```
cvs -d :sspi:localhost:/myrepo cvscommand
```

Logging in

If you are using the SSPI protocol and you are "logged in" to your computer on the same domain and user account as you wish to use with CVSNT then there is no need to login. For all other protocols (or if using SSPI to log into a different domain and account) then you need to use the CVS LOGIN command with a CVSROOT, eg: .

```
cvs -d :pserver:myname@server.com:/myrepo login
```

Info

The CVS command *info* is used to browse CVSNT Servers on the network. Here are two commands, the first obtains a list of all CVSNT servers responding on the network and the second obtains detailed information about one of those servers. The Info command does not require a CVSROOT.

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example
C:\>cvs info -b
2ksrvr01.march-hare.local     CVSNT 2.5.01 (Travis) Build 1927
                              TI800 (3)
cluster.march-hare.local      W2K3CLSTR
                              acer64lap
w2k3clstr2.march-hare.local   w2k3clstr2

C:\>cvs info -r acer64lap
erver: acer64lap
Version: Concurrent Versions System (CVSNT) 2.5.02 (Servalan) Build 2064

Protocols:
  gserver
  pserver
  sspi

Repositories:
  /eclrepo                    Repository for Eclipse junk
  /manchester                 copy of manchester march hare repository
  /testrepo                   this is a test

Default repository: /testrepo

Default login: :sspi:acer64lap:/testrepo

C:\>
```

Import

The CVS command *import* is used to import a new module into CVS. It can often be more sensible to create the new module and add the files.

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example
C:\example\>cvs -d :sspi:localhost/myrep import -n -m "initial import" modulename
 cvs server: Importing /myrep/adminexample
 cvs server: Importing /myrep/adminexample/hello
 N ./adminexample/hello/hello.c
 cvs server: Importing /myrep/adminexample/include
 N ./adminexample/include/hello.h

 No conflicts created by this import

C:\example\>
```

Checkout

The *chechout* command is used to obtain a copy of the repository into a local sandpit.

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example
C:\example\>cvs -d :sspi:localhost/myrep  -r co -t -d myexample adminexample
 cvs server: Updating myexample
 cvs server: Updating myexample/hello
 U myexample/hello/hello.c
 cvs server: Updating myexample/include
 U myexample/include/hello.h

C:\example\>
```

Edit

The *edit* command is used to reserve (or lock) a file. It will reserve the file automatically if the file was imported with the –kc or -kx options. Without the *-c* switch it will not reserve the file, but will notify any users who are also working on the same file (if a suitable *Notify* script is in use).

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example\myexample\hello
C:\example\myexample\hello\>cvs edit hello.c

C:\>
```

Diff

The *diff* command is used to check the differences between the local file (in the sandbox) and the latest version in the repository.

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example\myexample\hello
C:\example\myexample\hello\>cvs diff hello.c
ndex: hello.c
 ===================================================================
 RCS file: /myrep/adminexample/hello/hello.c,v
 retrieving revision 1.1.1.1
 diff -r1.1.1.1 hello.c
 1a2
 > #include "../include/stuff.h"

C:\example\>
```

Add

The *add* command is used to add a file to the repository.

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example\myexample\include
C:\example\myexample\include\>cvs add stuff.h
 cvs server: scheduling file `stuff.h' for addition
 cvs server: use 'cvs commit' to add this file permanently

C:\example\myexample\include\>
```

Update

The *update* command checks that the versions of the files that were originally checked out are still the latest ones in the repository. If some other developer has committed changes to the repository since you checked out your copy then your changes and theirs will be merged.

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example\myexample
C:\example\myexample>cvs update -t
 cvs server: Updating .
 cvs server: Updating hello
 M hello/hello.c
 cvs server: Updating include

C:\example\myexample>
```

After the *update* is complete you should check that your changes still compile and execute as anticipated.

Commit

The *commit* command is used to store changes into the central repository. You should always use an *update* command before a commit command.

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example\myexample
C:\example\myexample\>cvs -r commit -c -B 1234 -m "Updated as per customer request"
 cvs commit: Examining .
 cvs commit: Examining hello
 cvs commit: Examining include
 Checking in hello/hello.c;
 /myrep/adminexample/hello/hello.c,v  <--  hello.c
 new revision: 1.2; previous revision: 1.1
 done
 RCS file: /myrep/adminexample/include/stuff.h,v
 done
 Checking in include/stuff.h;

C:\example\myexample\>
```

Rollback / Undo a Commit

Every commit transaction in a CVS repository has a unique commit ID. This ID can then be used to re-action the exact same change set at a later date. For example, to rollback a commit that you just performed (replace *filename or modulename*) with the name of a file you commtted (hello.c in the above example):

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example\myexample
C:\example\myexample\> cvs status filename or modulename
.
.
   Commit identifier:    a38504385fe483
.
.

C:\example\myexample\> cvs update -e -j @commitid -j "@<commitid"
```

Enable Unreserved Mode

Each file can have unreserved, reserved or cop-operative mode enabled – and this can vary over time and branches. Here is an example of switching to unreserved mode:

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example\myexample
C:\example\myexample\> cvs edit
C:\example\myexample\> cvs update -k-c
C:\example\myexample\> cvs ci -f -R -m "change from reserved to unreserved"
.
.
.

C:\example\myexample\> cd ..
C:\example\ > rd /s /q myexample
```

Rename

The *rename* command is used to rename a file in a sandpit where the source and destination are in the same directory. A rename should only be performed on files that have no changes made to them (i.e.: commit all changes first) and should only be done in the same directory (i.e.: no "moves" until EVS 3.1.01 server is released).

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example\myexample\hello
C:\example\myexample\hello\>cvs update hello.c
C:\example\myexample\hello\>cvs edit hello.c
C:\example\myexample\hello\>cvs rename hello.c hello.cpp
C:\example\myexample\hello\>cd ..
C:\example\myexample>cvs commit -m "renamed hello.c to hello.ccp"
 cvs commit: Examining .
 cvs commit: Examining hello
 cvs commit: Examining include
 /myrepo/testc2/.directory_history,v  <--  directory update
 new revision: 1.2; previous revision: 1.1
 done

C:\example\myexample>
```

Release

The *release* command is used to delete the files from a sandpit.

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd example
C:\example\>cvs -d :sspi:localhost/myrep release -d myexample
 Are you sure you want to release (and delete) directory `myexample': y

C:\example>
```

Troubleshooting

Some basic troubleshooting information is also listed in the appendix *Frequently Asked Questions*.

Performance

Note that setting up CVS Suite Server and Clients is NOT as simple as installing the software. The sections *CVS Architecture*, *Setting up CVS Suite Server* and *Server Administration* provide the necessary information to install and configure the CVS Suite system. If you do not follow these guides then severe performance problems can occur.

Performance problems are most often caused by connectivity issues. CVS Suite Client is very robust and will often appear to *work* but be very slow if the connectivity and authentication are not correctly configured.

For the reasons listed above, if you experience performance problems the best action is to contact March Hare Software Support and an engineer can work through traces and your settings to determine where the performance problem is.

The following sections give a very rough guide as to the main causes of performance issues:

- Anti Virus software
- RAID drivers, disks and Network devices
- Reverse DNS
- Authentication delays (particularly with Active Directory, Domain and LDAP authentication)
- Network Latency
- Memory and CPU constraints

If you have followed the installation procedure as described in this manual then performance problems usually do not occur, therefore the diagnostic process for performance problems can be complex – so please contact March Hare Software technical support and be prepared to answer many questions. Diagnosing the cause of a performance problem may take several days, however they are always solvable.

Connectivity

If you are using *pserver* or *SSPI* protocols the following section may help you debug what is going wrong.

Pserver and SSPI protocols

Errors along the lines of "connection refused" typically indicate that the cvsservice is not listening for connections on port 2401 whereas errors like *connection reset by peer* or *recv() from server: EOF* typically indicate that the cvsservice is listening for connections but is unable to start cvsnt.exe.

The error *unrecognized auth res*ponse is caused by a bad command line or invalid option used to start the CVS Server on Unix or Linux systems (eg: forgetting to put the *authserver* command at the end of the line). Another less common problem is an invisible control character that your editor "helpfully" added without you noticing.

Using Telnet to Debug

One good debugging tool is to *telnet servername 2401*. After connecting, send any text (for example "foo" followed by return). If CVS Server is working correctly, it will respond with

cvs [authserver aborted]: bad auth protocol start: foo

If it fails to work at all, then make sure the CVS Server *cvsservice* is working right.

Simultaneously attempting to run CVS

If several developers try to run CVSNT at the same time, one may get the following message:

[11:43:23] waiting for bach's lock in /usr/local/cvsroot/foo

CVS will try again every second, and either continue with the operation or print the message again, if it still needs to wait.

Checkouts are atomic

If the user Fred commits some changes on one workstation, and simultaneously another user Mary performs an update at another workstation, Mary will either get all of Fred's changes, or none of them.

By default atomic checkouts occur across an entire checkout, (provided the *deadlock server* is running), but commits are only atomic at the file level. For example, given the files

```
a/one.c
a/two.c
b/three.c
b/four.c
```

If someone runs:

```
cvs ci a/two.c b/three.c
```

And someone else runs CVS *update* at the same time, the person running *update* will either get all of the changes, or none of them. If however, the person running *cvs commit* suffers a power failure in the middle of the commit, it is possible that only one of the files will be committed / checked in.

Eclipse

If you have a problem with the interoperability try to get a client/server trace from eclipse before reporting bugs as it is difficult for the developers without diagnose problems without this detail.

Create a Client/Server Trace

There are tracing facilities in Eclipse that will allow you to see what messages are being communicated between the CVS client and server. This section describes how.

Pre-requisites

You will need Eclipse and a Java Run-time installed on the client. On Windows Eclipse does not usually require a separate Java run time, but to enable debugging you will need this as well.

Step 1

Create a file named `.options` in the directory you start Eclipse from (in most cases this is the directory that contains the executable but it may differ in some cases: for instance, if you use a shortcut in windows and specify a different starting directory) that contains the following 2 lines that enable CVS debugging.

```
org.eclipse.team.cvs.core/debug=true
org.eclipse.team.cvs.core/cvsprotocol=true
```

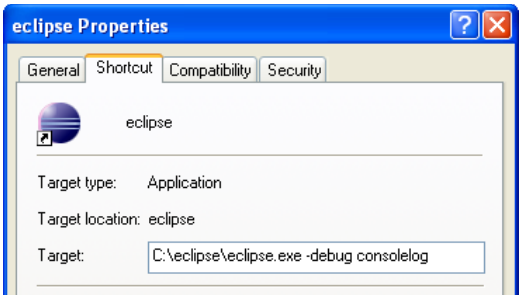
Step 2

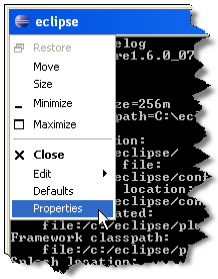
Start Eclipse with the following parameters tailored to you local setup (The below example is for a typical windows setup). The important aspects are the use of java.exe instead of javaw.exe and the inclusion of the -debug and -consolelog options. These will cause the debug console to be visible and for debugging output to appear in the console.

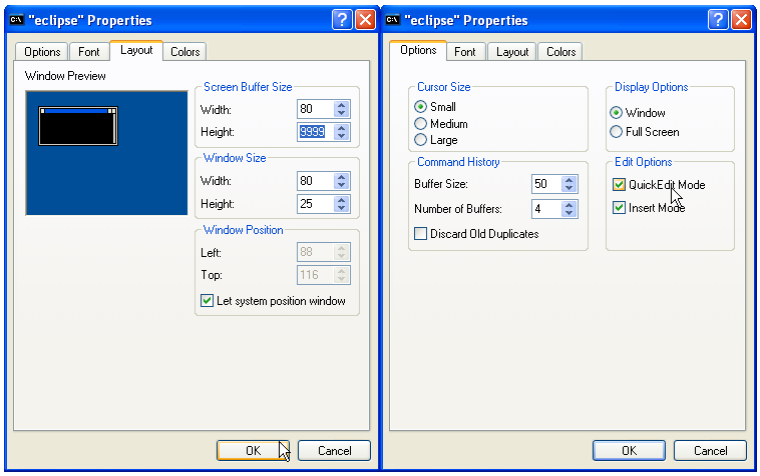
```
C:\eclipse\eclipse.exe
        -vm C:\jre\bin\java.exe
        -data C:\eclipse\workspace
        -debug
        -consolelog
```

The -data should be the location of your workspace, and the -vm should be the location of your Java Runtime.

Alternatively you can create a windows shortcut:

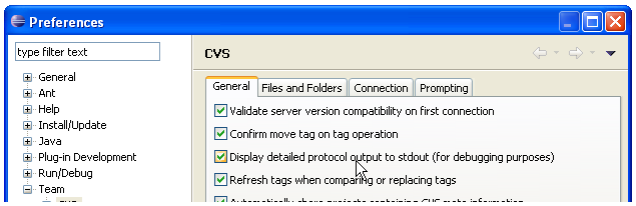






Step 3

Inside Eclipse, in the preferences set the checkbox:
Team->CVS->Display detailed protocol output to stdout.



Step 4

Inside Eclipse, create your repo location and expand it in the repositories view (for example). The CVS command traffic in the debug console should contain an invocation of the update command that looks something like (this is output from dev.eclipse.org):

```
CMD> cvs -n update -d "."
...
update
E cvs server: Updating .
E cvs server: New directory `CVSROOT' -- ignored
E cvs server: New directory `jdt-core-home' -- ignored
E cvs server: New directory `jdt-debug-home' -- ignored
...
```

Create a text file containing all of the output from the debug console by using cut and paste from the debug console window to a notepad window.

Send Debugging information to March Hare Software

Find the ".metadata" directory in your workspace directory and zip it along with the output of the debug console to your March Hare Software Technical Account manager for further assistance.

The .metadata directory contains your workspace preferences, and in particular these files are vital to the support team reproducing your environment:

```
.metadata\.plugins\org.eclipse.core.runtime\.settings

   org.eclipse.team.core.prefs
   org.eclipse.team.cvs.core.prefs
   org.eclipse.team.cvs.ui.prefs
   org.eclipse.team.ui.prefs
```

Administration

Server Settings

The CVSNT Server has several settings that control its behaviour. These settings are used for every repository accessed using that server.

Also see the sections *Protocols* and *Authentication* above.

Running CVSNT as a non-privileged user

The CVS Server will execute on the server hardware as the same user who is checking in the file. For some security requirements this is inadequate and the CVS Server should always execute as a specified user who only has authority in the CVS repository and does not have authority to change files elsewhere on the server.

Use the *RunAsUser* setting in the /etc/cvsnt/Pserver file (on unix) or the *RunAsUser* setting in the Windows registry.

Repository Administrators

If *SystemAuth* is Yes (see above section on *Config*) the user is an Administrator if they are listed in the CVSROOT/admin file or they are in the Windows User Group 'Administrators' or Unix user group 'cvsadmin'.

If *SystemAuth* is No then only the CVSROOT/admin file is checked.

The CVSROOT/admin file contains a list of usernames who are designated as administrators. This file should not be added put under CVS control since it may provide a security risk.

Repository administrators are automatically made members of the CVS User Group *admin*.

CVS Administrators may run *cvs admin* commands.

Read-only Repository Access

There are two methods for defining users as having read only access to the repository. If you are using the *pserver* protocol then you can use the CVSROOT/readonly configuration file (similar to the CVSROOT/admin file). This feature is backwards compatible with older CVS versions.

New CVSNT installations should use Access Control Lists to define what users and groups should have access to which modules and branches.

Read-only Repository

If a repository is set up as a hot standby or mirror, then it should be defined as a read only repository. This can be configured in the Windows Control Panel or on Unix using the /etc/cvsnt/PServer configuration file.

Character sets supported for filenames

CVSNT clients can only version files with filenames that are supported by the character set of the server. If the server is running in Unicode mode then CVSNT clients are able to store filenames encoded in any character set (not just unicode), the client handles the translation to the correct character set for the client.

UTF8 Server Locale

To enable multi lingual filenames run the CVSNT server in UTF8 mode by specifying the *Locale=en_GB.UTF-8* option in the /etc/cvsnt/PServer configuration file (on Unix, Mac OS X, Linux).

If you want to enable this mode on Windows use the windows control panel. The option on Windows has not been tested as thoroughly as on Unix so it has an "experimental" label. Please e-mail support if you have any questions about this.

Temporary Directories on the Server

The CVSNT Server will need to create temporary files and directories on the server. On Windows the location of these files is specified using the Control Panel *CVSNT Server*. This directory should not be virus scanned.

On Unix the location of these temporary files and directories is specified using the environment variable TMPDIR or the *-t* switch.

If you change this directory then you should stop and restart the CVSNT server service (it is not necessary to reboot the server).

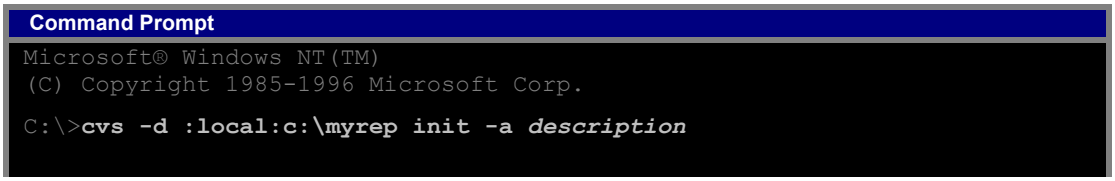
Pserver, TEMP and older versions of CVSNT

Using pserver authentication protocol on a windows server is not recommended. Using versions of CVSNT server for windows prior to 2.5.01.2026 with pserver authentication will require open security settings on the TEMP directory and possibly also the repository. Open security is defined as either Everyone Full Control access, or "Run as User" set to a privileged user.

Creating a Repository

Repositories on a Windows server should always be created on an uncompressed NTFS partition. They should never under any circumstance be created on compressed drive or a network share. Most NAS (Network Attached Storage) devices are merely a network share and unsuitable for a repository. A local disk or RAID array, NAS which is connected to the server using iSCSI or any SAN (Storage Area Network) are suitable for the repository.

There are two ways to create a repository, either using the command line or using the CVSNT Server control panel. To create a repository using the command line use the :local: protocol and the *init* command:



When you define a new repository you can specify:

Location: The absolute path to a local hard drive where the repository will be stored. For example, "c:\my repositories\repository1". Using reserved words like CVS or CVSROOT in repository names can create compatibility and reliability problems and is not recommended. If there is no repository in the location specified in the *add* dialog box then it is created.

Name: A unix-style absolute pathname with a leading / and no spaces or special characters (- and _ are OK). For example, "/repository1". Using reserved words like CVS or CVSROOT in repository names can create compatibility and reliability problems and is not recommended. Using drive letters in repository names can create compatibility problems with Unix client and is not recommended.

Description: A text description of the repository. For example, "Repository 1 for the Alpha group". This is the name displayed by repository browsers such as CVS Suite Studio.

Publish Repository: Enable auto-discovery of the repository when using smart browsers like CVS Suite Studio.

Default Repository: If short repository names are accepted then this is the default repository (usually used in conjunction with an anonymous login for read-only access).

Online: The repository is available for read and write connections.

Administration files

When the repository was created a directory named *CVSROOT* was created containing several configuration files. These are the CVS respoitory administration files.

Administrative file syntax

The administrative files such as *commitinfo*, *loginfo*, *rcsinfo*, *verifymsg*, etc., all have a common format. The purpose of the files is described above. The common syntax is described here.

Each line contains the following:

- A regular expression. This is a basic regular expression in the syntax used by GNU emacs
- A whitespace separator--one or more spaces and/or tabss
- A file name or command-line templates.

Blank lines are ignored. Lines that start with the character # are treated as comments. Long lines unfortunately cannot be broken in two parts in any way.

The first regular expression that matches the current directory name in the repository is used. The rest of the line is used as a file name or command-line as appropriate.

Regular Expressions

A regular expression search term in CVS Administrative files usually must match a module, path and filename on a single line. A regular expression search term may be made up of known and unknown parts, For example:

Search	Matches
Regular Expression	
mymodule	MyModule
	wellmymodulesh
mym.dule	mymodule
	MyModule
	mymindule
components.*	components_are_the_best
	components
	components.dir
components[[:blank:]]*	Components are the best

Match any character "."

The period "." character has a special meaning in a regular expression. It matches any one character. In the examples above the period "." matches either m, M, T. If you want to find an actual period in your filename then you should use the regular expression "\.".

Match previous expression any number of times

The asterisk "*" character has a special meaning in a regular expression. It matches the previous expression any number of times. It is usually used in conjunction with either "." or another regular expression. Note: "components*" would find instances of "components" and "componentssssss" etc.

If you want to find an actual asterisk in your filename then you should use the regular expression "*".

Match a range of characters "[]"

The bracket "[]" characters have a special meaning in a regular expression. It matches a range of characters. In the examples above the bracket expression "[[:blank:]]" matches a single white space character such as space or tab. In the example above the bracket expression is followed by the asterisk to match multiple while space characters.

If you want to find an actual bracket in your filename then you should use the regular expression "\[".

Characters that have special meaning in a regular expression

The following characters all have special meaning within a search expression: "^.[$()|*+?{\". If you want to find one of these characters in your filename then you should use the backslash character immediately before the character that has special meaning.

Expansion in Administrative files

Sometimes in writing an administrative file, you might want the file to be able to know various things based on environment CVS is running in. There are several mechanisms to do that.

HOME

To find the home directory of the user running CVS (from the HOME environment variable), use ~ followed by / or the end of the line. Likewise for the home directory of user, use *~user*. These variables are expanded on the server machine, and don't get any reasonable expansion if the pserver protocol is in use; therefore user variables (see below) may be a better choice to customize behaviour based on the user running CVS.

Internal Variable

One may want to know about various pieces of information internal to CVS. A CVS internal variable has the syntax *${variable}*, where variable starts with a letter and consists of alphanumeric characters and _. If the character following variable is a non-alphanumeric character other than _, the *{* and *}* can be omitted. The CVS internal variables are:

CVS Internal Variables Description	*Variable*
Internal Variable	
This is the name of the current CVSNT repository as the user sees it.	CVSROOT VIRTUAL_CVSROOT
This is the physical location of the current CVSNT repository. Avoid displaying this value to users as it is an information leak..	REAL_CVSROOT
These all expand to the same value, which is the editor that cvsnt is using	CVSEDITOR VISUAL EDITOR
Username of the user running cvsnt (on the cvsnt server machine). When using pserver, this is the user specified in the repository specification which need not be the same as the username the server is running as	USER
Parent process ID of the cvsnt process.	CVSPID
Unique Session ID of cvsnt process. This is a random string of printable characters that may be up to 256 characters long.	SESSIONID COMMITID
Repository prefix, if any, currently active on the server.	PREFIX

If you want to pass a value to the administrative files which the user who is running CVS can specify, use a user variable. To expand a user variable, the administrative file contains *${=variable}*. To set a user variable, specify the global option *-s* to CVS with argument *variable=value*. It may be particularly useful to specify this option via *.cvsrc* .

For example, if you want the administrative file to refer to a test directory you might create a user variable TESTDIR. Then if CVS is invoked as

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

And the administrative file contains *sh ${=TESTDIR}/runtests*, then that string is expanded to *sh /work/local/tests/runtests*.

All other strings containing *$* are reserved; there is no way to quote a *$* character so that *$* represents itself.

Environment variables passed to administrative files

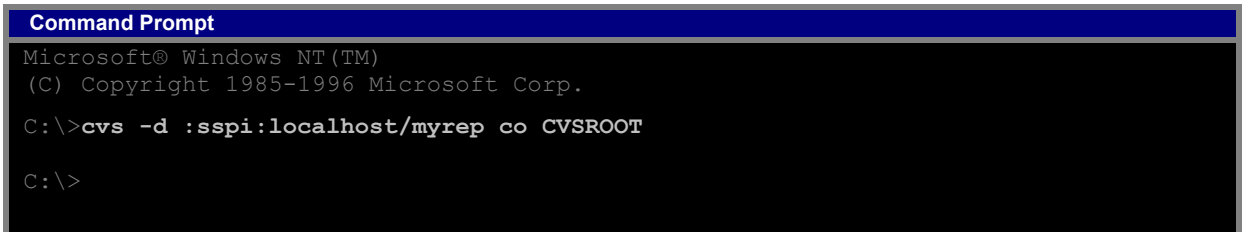
The following environment variables are passed to administrative files:

CVS_USER

The CVS specific username provided by the user, if it can be provided (currently just for the pserver protocol), and to the empty string otherwise. (CVS_USER and USER may differ when *$CVSROOT/CVSROOT/passwd* is used to map CVS usernames to system usernames.)

How to edit Administrative files

Never edit these files directly, use a CVS client to check them out to a local directory to edit them:



modules

The modules file records your definitions of names for collections of source code.

The modules file may contain blank lines and comments (lines beginning with #) as well as module definitions. Long lines can be continued on the next line by specifying a backslash (\) as the last character on the line.

There are three basic types of modules:

- alias modules
- regular modules
- ampersand modules

The difference between them is the way that they map files in the repository to files in the working directory. In all of the following examples, the top-level repository contains a directory called *first-dir*, which contains two files, *file1* and *file2*, and a directory *sdir*. *first-dir/sdir* contains a file *sfile*.

Alias modules

Alias modules are the simplest kind of module:

```
mname -a aliases...
```

This represents the simplest way of defining a module *mname*. The *-a* flags the definition as a simple alias: CVS will treat any use of *mname* (as a command argument) as if the list of names aliases had been specified instead. Aliases may contain either other module names or paths. When you use paths in aliases, checkout creates all intermediate directories in the working directory, just as if the path had been specified explicitly in the CVS arguments.

For example, if the modules file contains:

```
amodule -a first-dir
```

Then the following two commands are equivalent:

```
$ cvs co amodule
$ cvs co first-dir
```

Each of these commands would provide output such as:

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

Regular modules

In the simplest case, this form of module definition reduces to *mname* dir. This defines all the files in directory *dir* as module *mname. dir* is a relative path (from $CVSROOT) to a directory of source in the source repository. In this case, on checkout, a single directory called *mname* is created as a working directory; no intermediate directory levels are used by default, even if dir was a path involving several directory levels.

```
mname [ options ] dir [ files... ]
```

For example, if a module is defined by:

```
regmodule first-dir
```

Then *regmodule* will contain the files from *first-dir*:

```
$ cvs co regmodule
cvs checkout: Updating regmodule
U regmodule/file1
U regmodule/file2
cvs checkout: Updating regmodule/sdir
U regmodule/sdir/sfile
$
```

By explicitly specifying files in the module definition after *dir*, you can select particular files from directory *dir*. Here is an example:

```
regfiles first-dir/sdir sfile
```

With this definition, getting the *regfiles* module will create a single working directory *regfiles* containing the file listed, which comes from a directory deeper in the CVS source repository:

```
$ cvs co regfiles
U regfiles/sfile
$
```

Ampersand modules

A module definition can refer to other modules by including &module in its definition.

```
mname  [ options ] &module ...
```

Then getting the module creates a subdirectory for each such module, in the directory containing the module. For example, if modules contains

```
ampermod &first-dir
```

Then a checkout will create an *ampermod* directory that contains a directory called *first-dir*, which in turns contains all the directories and files that live there. For example, the command

```
$ cvs co ampermod
```

Will create the following files:

```
ampermod/first-dir/file1
ampermod/first-dir/file2
ampermod/first-dir/sdir/sfile
```

There is one quirk/bug: the messages that CVS prints omit the *ampermod*, and thus do not correctly display the location to which it is checking out the files:

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

Excluding directories

An alias module may exclude particular directories from other modules by using an exclamation mark (!) before the name of each directory to be excluded.

For example, if the modules file contains:

```
exmodule -a !first-dir/sdir first-dir
```

Then checking out the module *exmodule* will check out everything in first-dir except any files in the subdirectory *first-dir/sdir*.

Module options

Either regular modules or ampersand modules can contain options, which supply additional information concerning the module.

Module options	*Switchl*
Module	
Name the working directory something other than the module name	-d name

Module options	Switchl
Specify a program prog to run whenever files in a module are exported. prog runs with a single argument, the module name	-e prog
Specify a program prog to run whenever files in a module are committed. prog runs with a single argument, the full pathname of the affected directory in a source repository. The commitinfo, loginfo, and verifymsg files provide other ways to call a program on commit	-i prog
Specify a program prog to run whenever files in a module are checked out. prog runs with a single argument, the module name.	-o prog
Assign a status to the module. When the module file is printed with cvs checkout -s the modules are sorted according to primarily module status, and secondarily according to the module name. This option has no other meaning. You can use this option for several things besides status: for instance, list the person that is responsible for this module.	-s status
Specify a program prog to run whenever files in a module are tagged with rtag. prog runs with two arguments: the module name and the symbolic tag specified to rtag. It is not run when tag is executed. Generally you will find that taginfo is a better solution.	-t prog

How the modules file "program options" programs are run

For *checkout*, *rtag*, and *export*, the program is server-based, and as such the following rules apply:

- If using remote access methods (pserver, ext, etc.), cvsnt will execute this program on the server from a temporary directory. The path is searched for this program
- If using "local access" (on a local or remote NFS filesystem, i.e. repository set just to a path), the program will be executed from the newly checked-out tree, if found there, or alternatively searched for in the path if not.

The programs are all run after the operation has effectively completed.

modules2

The *modules2* file provides a lower level definition of modules than the *modules* file. CVS Clients see the *modules2* structure as if it existed physically on the server.

How the modules2 file differs from the modules file

The *modules* provides different types of module, which are 'high level', in that checking out a module is equivalent to calling checkout multiple times on different directories. This approach works well for simple cases, but breaks down in the more complex cases, causing unwanted interactions with the update command for example.

The *modules2* has only one way of describing a module, but operates on a much lower level. Clients are unaware that the directory structure that they are checking out does not actually exist, and all CVS commands behave as normal. A file or directory defined by *modules2* may have a completely different name to its real name, and updates/merging will be handled correctly even if multiple clients checkout under different names.

Which file you choose depends on your requirements. It isn't recommend that usage is mixed between the two files as they both serve a similar function and it would get confusing.

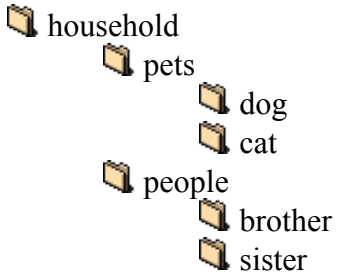
Modules2 syntax

The *modules2* file is structured in a similar way to the familiar Windows .ini file. Each section defines a module, and within each section is a description of the files and directories within that module.

An example modules2 file is:

```
[pets]
dog
cat

[people]
brother
sister

[household]
pets
people
```

Checking out *household* will create the directory structure:



In this example the *household*, *pets*, and *people* directories don't have any files in them - they're just containers. However let's say we want to put the files listing pet food in the *pets* directory, above all the pet specific directories.

Modules2 lets you override what goes in the root of a module, to overlay another module in it:

```
[pets]
/ = !petfood
dog
cat

[people]
brother
sister

[household]
pets
people
```

When we checkout we get the same directory structure as above, but the pets directory contains the contents of *petfood*. Note also we said that we don't want any subdirectories of *petfood*, using the *!* prefix. This makes sure that the directory is never recursed into, even during an *update -d*. We still get the *dog* and *cat* directory of course.

You can simply rename an entire directory tree using this method. The following:

```
[project1]
/ = myproject

[project2]
/ = myproject
junk =
total_junk =
project/old_project = myproject/junk
```

project1 will checkout the entire *myproject* tree. *project2* is the same, except the *junk* directory is removed, and moved to *project/oldproject*. The *total_junk* directory is hidden completely.

You can also mask certain files within a directory, or certain subdirectories using an extended regular expression.

```
[project1]
/ = myproject

[project2]
/ = myproject (*\.cpp$|*\.[ch]$|*/$)
junk =
total_junk =
project/old_project = myproject/junk
```

Directories are subject to the same filtering, except they have a '/' directory separator after their name. To filter some files and allow subdirectories add '|*/$' as an option.

Further options

The '+' prefix stops processing, so that entries that would be potentially recursive can be defined to be nonrecursive.

Spaces can be used in the file, delimited by quotes or using backslash escapes. File separators must always be forward slashes.

Comments are on a line beginning '#'

cvswrappers

wrappers refers to a CVS feature which allows you control certain settings based on the name of the file which is being operated on.

The most common use of this file is to set files with certain file extensions to:

- Be of type Binary
- Use Binary Deltas
- Checkout reserved by default
- Do not commit changes unless forced
- Be of type Unicode

The basic format of the file cvswrappers is:

```
wildcard     [option value][option value]...

where option is one of
-m    update methodology      value: MERGE or COPY
-k    keyword expansion       value: substitution mode and encoding
-x    xdiff wrappers          value: name of xdiff DLL, plus options.
-t    mime types              value: new mime type
```

And *value* is a single-quote delimited value. See *Substitution modes and Encoding* below for more details.

In addition, a file pattern of `*.*` or just `*` will be used as a default where no wrappers exist. This pattern may also contain addive or subtractive wrapper options (eg -k+x), in which case it will always be applied.

Wrappers may also be used from the command line client, for example, the following command imports a directory, treating files whose name ends in .exe as binary:

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

Substitution modes (Flags) and Encoding

Keyword substitution modes are stored for each version of a file. When you commit a new revision that version will be stored exactly as it is in the sandbox, including the substitution mode.

You can override the substitution mode by using the -k option to cvs add, -k or -A options to cvs checkout or cvs update. cvs diff also has a -k option.

The cvs update and cvs checkout commands allow you to modify existing substitution modes without overwriting the existing ones. This is done by prefixing the mode with '+' (for add) or '-' (for remove). For example to switch off keyword substitution for all files in a subtree:

```
$ cvs update -k+o
$ cvs commit -fm "Change substitution mode"
```

The modes available are defined by combining an optional encoding with a series of options.

Some combinations were not available on older CVS versions so be careful if you want to access your repository from older clients. The CVSNT server will automatically downgrade some of these options if an older client fetches a file.

Encodings	
t	Treat the file as a text file. This is the default setting if no encoding is specified. MBCS character sets that don't change the behaviour of CR/LF and NULL should also work in this mode. eg. Shift-JIS and EUC.
b	Treat the file as binary. No interpretation is done of the contents and they are stored verbatim. Be default no keyword expansion is done. Binary files are considered non-mergable by CVS.
B	Treat the file as binary. No interpretation is done of the contents and they are stored verbatim. Be default no keyword expansion is done. Binary files are considered non-mergable by CVS. In addition, an alternate storage algorithm is used that is optimised for storage of binary files.
U	Treat the file as Unicode. The file will be checked in/out in UCS-2 (or UTF-16) encoding and internally stored as UTF-8 by the server.
{...}	Use an extended encoding. Any encoding supported by the client-side iconv library can be used, however beware of mismatches between clients (the Win32 version does not currently support EBCIDIC encodings for example). The following list will work on all platforms that are using Unicode-capable CVSNT:
ucs2le, utf16le	Little-endian UCS-2 without BOM.
ucs2be, utf16be	Big-endian UCS-2 without BOM.
ucs2le_bom, utf16le_bom	Little-endian UCS-2 with BOM.
ucs2be_bom, utf16be_bom	Big-endian UCS-2 with BOM.

Flags	
k	Preserve the keyword string (default). When combined with the v flag this generates results using the default form, e.g. $Revision$ for the Revision keyword. On its own it produces output with no keywords expanded.
v	Generate keyword values for keyword strings. Normally paired with the k option. If it is used on its own the effect is to strip keywords from the output - for example, for the Revision keyword, generate the string 5.7 instead of $Revision: 1.1.2.1$. This can help generate files in programming languages where it is hard to strip keyword delimiters like $Revision$ from a string. However, further keyword substitution cannot be performed once the keyword names are removed, so this option should be used with care.
l	Insert the lockers name if the given revision is currently locked. The locker's name is only relevant if cvs admin -l is in use.
o	Generate the old keyword string, present in the working file just before it was checked in. For example, for the Revision keyword, generate the string $Revision: 1.1.2.1;$ instead of $Revision$ if that is how the string appeared when the file was checked in.
L	When checking out, always create a file with Unix line endings no matter what the native line ending format is.
Z	Compress the files and deltas when they are stored. This sacrifices speed for disk space - only use if disk space is at a premium.
1	Unversioned. Only ever keep a single revision in the repository. History is lost, and the file is merely kept as the latest copy. Only one branch (HEAD) ever exists and an update o any revision will return the single revision.
s	File is never considered modified on the client. A normal commit will never commit this file, unless -f is used to force it. Use this option with care, and for files that change infrequently, since local changes will be lost on update.

The commit support files

The *-i* flag in the *modules* file can be used to run a certain program whenever files are committed (the section above). The files described in this section provide other, more flexible, ways to run programs whenever something is committed.

There are three kind of programs that can be run on commit. They are specified in files in the repository, as described below. The following sections summarise the file names and the purpose.

commitinfo

The program is responsible for checking that the commit is allowed. If it exits with a non-zero exit status the commit will be aborted.

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

Each line in the *commitinfo* file consists of a regular expression and a command-line template. The template can include a program name and any number of arguments you wish to supply to it. The full path to the current source repository is appended to the template, followed by the file names of any files involved in the commit (added, removed, and modified files).

The first line with a regular expression matching the directory within the repository will be used. If the command returns a non-zero exit status the *commit* will be aborted.

If the repository name does not match any of the regular expressions in this file, the DEFAULT line is used, if it is specified.

All occurrences of the name ALL appearing as a regular expression are used in addition to the first matching regular expression or the name DEFAULT.

Note: when cvsnt is accessing a remote repository, commitinfo will be run on the remote (i.e., server) side, not the client side.

verifymsg

The specified program is used to evaluate the log message, and possibly verify that it contains all required fields. This is most useful in combination with the *rcsinfo* file, which can hold a log message template (the section *rcsinfo*).

The *verifymsg* script may or may not be able to change the log message depending on the value of the *RereadLogAfterVerify* setting in the *config* administrative file.

For example you can evaluate that message to check for specific content, such as a bug ID. Use the *verifymsg* file to specify a program that is used to verify the log message. This program could be a simple script that checks that the entered message contains the required fields.

The *verifymsg* file is often most useful together with the rcsinfo file, which can be used to specify a log message template.

Each line in the *verifymsg* file consists of a regular expression and a command-line template. The template must include a program name, and can include any number of arguments. The full path to the current log message template file is appended to the template.

The ALL keyword is not supported. If more than one matching line is found, the first one is used. This can be useful for specifying a default verification script in a directory, and then overriding it in a subdirectory.

If the repository name does not match any of the regular expressions in this file, the DEFAULT line is used, if it is specified.

If the verification script exits with a non-zero exit status, the commit is aborted.

taginfo

The "taginfo" file is used for triggering events that occur when objects are tagged or branched.

notify

The "notify" file controls where notifications from watches set by the CVS command *watch add* or *edit* are sent.

The first entry on a line is a regular expression which is tested against the directory that the change is being made to, relative to the $CVSROOT. If it matches, then the remainder of the line is a filter program that should contain one occurrence of *%s* for the user to notify, and information on its standard input.

"ALL" or "DEFAULT" can be used in place of the regular expression.

You may specify a format string as part of the filter. The format characters are:

Format Character Meaning	*character*
Notify	
User being notified	s
Bug identifier	b
Message supplied on command line	m
Date of action	d
User performing the unedit	u
Tag or branch being edited	t

For example:

```
ALL mail %s -s "CVS notification for bug %b"
```

editinfo

Editinfo is no longer available. Use *Verifymsg* instead.

loginfo

The specified program is called when the commit is complete. It receives the log message and some additional information and can store the log message in a file, or mail it to appropriate persons, or maybe post it to a local newsgroup, or... Your imagination is the limit!

The *loginfo* file is used to control where CVS *commit* log information is sent. The first entry on a line is a regular expression which is tested against the directory that the change is being made to, relative to the $CVSROOT. If a match is found, then the remainder of the line is a filter program that should expect log information on its standard input.

If the repository name does not match any of the regular expressions in this file, the DEFAULT line is used, if it is specified.

All occurrences of the name ALL appearing as a regular expression are used in addition to the first matching regular expression or DEFAULT.

The first matching regular expression is used.

The user may specify a format string as part of the filter. The string is composed of a % followed by a space, or followed by a single format character, or followed by a set of format characters surrounded by { and } as separators.

The format characters are:

Format Character Meaning	*character*
Loginfo	
Module name, followed by the list of filenames	s
Current version number (pre-checkin)	V
New version number (post-checkin)	v
Message supplied by user	M
Status string	T
Directory name relative to the current root	P
Bug identifier	b
Tag	t
Commit type: update, modify, add etc	y
File name, old version number (pre-checkin), new version number (post-checkin) compatible with previous versions of CVS	{sVv}

All other characters that appear in a format string expand to an empty field (commas separating fields are still provided).

For example, some valid format strings are %, %s, %{s}, and %{sVv}.

The output will be a string of tokens separated by spaces. For backwards compatibility, the first token will be the repository subdirectory. The rest of the tokens will be comma-delimited lists of the information requested in the format string. For example, if */u/src/master/yoyodyne/tc* is the repository, %{sVv} is the format string, and three files (*ChangeLog*, *Makefile*, *foo.c*) were modified, the output might be:

```
yoyodyne/tc ChangeLog,1.1,1.2 Makefile,1.3,1.4 foo.c,1.12,1.13
```

As another example, %{} means that only the name of the repository will be generated.

Note: when CVS is accessing a remote repository, *loginfo* will be run on the remote (i.e., server) side, not the client side.

Loginfo default standard input format

For both commit and import the first two lines are the following:

```
Update of %r/%p
```

```
In directory %H:%P
```

Next part is different for import and commit. For commits there comes line with the current operation/operations, namely "Added Files:", "Removed Files:" or "Modified Files:". In the next lines there are, indented with TAB, space separated list of added, removed or modified files. There is no such section for added directories (because one can remove empty directories only with checkout/update with -P option, not by commit). There is instead "Directory $CVSROOT/subdirectory added to repository" log message. For import next part (separated by empty line) is the log message:

```
Log Message:
```

```
%m
```

This part is also after commit, but for commit it is at the very end of input, and is _not_ separated by an empty line. Further parts are for import solely. After log message, separated by empty line comes:

```
Status:
```

Next is the information about release and vendor tag (see 'cvs import' syntax), separated of course from log message by an empty line, namely

```
Vendor Tag: vendor_tag
Release Tags: release_tag
```

Next, separated by an empty line, is the output of the import command. The format is

```
X module_dir/subdir/file
```

where X is one letter indicator of status. The last line is the status of import command, e.g.

```
No conflicts created by this import
```

triggers

This file specifies the location of a trigger library to be loaded on server startup. On Windows this may also be the Class ID of a COM object resident on the server.

postcommit

The specified program is called when the commit is complete, and all locks have been released from the repository prior to returning to the user. This is useful to maintain checked-out copies of repositories and to perform CVS actions after a commit.

The post commit trigger is passed the name of the last directory processed relative to the root. This directory may or may not have any processing performed in it. Eg:

If the user checks out the module uwork:

```
cvs -d :sspi:localhost:/myrepo uwork
```

And then makes changes to `asn/fred.asn` and commits then the directory passed to postcommit trigger is `myrepo/wilma/textfiles`. This directory is passed even though the user commits at the root `uwork`, and the only file changed is in `uwork/asn`.

postcommand

The specified program is called when the command is complete, and all locks have been released from the repository prior to returning to the user. This is useful to maintain checked-out copies of repositories and to perform CVS actions after any action (eg: tag/commit).

The post command trigger is passed the name of the last directory processed relative to the root. This directory may or may not have any processing performed in it. Eg:

If the user checks out the module uwork:

```
cvs -d :sspi:localhost:/myrepo uwork
```

And then makes changes to `asn/fred.asn` and commits then the directory passed to postcommit trigger is `myrepo/wilma/textfiles`. This directory is passed even though the user commits at the root `uwork`, and the only file changed is in `uwork/asn`.

historyinfo

This is called when any action that causes an entry in the history file is initiated. Its standard input receives the line to be written to the history log, in a semi-compressed format.

Each time an action is performed on the repository that generates a line in the history file, scripts *historyinfo* are executed so that you can perform advanced logging, for example to a separate machine.

The script is passed a line of history via its standard input in the following format:

```
Action|Date|Repository|Directory|Version|File
```

rcsinfo

The *rcsinfo* file can be used to specify a form to edit when filling out the commit log. The rcsinfo file syntax is similar to the *verifymsg*, *commitinfo* and *loginfo* files. The section *The common syntax* describes this. Unlike the other files the second part is not a command-line template. Instead, the part after the regular expression should be a full pathname to a file containing the log message template.

If the repository name does not match any of the regular expressions in this file, the DEFAULT line is used, if it is specified.

All occurrences of the name ALL appearing as a regular expression are used in addition to the first matching regular expression or DEFAULT.

The log message template will be used as a default log message. If you specify a log message with *cvs commit -m message* or *cvs commit -f file* that log message will override the template.

When CVS is accessing a remote repository, the contents of *rcsinfo* at the time a directory is first checked out specifies a template that does not then change. If you edit *rcsinfo* or its templates, you may need to check out a new working directory.

cvsignore

There are certain file names that frequently occur inside your working copy, but that you don't want to put under CVS control. Examples are all the object files that you get while you compile your sources. Normally, when you run the CVS command *update*, it prints a line for each file it encounters that it doesn't know about.

CVS has a list of files (or file name patterns) that it should ignore while running update, import and release. This list is constructed in the following way:

- The list is initialized to include certain file name patterns: names associated with CVS administration, or with other common source control systems; common names for patch files, object files, archive files, and editor backup files; and other names that are usually artifacts of assorted utilities. Currently, the default list of ignored file name patterns is:

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

- The per-repository list in $CVSROOT/CVSROOT/cvsignore is appended to the list, if that file exists
- The per-user list in .cvsignore in your home directory is appended to the list, if it exists
- Any entries in the environment variable $CVSIGNORE is appended to the list
- Any -I options given to cvsnt is appended.

As CVS traverses through your directories, the contents of any *.cvsignore* will be appended to the list. The patterns found in *.cvsignore* are only valid for the directory that contains them, not for any sub-directories.

In any of the 5 places listed above, a single exclamation mark (!) clears the ignore list. This can be used if you want to store any file that normally is ignored by CVS.

Specifying *-I !* to the CVS command *import* will import everything, which is generally what you want to do if you are importing files from a pristine distribution or any other source which is known to not contain any extraneous files. However, looking at the rules above you will see there is a fly in the ointment; if the distribution contains any *.cvsignore* files, then the patterns from those files will be processed even if -I ! is specified. The only workaround is to remove the *.cvsignore* files in order to do the import. Because this is awkward, in the future *-I !* might be modified to override *.cvsignore* files in each directory.

Note that the syntax of the ignore files consists of a series of lines, each of which contains a space separated list of filenames. This offers no clean way to specify filenames which contain spaces, but you can use a workaround like *foo?bar* to match a file named *foo bar* (it also matches *fooxbar* and the like). Also note that there is currently no way to specify comments.

checkoutlist

It may be helpful to use CVS to maintain your own files in the CVSROOT directory. For example, suppose that you have a script *logcommit.pl* which you run by including the following line in the *commitinfo* administrative file:

```
ALL    $CVSROOT/CVSROOT/logcommit.pl
```

To maintain logcommit.pl with CVS you would add the following line to the checkoutlist administrative file:

```
logcommit.pl
```

The format of *checkoutlist* is one line for each file that you want to maintain using CVS, giving the name of the file.

After setting up *checkoutlist* in this fashion, the files listed there will function just like CVS's built-in administrative files. For example, when checking in one of the files you should get a message such as:

```
cvs commit: Rebuilding administrative file database
```

And the checked out copy in the CVSROOT directory should be updated.

Note that listing *passwd*, *group*, *admin*, *readonly* etc in *checkoutlist* is not recommended for security reasons.

users

The users file contains a list email addresses for users. See the section *Integration with Bugzilla, Mantis or Jira Defect Tracking* and the sub heading *Configure the commit support files* for more detail on using this.

passwd

The password file contains a list of users who are allowed to access CVS.

pserver and sserver protocol

If SystemAuth is set to YES then CVSNT will authenticate using the operating system first, then if it fails use the passwd file. If SystemAuth is set to NO then CVSNT will authenticate using the passwd file.

Sspi protocol

If SystemAuth is set to YES then this file is not used. If SystemAuth is set to NO then it is used as a filter, where only the listed users may log in but the password is authenticated using the Windows Active Dircetory.

Syntax

The passwd file has three fields:

- User name
- Encryted passwd (optional)
- Alias user (optional)

group

The group file contains a list of groups and which users belong to them. This is primarily used in the Access Control (see ACL's).

If SystemAuth is set to YES then CVSNT will use the operating system definition of groups and then the Group file in the CVSROOT.

If SystemAuth is set to NO then CVSNT will use the Group file in the CVSROOT only.

Syntax

The group file has two fields:

- Group name
- Space separated list of users who belong to this group

config

The administrative file *config* contains various miscellaneous settings which affect the behaviour of CVS. The syntax is slightly different from the other administrative files. Variables are not expanded. Lines which start with # are considered comments. Other lines consist of a *keyword*, =, and a *value*. Note that this syntax is very strict. Extraneous spaces or tabs are not permitted.

config Administration File Keyword Descriptions	***Keyword***
config	
If value is yes, then CVS Server should check for users in the system's user database if not found in *CVSROOT/passwd*. If it is no, then all CVS Server users must exist in *CVSROOT/passwd*. The default is yes.	SystemAuth=value

config Administration File Keyword Descriptions	*Keyword*
Specify if ACL's should automatically permit all users or deny all users. Ie: if you automatically deny all users then an administrator will have to permit a user before they can access the repository. Compat = default value meaning all users are permitted and have to be denied access to restrict security on a file, directory or module. Normal = denys all users unless specifically granted access by an ACL.	AclMode=value
Set 'Watcher' to set a user who gets all notify events within the repository whether or not the file is watched.	Watcher=watch_user
Modify the checkout command to create a CVS directory at the top level of the new working directory, in addition to CVS directories created within checked-out directories. The default value is no. This option is useful if you find yourself performing many commands at the top level of your working directory, rather than in one of the checked out subdirectories. The CVS directory created there will mean you don't have to specify CVSROOT for each command. It also provides a place for the CVS/Template file	TopLevelAdmin=value

config Administration File Keyword Descriptions	Keyword
Uses the CVS deadlock server to handle locking rather than using files in the repository. This is useful if you want to let users read from the repository while giving them write access only to directory, not to the repository. CVSNT 2.0.15 and above use the Deadlock Server (Lockserver) by default and other methods of locking are depreciated. You can override this behaviour by using the line *LockServer=none*. Note however that future versions may not allow this override. CVS 1.10 does not support LockServer but it will print a warning if run on a repository with LockServer enabled.	LockServer=hostname[:port]
Control what is logged to the *CVSROOT/history* file. Default of TOFEWGCMAR (or simply all) will log all transactions. Any subset of the default is legal. (For example, to only log transactions that modify the *,v files, use LogHistory=TMAR.)	LogHistory=value
If enabled the log message parsed by verifymsg is reread after the script has run. The default behavoiur is to not reread this file	RereadLogAfterVerify=value

Defining Binary File Types

CVS assumes that all files stored in the repository are text files. Therefore it is necessary to configure it so that some files are stored as binary. This configuration is based on *filename extension*. The *filename extension* is the letters after the final period (.) in a filename.

It is also possible to define keyword expansion policies for files in the same way.

See the earlier documentation on *cvswrappers* for more information.

Wherever possible you should define binary files (including word documents) as –k 'B' which uses an alternate storage algorithm is used that is optimised for storage of binary files. Binary files are considered non-mergable by CVS.

Importing Modules

You may create your repository using the CVS command *import* or by using the *add* command on each file and directory.

The best combination is to use the CVS command *import* with

-*n* Don't create the vendor branch and release tags

-*C* Create CVS directories on import

-*m* supply a comment

Using these options will leave you with a usable sandpit in place of the original files.

How to import existing projects that have multiple versions.

If you are implementing version control for the first time – but have archived copies of previous versions of your software (eg: on CD) then you can use the *fixud6* program (windows only) to import multiple versions of your project.

Here is a brief overview of the procedure:

1) Import oldest version using *cvs import -n -C* as descriped above in the section *Importing Modules*.
2) Tag this version with the release number using *cvs tag rel1-0* , or create a branch using *cvs tag -b rel1-0*
3) Use *fixud6 -del* to delete all the files from the workspace but leave the CVSNT administration files intact
4) Copy the next version into the first directory
5) Use *fixud6 -add* to add all the new files and directories in this version (it will ignore files that begin with a period (.) and files with the following suffixes: ".obj", ".lib", ".lst", ".alt", ".std", ".tst", ".sav", ".old"
6) Use the *cvs commit* command to put this version into the repository
7) Go to step 2)□

Importing Vendor Code (Vendor Branches)

Vendor branches is a feature of CVSNT that allows you to track changes to code that a vendor supplies and you customise.

Example

The most common vendor code of this type is configuration (.INI) files. The vendor provides a default INI file and you customise the application settings and preferences resulting in a modified INI file. When you install an upgraded copy of the program in a new directory you find that the INI file has new settings and will not recognise the older INI files.

In this situation you can merge your changes to the original file with the vendors new file. This is the purpose of a vendor branch.

Limitations

Vendor branches will not work with very short files. If the file has only a few lines in it then CVSNT cannot accurately determine what has changed and will assume that the files are completely different.

Sequence of operation

Unmodified vendor code must be imported into the CVSNT repository on a "Vendor branch" and your modifications made on the Trunk. The following describes the sequence:

- *cvs import -m "Import of March Hare SuperApp version 1.0" march-hare/SuperApp MARCH-HARE SUPERAPP_1_0*
- *cvs co march-hare/SuperApp*
- make your modifications
- commit your changes
- *cvs import -m "Import of March Hare SuperApp version 1.2" march-hare/SuperApp MARCH-HARE SUPERAPP_1_2*
- Go to the directory holding your currrent configuration files and merge in the changes using *cvs update -j SUPERAPP_1_2*

Adding Administration Files to the CVSROOT

See the earlier section on *checkoutlist*.

Messing with the RCS files

In general you should not ever need to modify the RCS files directly.

In the following situations you may find it useful to directly move or delete the RCS files:

- If a file was imported into the wrong directory
- If a file was imported that should have been ignored.

It was common practice to also modify the RCS files to rename a file (eg: from uppercase to mixed case). The next major release of CVSNT handles this situation natively, and modifying the RCS files in this way directly should be avoided.

Creating Branches

Creating branches is simply a matter of running the CVS command *tag* with the *-b tag* option on a checked out copy (sandpit) .

It is also possible to create the branch on the server using rtag however it is more obvious which files and versions are being tagged by performing the operation in a sandpit.

Also see the section *Advanced Client Functions* for information on this topic.

Creating Promotion Levels

CVS uses branches to implement a promotion level. The difference is simply how you use them.

See the section *Advanced Client Functions* for information on this topic.

Promoting

There are two ways to promote a revision from the Trunk (or a branch) to a promotion level: Move and Merge. See the section *Advanced Client Functions* for information on the Promote Move and Promote Merge topics.

Merging-in Branches

You can merge changes made on a branch into your working copy by giving the CVS command *update* with the *-j branchname* flag. See the section *Advanced Client Functions* for information on Merging-in Branches and Using Mergepoints.

Bug ID's (User Defined Change Sets)

Most CVS commands now support a a user defined change set, bug or defect identifier. The identifier can be used to group or mark files, and the identifier can also be used to operate on files with that identifier. See the *Client* section for more information on how to use this feature in the client, and the *Integration with Bugzilla* heading in this section for how to use the bug number on the server.

Generic Triggers

New triggers *precommand*, *premodule* and *postcommand*, *postmodule* simplify the complexity of triggers, and allow a greater level of flexibility in defining CM Policy on the CVS Server.

Customers are advised to code carefully to avoid recursion when using the *postcommand* trigger.

Access Control Lists

With Access Control Lists you can define complex rules for which users or groups of users can perform different tasks on parts of your CVS repository. Access Control Lists are most commonly used to prevent unauthorised merging, update or delete of objects from test or production branches.

This security information exists outside of the permissions on the individual files, and therefore is kept in place when the repository is restored from backup, moved between servers or moved between server platforms (eg: Windows to Unix).

Access Control Lists are primarily designed to control access to CVS modules and directories. It is possible to set different access control levels to different files in a directory however the results of the "none" keyword may not be intuitive. You should always assume that if a person has permission to read a directory that contains a file that they also have permission to read that file.

Historyfile

The *historyfile* is created automatically when you create a new repository, and many CVS commands that connect to the server will cause a record to be written to this file. The history can then be browsed and searched using the *cvs history* command.

Access Control Lists do not apply to the *historyfile*. This means that if you are using the history then users without access to some files and directories will be able to see what the names of those files and directories are and the activity which has occurred on them.

You can disable the history by removing the historyfile on the server. Note: removing the historyfile does not affect the CVSNT client operation, however some non-CVSNT clients such as SmartCVS may no longer work. SmartCVS have been advised of an alternative way to obtain the information that their product currently receives from history.

Chacl command

Using CVS command *chacl* you can define access control lists to manage security on the CVS repository.

```
cvs chacl [-R] [-r branch] [-u user] [-j branch] [-n] [-p priority] [-m message] [-a
[no]{read|write|create|tag|control}[,...]]
[-d] [file or directory...]
        -a access       Set access
        -d              Delete ACL
        -j branch       Apply when merging from branch
        -m message      Custom error message
        -n              Do not inherit ACL
        -p priority     Override ACL priority
        -r branch       Apply to single branch
        -R              Recursively change subdirectories
        -u user         Apply to single user
```

The *chacl* command operates with a locally checked out copy of your files.

Access Roles

CVSNT defines the following access roles:

- Read - able to read this file
- Write - able to add revisions to this file
- Create - able to add a file to the repository
- Tag - able to apply tags to files and create branches
- Control - able to administer this file and change ACL's

Access by Groups

The Access Control Lists by default assign permissions to a user. Any groups defined in the group administration file can also be used as a username. See the section on the group administration file for more information.

Typically if you are using the SSPI protocol on a Windows server the groups are the Windows Active Directory Groups, not the groups defined in the group administration file.

Default ACL's

Every object stored in the CVSNT repository has a default ACL. This can be set using the following command:

```
cvs chacl -a access -R
```

If no access is specifically set, then the default is that all users can *read*, *write*, *tag* and *create*. The *control* permission is reserved for the members of the *admin* group (Administrators on windows).

Branch ACL's

Each branch will use the repository default ACL unless set otherwise. The branch access can be set using the following command:

```
cvs chacl -r branch -a access -R
```

Commit ID's

Every commit transaction in a CVS repository has a unique commit ID. This ID can then be used to re-action the exact same change set at a later date.

Commit ID's are also available in the administrative files as a private variable.

For example, to rollback a commit that you just performed:

```
cvs update -j @commitid -j "@<commitid" filename or modulename
```

"@<commitid" revision before that commit

@commitid revision of that commit

Release tags

tag=tag

To allow customers with multiple deployment environments for their applications it is now possible to base one tag on another.

So the production environments can checkout the branch *Production* rather than *prod_1_2_2*. When release 1.2.3 is ready for production the administrator sets *Production=prod_1_2_3* and the next time a CVS *update* command is issued in the production environment the new production version will be installed.

Branch alias

Branch x is also known as y

```
cvs tag -A -r existing-tag new-tag
```

Branch point in time alias

Branch x at this point in time is known as y

```
cvs tag -r existing-tag new-tag
```

Implementing Methodologies

In the first chapter we discussed four methodologies for versioning, and other decisions related to your management objectives and company culture.

This section provides a guide to how to implement those choices with server side configuration.

Some configuration is also required in how the client is used which is discussed in the Clients section.

Implementing the methodology is generally a lot simpler than deciding what it should be. It is strongly recommended that you should decide what versioning methodology will best meet your business objectives and fit within your company culture before installing and configuring CVSNT server or any clients.

Reserved / Unreserved

By default CVSNT server uses the unreserved model. To enforce use of a reserved model you will need to make configuration changes to the server, and also to the client.

Reserved

CVSNT has four implementations of the reserved model:

Administration Reserved – Not recommended

The administrator can reserve all versions and all branches of a file. This is a very basic lock and its use is not recommended.

All reserved modes use watches

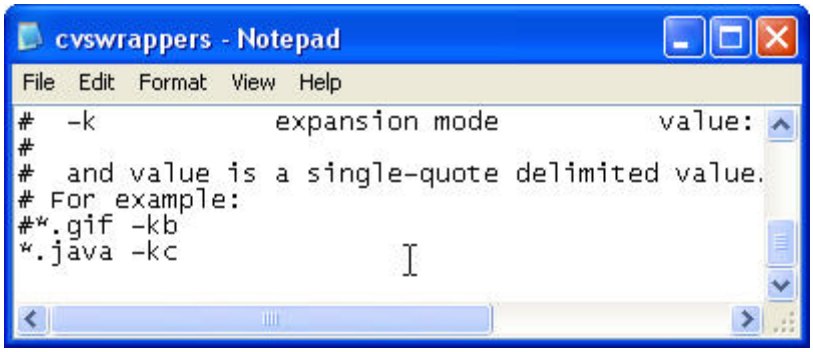
To use any of the reserved modes it is necessary on the client to enable watches using “cvs watch on” in the working directory (where the repository is checked out). Clients such as Eclipse and Tortoise have settings that enforce this automatically. See the Clients section for more information.

Reserved for Communication Only

This is not really reserved at all however is worth mentioning. Using this technique all users can edit the same version of the same file at the same time, however each is informed that other users are editing it. To enable this requires no changes on the server.

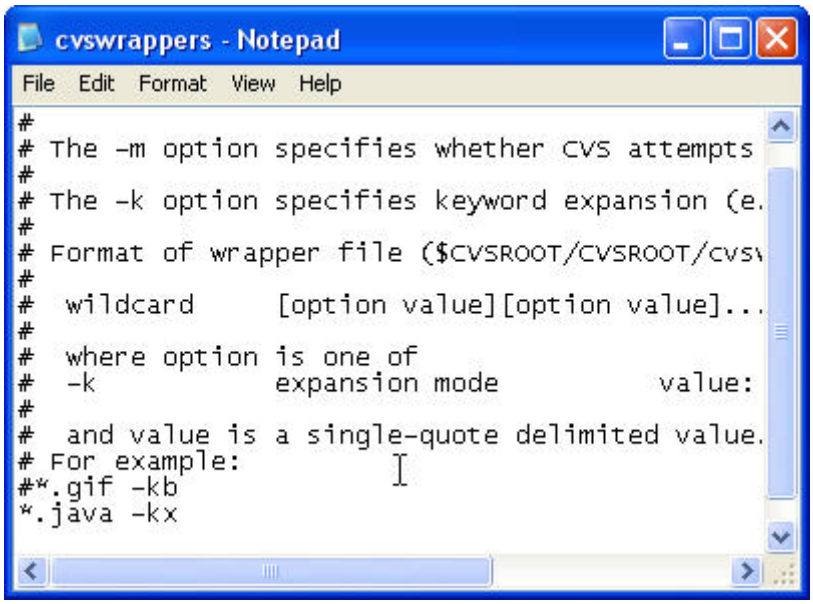
Reserved Co-operatively

This is traditional reserved editing while still allowing users to override it and work in an unreserved manner. To enable this edit the *cvswrappers* administration file, set the file types which should use this mode to –kx (or –kvx etc).



Reserved Exclusive

This is traditional reserved editing and will prevent CVSNT working in an unreserved manner. To enable this edit the *cvswrappers* administration file, set the file types which should use this mode to –kx (or –kvx etc). By setting this in the administration file it is possible to allow some repositories to use reserved exclusive and some to use other modes.



Distributed / Centralised

By default CVSNT clients are designed to work in a distributed mode. If you are working distributed then you will checkout an entire module to your local drive and work on it there.

If you do not have support with March Hare Software LLC then you will not receive any assistance with attempting to use CVSNT clients on a network share.

Checking out single files or only a portion of the source code to the local disk and the remainder of the files being available in a central shared area requires setting up a trigger on the server.

When a file is modified and committed to the CVSNT server then the relevant directory will be updated with the new *reference copy*. For example:

Location	*branch*
All Files and Directories	
/usr/opt/devel	TRUNK
/usr/opt/test_1_0	Test_1_0
/usr/opt/prod_1_0	Prod_1_0
/usr/opt/test_1_1	Test_1_1
/usr/opt/prod_1_1	Prod_1_1

Updating Centralised Copy on Windows Server

The CVSNT Server for Windows includes a plugin for build management. When the commit has completed the file `build_make.bat` in the CVSROOT will be executed with the module name and branch as parameters.

A simple DOS batch script can be used to run **cvs update** commands on the correct directory for each branch.

Updating Centralised Copy on Unix, Linux or Mac OS X Server

When the commit has completed the server will execute the shell scripts or perl scripts specified in the **postcommand** administration file.

Create a shell script or perl script to update the correct reference directory based on your own rules. Alternatively if you want to use a simpler interface then you can use the build management plugin and a `build_make.sh` script in CVSROOT.

Store Object Code

By default CVSNT will store all code, though some clients such as Eclipse have settings that can ignore object code overriding this.

If you decide to store object code then you will have to decide the point in time that the code should be checked in. Generally this is either the same time as the source is checked in (from the developer) or at build time.

Checking the object code in from the build environment ensures that the object code is compiled using a pre-defined environment not subject to developer preferences.

Keeping object code, from the developer

If you decide to keep the developers copy of the source code there is nothing more to do.

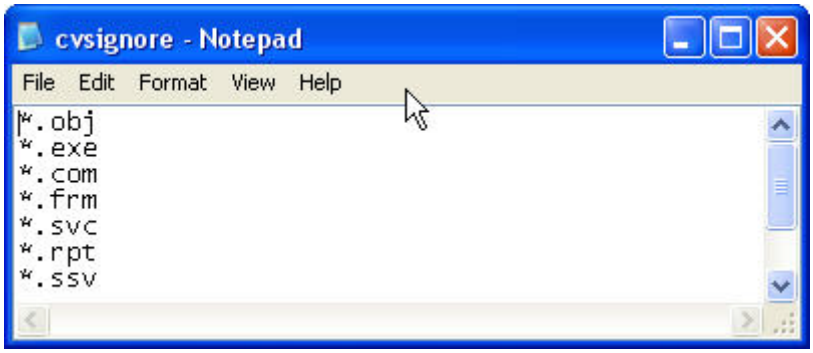
Keeping object code, from the build environment

The following needs to be configured:

- Restrict access permissions on the directories containing the object code so that developers cannot commit changes to these files.
- Structure the repository so that developers can easily commit source code and not object code (eg: /src /bin /lib).
- Use the build manager plugin to begin a build when changes are committed. At completion of the build a commit should run from the "reference area" which includes the build log file. Optionally e-mail the build log to the author.

Ignoring object code

To configure CVSNT to ignore object code set each "file extension" for the object code files in the *cvsignore* administration file, eg:



Promotion Model

Whilst promotion levels are logically managed on the server, the creation and manipulation of them occurs on the client.

It is usual to configure promotion branches so that developers cannot commit changes. This is usually reserved for QA staff.

Insulation and Communication

Insulation

The level of insulation between users is a factor of:

- Branching / which "reference area"
  For configuring this please see the section on setting up centralised areas using triggers.

  If two users share a branch and a "reference area" then when one completes their work then the other will immediately see the same changes. If they work on separate branches then they will have different "reference areas" and will not see each others changes until there is an explicit merge into their branch.
- Shared "working directories"
  If two users share a working directory on a network drive then they will have no insulation.

Communication

Automated communication is set up using the Email Integration or CVSMailer. See the earlier sections on Email integration and/or CVSMailer for more information.

Client Connection and Configuration

Each CVS client can connect to the server using any of the authentication techniques installed on the server. This section describes how to set up each of the most popular clients.

Note that setting up CVS Suite Server and Clients is NOT as simple as installing the software. The sections *CVS Architecture*, *Setting up CVS Suite Server* and *Server Administration* provide the necessary information to install and configure the CVS Suite system. If you do not follow these guides then severe performance problems can occur.

Client Time Synchronisation to CVS Server

For CVS Authentication to operate correctly, and to maximise performance of the CVS System CVS Clients must have their time synchronised to the CVS Server or the same time source as the CVS Server. If the CVS Client will use a network share or disk as a workspace / sandbox then the network share or disk should be synchronised to the same reliable time source as the CVS Server.

Client Time Synchronisation	
Windows	
Windows 2000	**w32tm -s**
Windows XP/2003	**w32tm /resync /rediscover**
For further instructions see: http://support.microsoft.com/?kbid=216734	
Unix	
HPUX	**/usr/sbin/ntpdate -u *servername***
Solaris	**/usr/sbin/ntpdate -u *servername***
Red Hat Enterprise Linux	**/usr/sbin/ntpdate -u *servername***
SuSE Enterprise Linux	**/usr/sbin/ntpdate -u *servername***
Unix systems must run this command as root and the command can be put in a cron job to run at regular intervals. Alternatively some systems may have ntp client daemons that can achieve this task.	

General Guide to Authentication

Storing passwords securely (CVSAGENT & Putty PAGEANT)

By default CVSNT client will store your password insecurely. There are several ways this can be done:

- cvs login
- saved CVSROOT with password (WinCVS, TortoiseCVS, CVS Suite Studio)
- ssh keys in unencrypted .ppk file

We strongly recommend that you never let CVSNT use its default password storage. Do not create connection profiles that include passwords in CVS Suite Studio, TortoiseCVS, WinCVS etc.

Safe passwords on Windows

On Windows your connection string (CVSROOT) usually not contain a password or username, eg:

```
:sspi:host:/myrepo
```

When using SSPI protocol in this way Windows passes the authentication token generated at login to the server and you are authenticated against the active directory securely with no password ever being stored.

If you do need to use a username and password, eg: if you are using pserver protocol or SSPI without an active directory, then you should run CVSAGENT when you login to your desktop (put it in startup items). Do NOT use a password in your connection string, use only the username, eg:

```
:pserver:myuser@host:/myrepo
```

CVS Suite and all the client programs will automatically store your password in memory. Note: the password prompt will come from CVSAGENT and not from WinCVS or TortoiseCVS as previously. You should use this for any password based logins, including pserver, sspi (no active directory) and ssh.

Safe keys on Windows with Putty SSH

If you are using SSH to connect to the CVSNT Server from Windows, then often you will use a public/private key pair.

If you supply an unencrypted private key file in the connection string CVSROOT then this is just as insecure as storing your password in plain text, eg:

```
:ssh;privatekey=d:\me.ppk:user@host:/repo
```

Instead, run Putty PAGEANT on login (eg: in startup items) and load your private key, either from a password protected file on the local or network drive, or from a secure USB drive (eg: with a fingerprint reader or other 2FA method). CVSNT client will automatically ask PAGEANT to supply they key when required.

PSERVER

Many people set up their clients to use *pserver* authentication because that is the easiest, or the most commonly documented. This manual does not describe the *pserver* connection method because it is insecure and unreliable. It is possible to set up Eclipse, the Visual Studio .NET Integration, CVS Suite Studio, WinCVS, TortoiseCVS and Mac CVS X to connect using SSPI, SSERVER or SSH secure protocols.

SSH

SSH is a recommended secure protocol for connecting to CVS Servers hosted on Unix or Mac OS X. When using SSH with Tortoise or WinCVS then it is best to also use CVSNT Password Agent (see the section *CVSNT Password Agent* below for more information).

SSPI

SSPI is a recommended secure protocol for connecting to CVS Servers hosted on Windows. When using SSPI it is not necessary to use the "cvs login" command to log in to the CVS Server. Each time a command is ran the current windows credentials are passed to the server and each command is authenticated separately.

If you accidentally use the *cvs login* command with the SSPI protocol then your password will be cached in the Windows registry and when you change your password then your client will stop authenticating successfully with the server..

Setting up CVS Suite Clients on Windows

Refer to the CVS Suite for Windows Installation Gudie for information about windows clients.

Setting up IntelliJ IDEA Client on Linux

IntelliJ IDEA is a popular IDE / Editor for use in Java programming environments and it is also the basis for Android Studio. The CVS Suite for Windows Installation Guide includes information on how to set up this client on windows, and the process is identical on Linux. Please refer to the CVS Suite Installation Gudie for Windows.

Setting up Eclipse Client on Windows

Eclipse is a popular IDE / Editor for use in Java programming environments, it is also the basis of the IBM WebSphere Application Developer. The CVS Suite for Windows Installation Guide includes information on how to set up this client on windows, and the process is identical on Linux. Please refer to the CVS Suite Installation Gudie for Windows

CVSNT Workflow

This section is intended to give the practical workflow when working with one of the popular methodologies discussed in the theory section. This section assumes that your repository, triggers (eg: keeping a read only "reference" copy of the source), integrations and access controls are already set up.

Workflow Definition

There are many techniques to define the workflow of your organisation in CVSNT, however these are not usually implemented in the client – but in the server using rules. There are many places on the server that rules can be defined depending on how they affect the versioning process. See the *Administration* section earlier in this book or *Implementing Methodoligies* below for more information about defining the workflow.

Version Numbers

The internal version numbers used by CVSNT do not specifically correlate to the version identifiers used by you or your organisation to identify individual file revisions or the revision of a group of files (such as a book or a software application).

Therefore it is important to distinguish between the version numbers used internally by CVSNT and external numbering schemes used by you. These external numbering schemes can and should be recorded in CVSNT as tags or labels.

Bug ID's

Most CVS commands now support a bug or defect identifier. Often this is much more useful to the end user than the physical or internal version number of the file. The bug identifier can be used to group or mark files, and the identifier can also be used to operate on collections of files already marked with that identifier, including merge and commit.

Edit

-B Mark this file with this identifier. A working revision can only have a single identifier.

Unedit

-b Unedit files with this identifier

Update (Merge)

-b When merging development branch (or Trunk) into the release branch it will take the lowest revision with that bug number and the highest revision with that bug number and merge the differences into the current branch. This is not a mergepoint.

Commit

-b Commit files with this identifier.

-B Commit files and mark the new version with this identifier.

Creating Branches

Creating Promotion levels and branches is not a part of the regular workflow – these are administration functions. Merging changes up (promoting) is also an administration function. Generally this section does not describe these activities since they are not generally performed from the graphical clients. These administration activities are best performed using the command line CVS tool. See the Administration section for more information.

Creating Sandboxes

Creating sandboxes is not a regular part of the users workflow. If you are using the Distributed model of CM then each user generally has a complete "checked out" copy of the source from the server in a sandbox. This sandbox usually resides on the local workstations hard drive, but may exist on a network share (see system requirements). This checkout is generally done once and rarely deleted (released) and started again.

If you are using the Centralised Reserved model then each user checks out and begins work on a file or set of files in one action.

CVS Suite includes the CVS Suite Studio which is the ideal tool for creating workspaces. TortoiseCVS and WinCVS can also be used to create workspaces.

Implementing Methodologies

In the first chapter we discussed four methodologies for versioning, and other decisions related to your management objectives and company culture.

This section provides a guide to how those choices map to use on the client.

Reserved / Unreserved

The majority of graphical CVS clients are designed to allow you to work in unreserved mode. Some clients allow you to also work in reserved mode to varying degrees.

To nominate a particular file type as one that should use "reserved" mode, define it in the *cvswrappers* administrative file with **-kc** (for cooperative reserved, or reserved for communication only) or **-kx** (for traditional reserved). More information on the administrative files can be found in the administration section on *cvswrappers*.

If you have a file type that should be reserved in some projects but not in others – create separate repositories for these two types of project.

There is nothing to set to enable unreserved mode.

Reserved

Working in reserved mode requires two steps in a "checkout process":

- Checkout (copy the file(s) to a work area)
  use the **cvs checkout** command.
- Reserved (allow this file to be worked on by the user)
  use the **cvs edit** command.

All reserved modes use watches

To enforce the need to "cvs edit" the file before changing it there must be a mechanism for CVSNT to know that the files should be checked out read-only. This is usually achieved by setting watches on. Note: if you are using –kx or -kc defined in the cvswrappers then this is not required.

To set watches on, checkout a module from the cvs repository and issue the command: *cvs watch on* to activate. In future all workspaces created for that module will be created read-only.

Reserved for Communication Only

This method employs the use of watches (as described above) but not –kc or –kx locking modes. In this way the files will be read only, but there is no hindrance to how many files and how many users of a file can access it at a time.

Reserved Co-operatively

Co-operative reserved uses the -kc *cvswrappers* option to ensure that files are not committed unless they are first editied. Since the watch mechanism merely makes files read-only, a smart user could simply remote the read-only attribute of the file using the unix *chmod* or DOS *attrib* commands.

If you are using co-operative edit enforced in cvswrappers then when that changed file is committed the CVSNT server checks that it has already received an edit notification. If it has not then the user cannot commit the changes until the *cvs edit* command is used. More than one user can still edit a file at the same time, but they are always informed of this (and can be alerted via e-mail with the Email plugin).

Reserved Exclusive

Exclusive reserved uses the -kx *cvswrappers* option to ensure that files are not committed unless they are first editied, and prevents more than one user at a time having edit access to the file. Since the watch mechanism merely makes files read-only, a smart user could simply remote the read-only attribute of the file using the unix *chmod* or DOS *attrib* commands.

If you are using exclusive edit enforced in cvswrappers then when that changed file is committed the CVSNT server checks that it has already received an edit notification. If it has not then the user cannot commit the changes until the *cvs edit* command is used. Since only one user at a time can have a file edited then it enforces the exclusivity of the lock.

Distributed / Centralised

The majority of graphical CVS clients are designed to allow you to work in distributed mode. Some clients allow you to also work in centralised mode to varying degrees. Implementing the Centralised model requires a read-only copy (shadow copy) of the workspace (or a branch of the workspace) available at all times. See the Section *Distributed / Centralised* in the *Administration* section.

Store Object Code

By default CVSNT will import and version all files in a directory. To exclude object code then set up the *cvsignore* administrative file.

Keeping object code, from the build environment

To keep the objects from the build environment only then change the access control list to allow developers to check out object code but not check it in. The build process should then "login" to cvs as an authorised user.

Promotion Model

See the section *Advanced Client Tasks* for more information about using the promotion model from the client.

Insulation and Communication

Insulation

Insulation is implemented by having separate workspaces for users who should be insulated form each others changes. If the projects are sufficiently different then they should also use different branches. See *CVS Suite Studio* for details on creating workspaces.

Communication

Communication is implemented by configuring the E-mail notification integration on the server. If two developers are both watching the same files then they will be alerted by e-mail when changes occur. Also see the headings *Find out about other peoples changes* and *All reserved modes use watches* in this section.

Unreserved Distributed Workflow

For a developer or an author working with source code files or documents controlled by CVSNT you will need to:

- Create a sandbox if one does not already exist
- Inform CVSNT that you have began working on a file (optional)
- Check your changes
- Synchronise your local file with the repository (update) or add a new local file to the repository
- Store your changes into the repository (commit)

The final stage also requires that you enter a description of your changes.

This workflow can be completed using any style of interface, we will look at Tortoise and WinCVS below.

Find out about other peoples changes

You can reserve a file for edit using CVSNT so that you are the only one who will have access to it while you are using it. However there are times that you do not want to lock people out – but need to know when changes are made. For example if one program depends on another.

It is possible to request that CVS notify you when other users begin work on a file or finish work on a file using the *notify* command.

Notify

For Notify to work your repository administrator must have already enabled e-mail notification via the *Notify* configuration file using a tool such as CVSMailer.

To enable watches you should use the CVSNT command line.

Use the *cvs watch on* command to enable watches and then *cvs watch add* each file that you want to be informed about.

Good comments

When using configuration management the kind of comments you enter and where you enter them becomes important.

If you do not have a tool such as CVS to automatically track who has changed what then you often add comments in the source code to indicate your initials and the change date. You may want to continue this practice however CVS will now automatically track this same information for you.

What is important are the comments you make when returning the file through version control. If in six months time you are trying to determine why a change was made a comment such as "fixes for September" is of little value – the date of the revision tells you that.

Good comments include:

- Bug number(s)
- References to other people / sponsors
- References to other documentation
- A succinct explanation of the purpose of the change

Here is an example of a good comment:

> *Bug 1234. Changes detailed in the functional spec h:\docs\fs23-1234.doc. Joe explained that the customer component should always display a warning before changes are written to the database. This is a first draft and probably needs some work.*

Here is a poor example of a comment:

> *Changes by Arthur. I made some changes to the WRITE trigger. Used askmess. A part of the September work.*

If you have a defect tracking system it is possible for the CVS repository administrator to have your comments at checkin automatically added to the defect tracking database.

Advanced client functions

This section is intended to give the administrators a quick overview as to how to use CVS clients to perform general administrative functions such as creating branches and doing promotes (merges).

It is recommended that administrators be familiar with using the command line tool for performing administrative functions.

Creating branches

To create a branch use the **cvs tag** command. Branches are designed to allow development on the same files by teams with different objectives to occur at the same time. The most common reason to create a branch will be in a software project when version 1 of the product is almost complete and version 2 is beginning work.

At this stage the development of version 2 will continue on the trunk and a branch will be created for finishing version 1. The test and release promotion levels will be branched of the version 1 development branch.

There are two ways to create a branch:

- With a sandbox of the "from" files (eg: creating a branch "from" the trunk you begin with a sandbox of the trunk. Creating a branch does not alter the working directory ad sandbox in any way.
- Without a sandbox. In this case the command takes two parameters for the "from" and new branch names. To ensure that the files you expect to be tagged are the ones that are tagged it is necessary to ensure that noone commits any changes to the repository. Eg: you decide to create dev1 branch and between you making the decision and running the rtag command someone commits some work which breaks the build.

Creating a branch from a sandbox

Use the following command on the sandbox:

cvs tag –b *branch-name*

after this command the local sandbox will not be altered in any way. If you want to make changes to this branch you will need to check out the newly created branch using the checkout command.

Creating a branch without a sandbox

You can create a branch without using a sandbox, however if another person commits changes while the branch is being created then it will not be immediately clear which versions of the files received the branch. This is because branch creation and commits are not atomic. This problem does not apply to checkouts (because a checkout is atomic).

To create a branch without a sandbox use:

cvs rtag –b *branch-name*

Creating Promotion Levels

CVS uses branches to implement a promotion level. The difference is simply how you use them. Because the promotion level is usually only accessed by a QA manager there are typically two additional steps to setting up a promotion level once the branch is created:

Change the default access so that developers cannot merge changes into the promotion level or commit changes onto it.

Change the default locking more to unreserved. This can be done using the cvs update command on a checked out sandbox of the new promotion level:

```
cvs update -k-x
cvs commit
```

Promoting

CVS uses branches to implement a promotion level. The difference is simply how you use them. There are two ways to promote a revision from the Trunk (or a branch) to a promotion level: Move and Merge. With the Move method you will not keep a history of the previous versions also moved to that promotion level, with the merge method CVSNT will keep track of the history using Mergepoints and these will be viewable using CVSGraph in TortoiseCVS or WinCVS.

Move Method

On the Trunk or branch you can push a version to a promotion branch (promote) using the CVS *tag* command with the options *-F -B -b tag*.

This is unsafe to do with a normal branch because you will lose the changes that have occurred on the branch between its creation and the promotion.

Merge Method

You will usually merge changes onto a promotion level by using bug identifiers. On a promotion branch versions are moved onto the promotion branch (promoted) using the CVS *update* command with the options *-B bug -j 1 -e newbug*. The *newbug* is a bug or task identifier for the tasks of promoting the bug to the new level.

Merging-in Branches

You can merge changes made on a branch into your working copy by giving the CVS command *update* with the *-j branchname* flag. With one *-j branchname* option it merges the changes made between the point where the branch was last merged and newest revision on that branch (into your working copy).

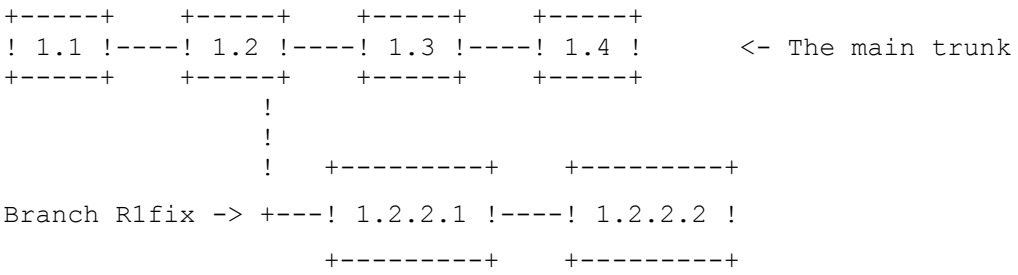
If you wish to revert to the older CVS behaviour of merging from the point the branch forked, specify the -b option.

If you are updating from an Unix CVS server of older CVSNT server that doesn't support merge points, then the merge will always be done from the branch point.

The -j stands for "join".

An example

Consider this revision tree:



The branch 1.2.2 has been given the tag (symbolic name) R1fix. The following example assumes that the module mod contains only one file, *m.c*.

```
$ cvs checkout mod               # Retrieve the latest revision, 1.4

$ cvs update -j R1fix m.c        # Merge all changes made on the branch,
                                 # i.e. the changes between revision 1.2
                                 # and 1.2.2.2, into your working copy
                                 # of the file.

$ cvs commit -m "Included R1fix" # Create revision 1.5.
```

A conflict can result from a merge operation. If that happens, you should resolve it before committing the new revision.

If your source files contain keywords, you might be getting more conflicts than strictly necessary.

The CVS command *checkout* also supports the *-j branchname* flag. The same effect as above could be achieved with this:

```
$ cvs checkout -j R1fix mod
$ cvs commit -m "Included R1fix"
```

It should be noted that the CVS command *update* with *-j tagname* will also work but may not produce the desired result.

Merging-in Branches using MergePoint

What is a mergepoint, and how does one use it?

Mergepoints help CVS find the common ancestor when trying to diff a file, which greatly reduces the effort required to merge in branches. It is automatically saved by CVS when you merge changes from one branch to another. Just make sure you commit after merging (before performing any other merges) and CVS will save the mergepoint field with the update.

Note that mergepoints are specific to CVSNT, and require that CVSNT is running on both client and server.

When you merge back and forth from dev branch to HEAD, it is a very simple operation--just specify the branch to merge from, and CVS takes care of the rest. If you read about merging in the regular CVS documentation, it looks like a big effort to *tag* before, *merge*, *commit*, retag... lots of tagging and remembering those tag names.

With mergepoint, merging is done simply by using:

cvs update -j branch-tag

Then you can correct any conflicts, test your integration, and commit without worrying about a lot of tagging operations.

You can see the mergepoint records in the output of the log command (look at rev 1.8, the last field before the comment):

Command Prompt

```
Microsoft® Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\temp\junk\test\>cvs log test.c
RCS file: /home/cvsroot/test/test.c,v
Working file: test.c
head: 1.8
branch:
locks: strict
access list:
symbolic names:
        grs-test1: 1.6.0.2
        R2: 1.3
        R1-Fix: 1.1.0.2
        R1: 1.1
keyword substitution: kv
total revisions: 12;    selected revisions: 12
description:
----------------------------
revision 1.8
date: 2003/11/06 21:53:13;  author: gstarret;  state: Exp;  lines: +0 -0;  kopt:
 kv;  commitid: 257f3faac2c9c824;  mergepoint: 1.6.2.1;
changed for some reason
----------------------------

C:\temp\junk\test\>
```

That mergepoint is in the record from the commit after I performed the merge.

Part IV – Appendix

What's new in CVS Suite 2009

CVS Suite 2009 is the fifth release of CVSNT by March Hare Software LLC. CVS Suite 2009 is a major upgrade to CVS Suite 2.5.03 and CVS Suite 2008 (CVS Suite 2008 was an incremental upgrade to CVS Suite 2.5.03 providing an easy migration for existing customers and is still supported).

CVS Suite delivers features that are essential when implementing an enterprise wide versioning solution. March Hare Software has worked closely with the original open source developers to strengthen CVSNT for the requirements of a commercial environment.

- Multi lingual filenames
- Synchronise Servers
- Server side external diff
- Plugins, including
  - ⇒ Comprehensive Audit
  - ⇒ Defect Tracking (Bugzilla, Jira and Mantis)
  - ⇒ Build Management
  - ⇒ E-mail Notification
  - ⇒ Shadow Update (Reference Copy Update)
- Support for OS X resource fork extensions
- Utf-8 Server
- Rendevous / Bonjour / Zero Config support
- New native file access system for Windows
- Multi threaded deadlock server
- Microsoft MSI Installer
- Update by bug number
- Definable Keywords
- General Improvements
- Versioned Rename
- Client Installer Package
- Visual Studio Integration (SCCI)
- CVS Suite Studio
- Release Manager

Why should you upgrade from CVS Suite 2.5.03 or 2008 to CVS Suite 2009:

- Improvded performance, particularly for update and short operations
- Distributed servers (caching servers, master servers)
- Improvded WAN performance with large files
- Support for Windows Server 2008 R2, Windows 7 Professional on both 32 and 64 bit platforms (as a 32 bit application)
- Client compatibility with CM Suite Server 2009
- Support for Bugzilla 3.2 and 3.4
- Support for Visual Studio 2008

Multi Lingual

CVSNT now handles files originating from any country with any possible encoding of the contents and filename.

Server Side External Diff

The new CVSNT command “cvs xdiff” allows an extendable diff system to be incorporated directly into CVSNT. In later releases March Hare will supply plugins for performing diff’s on the server for various popular file types.

Possible future xdiff plugins include:

- Word Documents
- XML Documents

Plugins

Plugins allow you to expand the functionality of the CVSNT server and integrate it with other business processes. The following plugins are provided with CVS Suite Server

- Defect Tracking and Change Sets (Bugzilla),
  Enhanced with private comments and perl script activating / e-mail notification
- Audit
  Enhanced with improved security
- Build Integration (make etc)
- Shadow Copy Update (Reference Copy Update)
- E-mail notification

Support for OS X resource fork extensions

This allows Mac CVS X to store not only the file but also the resource fork. If this option is specified in Mac CVS X then the files will not be usable on any other platform. Eg: if you commit a GIF file including the resource fork and then check the GIF file out to a web server (eg: Solaris Apache or Windows IIS) then the GIF file will be invalid and not viewable.

Utf-8 Server

To enable multi lingual filenames (eg: Korean, Thai, Chinese) enable the Utf-8 Server.

Rendevous, Bonjour, ZeroConf support

All repositories created in CVSNT are published using Rendevous, Bonjour, ZeroConf. This will allow future clients to connect to the CVSNT server without needing to specify the servername and root.

New native file access system for Windows

Reliability and performance have improved dramatically over CVSNT 2.0.x on Windows servers.

Multi threaded deadlock server

Reliability and performance have improved dramatically over CVSNT 2.0.x on Windows servers.

Update by bug number

Allows you to merge in changes from another branch based on Bug Number

Definable Keywords

You can enable or disable individual keywords, or change the formatting of existing keywords used in keyword expansion.

Client Installer Package

A single Windows Client package is now available which includes compatible versions of Tortoise, CVSNT, WinCVS and WinMerge.

CVS Suite Studio

Browse the available CVS Servers, checkout projects or create new ones simply using drag and drop as well as set access control and most of the functions you need to work with CVS Suite on a day to day basis.

Release Manager

If you have any client that should only “checkout” workspaces, such as a web server or a remote deployment, then install the release manager and run it to configure the location of each workspace.

Using the Release Manager it is also possible to mirror one remote workspace to other workspaces in the same location. For example a web server farm, a single web server would receive the latest web site from CVSNT and the other web servers would mirror it

Visual Studio Integration / Migrate from Visual SourceSafe

Use the Source Control features integrated into Microsoft Visual Studio 2005 to work with your CVS projects. This feature is ideal for people migrating from Visual SourceSafe. You can also migrate your version history from Visual SourceSafe to CVS Suite Server.

General Improvements

Improvements have been made to the following general areas:

- ACL's and editing of ACL's
- Configuration Control (specify allowable clients and version numbers of client software)
- Specify DOS line endings on checkout
- Confirmed reserved edit switches on "commit" and "editors" commands
- Improved crash dump handler

.

Versioned Rename / Move

Rename a file in a directory or move it between directories. Depending on the branch in use the file can be referred to using the original or new names.

```
cvs rename current-filename new-filename
```

Complex renames (across directories) should be avoided until EVS 3.1.01. If you do need to rename across directories then ensure that the 'destination' directory gets committed at the same time as the 'source' directory - otherwise the file can vanish completely.

What's coming in 2.8.02

CVS Suite 2.8.02

CVS Suite 2.8.02 is a standard upgrade from CVS Suite 2009. This allows us to provide a straightforward migration for existing CVS Suite 2009 customers. CVS Suite 2009 delivers improved stability and features including:

- Basic support for Bugzilla *time tracking*
- Suite Studio improvements, including *Send to* for ad-hoc release management
- Improved performance for Visual Studio Integration

CM Suite Server 3.1.02

CM Suite is the new versioning system by March Hare Software and builds on the enterprise versioning features of CVS Suite and CVSNT developed over the past three years.

CM Suite 3.1.02 will deliver the following features:

- Supports CVS, CVSNT, CVS Suite 2009, Team System and Subversion (SVN) clients
- View and modify your repository with web folders (eg: h:\trunk\projecta\spec.doc)
- Integrated defect trackign
- Integrated web browsing
- Promotion model support
- SQL relational database backend
- Rename support (files and directories)
- Fast tags
- Migrate from Subversion Server or CVS Suite Server 2.5.03/2008
- True multi-site SCM

Frequently Asked Questions

This is the list of frequently asked questions available on the March Hare Software CVSNT public internet site. Some of these may not be relevant to a customer of March Hare Software, but is reproduced here for reference.

What is the CVSNT Project

The CVSNT Project designs and delivers an advanced multiplatform version control system based on the industry standard CVS protocol. Protocol extensions and features are added to encourage wider use and effectiveness of Software Configuration Management.

The CVSNT Project was began by several developers in 1999 to resolve problems with the open source CVS system, in particular support for a CVS server on Windows, Case Insensitive Filenames, and usability issues.

What is CVS

CVS software is used to keep a track of changes to files stored on a computer

This is the function at the heart of all Source Code Management, Document Management and Configuration Management Systems.

CVSNT Professional and CVS Suite include additional tools to assist with deploying files to test and production environments, tracking who or what processes initiated the changes and much more.

What is the difference between CVS and CVSNT

CVSNT was started as a variation (or branch) of CVS in 1999 because patches contributed to CVS were not considered to be suitable for that project by those responsible for making such decisions at that time. These patches would add support for clients that work with the Microsoft Windows operating system(s) and in particular: case insensitive filenames.

CVSNT has continued to add features to the original CVS that support modern version control and configuration management best practice as well as maintain feature parity with CVS.

Today CVSNT is the most widely deployed version of CVS and is distributed with TortoiseCVS and WinCVS.

Why not merge CVSNT and CVS?

The short answer is that we have - the merged version is called CVSNT. CVSNT has all of the features of CVS 1.11 and CVSNT. For a much more complex answer read on.

Merging the projects and merging the repositories for the two projects has been discussed several times with different members of the CVS project team. The problem appears to be that the CVS project has an established charter that dictates that only some kinds of changes are suitable for inclusion in that project, see here and also here. This is considered the CVS Project Charter.

The organisation of the CVSNT project has been designed to not hinder development with such a charter. Not having such a limiting charter encourages creativity and adaptation. Specifically since the software is designed to manage changes we feel that the software itself should not hinder change. The CVSNT project accepts change and changes as a part of the basic concept of version management, including changes to expand the supported methodologies. This has allowed the CVSNT project to add features and make architectural changes that the CVS project has not and probably will not.

This difference in philosophy as to how to accept changes is the core problem in merging the project teams. Both teams feel that each philosophy produces a result that users of each product want. Generally that is:

- CVS users want a free version control tool that implements the unreserved distributed methodology for unix, linux and cygwin.
- CVSNT users want a free version control tool that implements latest "best practice" including unreserved distributed as well as reserved and centralised for windows, mac, unix, linux and os/400.

What sort of patches prevent the CVSNT project and the CVS project from merging?

Merging the projects today would be difficult because the development teams have different ideologies about how patches should be accepted.

However leaving that aside, a recent conversation between some of the developers from each team revealed that these kinds of patches would cause the most problems:

- Structural Changes.

  CVSNT has changed the structure of some of the source code to make it possible to reduce security risks through better memory management, and allow the dynamic removal of insecure protocols such as pserver.
- Support for alternative ideologies to the core "Unreserved Distributed" model
- Features which are not core version control fucntions but could be seen as extensions to version control
- Features that some porting developers would find it difficult to support on their platforms.

What Information should I provide with a bug report?

To receive a response from support you must provide the following information:

- Your customer name and/or order id
- Results of running cvsdiag on the server and also on the client.
- CVSNT server version number (eg: CVSNT 2.5.02.2048)
- Name and version of cvs client (eg: Eclispe 3.1)
- Name and version of Server operating system (eg: Windows XP SP2, or Red Hat Linux ES v4)
- Name and version of Client operating system (eg: Windows NT 4 SP6, or Solaris 9)
- The CVSROOT used by client (eg: :pserver:host:/cvsrepo)
- Screenshot or exact wording of error message
- Your e-mail address

If possible please also provide:

- Affected files (eg: the RCS file from the repository)
- Debug trace of CVSNT command which causes problem: Cvs –ttt
- If you are using the :ext: protocol, the extnt.ini from the CVSNT Installation Directory on the Windows client
- Dump of the client environment variables. On Windows type set > mh.log and send the file mh.log

Can I search for questions other people have asked and the answers they got?

Yes you can. Search for an answer using the mailing list archive at http://www.cvsnt.org/pipermail/cvsnt/. The easiest way is with Google Site Search. Search an answer in the bug database at http://customer.march-hare.com/webtools/bugzilla/tt.htm.

Is there a list of current outstanding bugs?

The list of current bugs is availabe in the trouble ticket database: http://customer.march-hare.com/webtools/bugzilla/tt.htm.

Where can I download stable releases of CVSNT?

CVS Suite and CVS Professional Support customers should use the "customer area" link at the bottom of this page to login and download the latest customer release of CVSNT.

The download page for stable releases of the free CVSNT is: http://march-hare.com/cvspro.

At the far right of the page is a "download box", choose the download you want, choose an export server and accept the license terms by pressing the button.

Is CVSNT available for VMS / OpenVMS ?

No CVSNT does not support OpenVMS. You can use a java or older CVS client for OpenVMS to connect to CVSNT server. Server side features of CVSNT such as Access Control Lists and Audit are available from all clients.

Who are March Hare Software?

March Hare Software Ltd (UK) is the software development arm of March Hare Pty Ltd (Australia). March Hare develop enterprise database software for companies worldwide and use the CVSNT system internally for all projects.

March Hare began in Sydney Australia in 1998 as a loose collective of expert database programmers and has grown into a worldwide firm based on our expertise in designing and building enterprise IT solutions.

March Hare Software development lab is based in Manchester UK, and iSeries development in Amsterdam, The Netherlands.

Why do all links to CVSNT go to March Hare Software?

March Hare Software are the primary contributors to the CVSNT project. We co-ordinate the testing and release process as well as manage the source tree.

Where can I find March Hare Software annual reports?

March Hare Software Ltd (UK) and March Hare Pty Ltd (Australia) are not publicly traded companies and therefore do not publish annual reports.

Who are the Directors and Shareholders of March Hare?

March Hare Software Limited (UK) and March Hare Pty Ltd (Australia) are not public companies and the shareholder registry is not a matter of public record. The existing shareholders wish to remain anonymous and no additional equity is currently being sought. The share registry is closed.

The current board of directors for each company can be found through the relavent legislative authority for UK and Australian Company Registrations.

Where is CVSNT for iSeries?

CVSNT for iSeries consists of a specially adapted version of the CVS client and a front end program which allows CVS to version Source Physical files such as RPG and COBOL source code.

March Hare Software license the iSeries front end separately to the client and it is not available without a support contract.

What priority do March Hare apply to support requests.

March Hare Software apply priority to support requests based on severity and service level in that order. i.e.: Level 3 support customers have their problems looked at before Level 1 (CVS Suite) customers for problems of the same severity.

The severity levels are:

- Repository corruption
- Server Security
- Workspace corruption
- Server crash (core dump)
- Client crash (core dump)
- General Faults / Installation
- CVSNT client compatibility
- Non CVSNT client compatibility
- Enhancements
- Platform support

Can I move my version history from Visual SourceSafe, PVCS, ClearCase or Perforce to CVSNT?

March Hare Software can provide consulting to migrate version history stored in any format to the CVSNT repository. Popular formats such as Visual SourceSafe do not require much time or effort to convert. March Hare Software can also provide training for system administrators as well as general users on the differences between the old and the new system.

Visual Source Safe, PVCS, ClearCase and Perforce are each excellent version control solutions each with many features, and each with unique features.

CVSNT is not an emulator designed to replace any of them. However these tools solve problems generally defined by the phrases version control and configuration management, and those are the same problems that CVSNT is designed to solve. Therefore in some situations it may be possible to switch one for the other and have very little impact on your business.

How many people use CVSNT?

We do not know how many people use CVSNT, however as at 25th August 2005 more than 1.4 million copies of the stable build of CVSNT had been downloaded since March Hare Software partnered with the CVSNT project in August 2004.

Gartner research has published statistics in the past on Software Configuration Management Market Share based on license revenue, however we have been unable to find any research done based on use in the field.

Most other industry analysts use this same approach (eg: "Worldwide Software Configuration Management Tools 2004 Vendor Shares", #33588, June 2005 by IDC). This research appears designed to assist vendors and investors in determining how much money is available in the marketplace for these tools and to which vendor the majority of the money is going. The research does not benefit the person who is choosing a tool based on popularity (in use by many people on a day to day basis) or effectiveness (has achieved productivity and reliability gains that software configuration management is designed to). In particular the studies:

- give a natural preference to software which is costly over software which is economical
- does not account at all for software implemented without a license cost
- provide a market share benefit to companies which sell software which is then later replaced due to implementation difficulties.

Gartner research have given us a quote for performing some study based on use in the field, however no budget has been allocated to fund this in 2005/2006. Furthermore it would not be possible to publish the results of this study since Gartner require a signed non-disclosure agreement from any person who views it or information distilled from it.

Built / executable copies of CVSNT are included in TortoiseCVS and WinCVS and also downloaded from the main March Hare web site. From these sites roughly 30,000 copies per week are downloaded. This does not include copies of CVSNT which are built from source in the "traditional open source" way or copies of any version which is in development or test (unstable).

The best research we have is incidental and that indicates that CVSNT is the most widely adopted versioning tool in use in the Software Development industry today.

Default Binary File Types

With no additional configuration CVSNT will recognise files with these extensions has being of type binary. This list can be generated with the command:

cvs info cvswrappers

.a	.ai	.avi
.bin	.bmp	.bz2
.chm	.class	.cur
.dll	.doc	.docx
.docm	.pptx	.pptm
.xlsx	.xlsm	.odt
.ods	.odp	.odb
.odg	.odf	.mpp
.dvi	.exe	.gif
.gz	.hqx	.ico
.ilk	.indd	.iso
.lib	.jar	.jpg
.jpeg	.lnk	.mpg
.mpeg	.mov	.mp3
.ncb	.o	.ogg
.obj	.pdb	.pdf
.png	.ppt	.psd
.res	.rpm	.sit
.so	.tar	.tga
.tgz	.tif	.tiff
.ttf	.wav	.wmv
.xls	.zip	.Z

Additional file types can be defined as binary by using the *cvswrappers* administrative file.

Valid codepages

If the CVSNT server is running in Unicode mode and is used with a CVSNT client then it will automatically translate between the following codepages (some codepages may not be available on all platoforms). Multiple codepages listed on one bullet are synonyms:

- ANSI_X3.4-1968 ANSI_X3.4-1986 ASCII CP367 IBM367 ISO-IR-6 ISO646-US ISO_646.IRV:1991 US US-ASCII CSASCII
- UTF-8
- ISO-10646-UCS-2 UCS-2 CSUNICODE
- UCS-2BE UNICODE-1-1 UNICODEBIG CSUNICODE11
- UCS-2LE UNICODELITTLE
- ISO-10646-UCS-4 UCS-4 CSUCS4
- UCS-4BE
- UCS-4LE
- UTF-16
- UTF-16BE
- UTF-16LE
- UTF-32
- UTF-32BE
- UTF-32LE
- UNICODE-1-1-UTF-7 UTF-7 CSUNICODE11UTF7
- UCS-2-INTERNAL
- UCS-2-SWAPPED
- UCS-4-INTERNAL
- UCS-4-SWAPPED
- C99
- JAVA
- CP819 IBM819 ISO-8859-1 ISO-IR-100 ISO8859-1 ISO_8859-1 ISO_8859-1:1987 L1 LATIN1 CSISOLATIN1
- ISO-8859-2 ISO-IR-101 ISO8859-2 ISO_8859-2 ISO_8859-2:1987 L2 LATIN2 CSISOLATIN2
- ISO-8859-3 ISO-IR-109 ISO8859-3 ISO_8859-3 ISO_8859-3:1988 L3 LATIN3 CSISOLATIN3
- ISO-8859-4 ISO-IR-110 ISO8859-4 ISO_8859-4 ISO_8859-4:1988 L4 LATIN4 CSISOLATIN4
- CYRILLIC ISO-8859-5 ISO-IR-144 ISO8859-5 ISO_8859-5 ISO_8859-5:1988 CSISOLATINCYRILLIC
- ARABIC ASMO-708 ECMA-114 ISO-8859-6 ISO-IR-127 ISO8859-6 ISO_8859-6 ISO_8859-6:1987 CSISOLATINARABIC
- ECMA-118 ELOT_928 GREEK GREEK8 ISO-8859-7 ISO-IR-126 ISO8859-7 ISO_8859-7 ISO_8859-7:1987 CSISOLATINGREEK
- HEBREW ISO-8859-8 ISO-IR-138 ISO8859-8 ISO_8859-8 ISO_8859-8:1988 CSISOLATINHEBREW
- ISO-8859-9 ISO-IR-148 ISO8859-9 ISO_8859-9 ISO_8859-9:1989 L5 LATIN5 CSISOLATIN5
- ISO-8859-10 ISO-IR-157 ISO8859-10 ISO_8859-10 ISO_8859-10:1992 L6 LATIN6 CSISOLATIN6
- ISO-8859-13 ISO-IR-179 ISO8859-13 ISO_8859-13 L7 LATIN7
- ISO-8859-14 ISO-CELTIC ISO-IR-199 ISO8859-14 ISO_8859-14 ISO_8859-14:1998 L8 LATIN8
- ISO-8859-15 ISO-IR-203 ISO8859-15 ISO_8859-15 ISO_8859-15:1998 LATIN-9
- ISO-8859-16 ISO-IR-226 ISO8859-16 ISO_8859-16 ISO_8859-16:2001 L10 LATIN10
- KOI8-R CSKOI8R
- KOI8-U
- KOI8-RU
- CP1250 MS-EE WINDOWS-1250
- CP1251 MS-CYRL WINDOWS-1251
- CP1252 MS-ANSI WINDOWS-1252
- CP1253 MS-GREEK WINDOWS-1253
- CP1254 MS-TURK WINDOWS-1254
- CP1255 MS-HEBR WINDOWS-1255
- CP1256 MS-ARAB WINDOWS-1256
- CP1257 WINBALTRIM WINDOWS-1257
- CP1258 WINDOWS-1258
- 850 CP850 IBM850 CSPC850MULTILINGUAL
- 862 CP862 IBM862 CSPC862LATINHEBREW
- 866 CP866 IBM866 CSIBM866
- MAC MACINTOSH MACROMAN CSMACINTOSH
- MACCENTRALEUROPE
- MACICELAND
- MACCROATIAN
- MACROMANIA
- MACCYRILLIC
- MACUKRAINE
- MACGREEK
- MACTURKISH
- MACHEBREW
- MACARABIC
- MACTHAI
- HP-ROMAN8 R8 ROMAN8 CSHPROMAN8
- NEXTSTEP
- ARMSCII-8
- GEORGIAN-ACADEMY
- GEORGIAN-PS
- KOI8-T
- MULELAO-1
- CP1133 IBM-CP1133
- ISO-IR-166 TIS-620 TIS620 TIS620-0 TIS620.2529-1 TIS620.2533-0 TIS620.2533-1
- CP874 WINDOWS-874
- VISCII VISCII1.1-1 CSVISCII
- TCVN TCVN-5712 TCVN5712-1 TCVN5712-1:1993
- ISO-IR-14 ISO646-JP JIS_C6220-1969-RO JP CSISO14JISC6220RO
- JISX0201-1976 JIS_X0201 X0201 CSHALFWIDTHKATAKANA
- ISO-IR-87 JIS0208 JIS_C6226-1983 JIS_X0208 JIS_X0208-1983 JIS_X0208-1990 X0208 CSISO87JISX0208
- ISO-IR-159 JIS_X0212 JIS_X0212-1990 JIS_X0212.1990-0 X0212 CSISO159JISX02121990
- CN GB_1988-80 ISO-IR-57 ISO646-CN CSISO57GB1988
- CHINESE GB_2312-80 ISO-IR-58 CSISO58GB231280
- CN-GB-ISOIR165 ISO-IR-165
- ISO-IR-149 KOREAN KSC_5601 KS_C_5601-1987 KS_C_5601-1989 CSKSC56011987
- EUC-JP EUCJP EXTENDED_UNIX_CODE_PACKED_FORMAT_FOR_JAPANESE CSEUCPKDFMTJAPANESE
- MS_KANJI SHIFT-JIS SHIFT_JIS SJIS CSSHIFTJIS
- CP932
- ISO-2022-JP CSISO2022JP
- ISO-2022-JP-1
- ISO-2022-JP-2 CSISO2022JP2
- CN-GB EUC-CN EUCCN GB2312 CSGB2312
- CP936 GBK MS936 WINDOWS-936
- GB18030
- ISO-2022-CN CSISO2022CN
- ISO-2022-CN-EXT
- HZ HZ-GB-2312
- EUC-TW EUCTW CSEUCTW
- BIG-5 BIG-FIVE BIG5 BIGFIVE CN-BIG5 CSBIG5
- CP950
- BIG5-HKSCS BIG5HKSCS
- EUC-KR EUCKR CSEUCKR
- CP949 UHC
- CP1361 JOHAB
- ISO-2022-KR CSISO2022KR
- 437 CP437 IBM437 CSPC8CODEPAGE437
- CP737
- CP775 IBM775 CSPC775BALTIC
- 852 CP852 IBM852 CSPCP852
- CP853
- 855 CP855 IBM855 CSIBM855
- 857 CP857 IBM857 CSIBM857
- CP858
- 860 CP860 IBM860 CSIBM860
- 861 CP-IS CP861 IBM861 CSIBM861
- 863 CP863 IBM863 CSIBM863
- CP864 IBM864 CSIBM864
- 865 CP865 IBM865 CSIBM865
- 869 CP-GR CP869 IBM869 CSIBM869
- CP1125

Codepages supported on each system are defined in the iconv library.

Valid keyword expansion options

You can use the cvsnt command line to list the supported keyword expansion options using this command:

```
cvs up -k?
```

Valid expansion modes include:

```
-kkv     Generate keywords using the default form.
-kkvl    Like -kkv, except locker's name inserted.
-kk Generate only keyword names in keyword strings.
-kv Generate only keyword values in keyword strings.
-ko Generate the old keyword string (no changes from checked in file).
-kb Generate binary file unmodified (merges not allowed) (RCS 5.7).
```

SQL Scripts for Audit Integration

These scripts are required to create the database schema for the audit integration.

MySQL

```
Create Table SchemaVersion (Version Integer);
Insert Into SchemaVersion (Version) Values (3);
Create Table SessionLog (Id Integer Auto_Increment Primary Key Not Null,
  Command varchar(32),
  StartTime datetime,
  EndTime datetime,
  Hostname varchar(255),
  Username varchar(255),
  SessionId varchar(32),
  VirtRepos varchar(255),
  PhysRepos varchar(255),
  Client varchar(64),
  FinalReturnCode Integer);
Create Table CommitLog (Id Integer Auto_Increment Primary Key Not Null,
  SessionId Integer,
  Directory varchar(255),
  Message text,
  Type char(1),
  Filename varchar(255),
  Tag varchar(64),
  BugId varchar(64),
  OldRev varchar(64),
  NewRev varchar(64),
  Added Integer,
  Removed Integer,
  Diff text);
Create Index Commit_SessionId On CommitLog(SessionId);
Create Table HistoryLog (Id Integer Auto_Increment Primary Key Not Null,
  SessionId Integer,
  Type char(1),
  WorkDir varchar(255),
  Revs varchar(64),
  Name varchar(255),
  BugId varchar(64),
  Message text);
Create Index History_SessionId on HistoryLog(SessionId);
Create Table TagLog (Id Integer Auto_Increment Primary Key Not Null,
  SessionId Integer,
  Directory varchar(255),
  Filename varchar(255),
  Tag varchar(64),
  Revision varchar(64),
  Message text,
  Action varchar(32),
  Type char(1));
Create Index Tag_SessionId on TagLog(SessionId);
```

SQLite 3 (not supported or recommended in CVS Suite 2009)

```
Create Table SchemaVersion (Version Integer);

Insert Into SchemaVersion (Version) Values (3);

Create Table SessionLog (Id Integer Primary Key Not Null,
  Command nvarchar(32),
  StartTime datetime,
  EndTime datetime,
  Hostname nvarchar(256),
  Username nvarchar(256),
  SessionId nvarchar(32),
  VirtRepos nvarchar(256),
  PhysRepos nvarchar(256),
  Client nvarchar(64),
  FinalReturnCode Integer);

Create Table CommitLog (Id Integer Primary Key Not Null,
  SessionId Integer,
  Directory nvarchar(256),
  Message text,
  Type char(1),
  Filename nvarchar(256),
  Tag nvarchar(64),
  BugId nvarchar(64),
  OldRev nvarchar(64),
  NewRev nvarchar(64),
  Added Integer,
  Removed Integer,
  Diff text);

Create Index Commit_SessionId On CommitLog(SessionId);

Create Table HistoryLog (Id Integer Primary Key Not Null,
  SessionId Integer,
  Type char(1),
  WorkDir nvarchar(256),
  Revs nvarchar(64),
  Name nvarchar(256),
  BugId nvarchar(64),
  Message text);

Create Index History_SessionId on HistoryLog(SessionId);

Create Table TagLog (Id Integer Primary Key Not Null,
  SessionId Integer,
  Directory nvarchar(256),
  Filename nvarchar(256),
  Tag nvarchar(64),
  Revision nvarchar(64),
  Message text,
  Action nvarchar(32),
  Type char(1));

Create Index Tag_SessionId on TagLog(SessionId);
```

MSSQL

```
Create Table SchemaVersion (Version Integer);
Insert Into SchemaVersion (Version) Values (3);
Create Table SessionLog (Id Integer Identity Primary Key Not Null,
  Command nvarchar(32),
  StartTime Datetime,
  EndTime Datetime,
  Hostname nvarchar(256),
  Username nvarchar(256),
  SessionId nvarchar(32),
  VirtRepos nvarchar(256),
  PhysRepos nvarchar(256),
  Client nvarchar(64),
  FinalReturnCode Integer);
Create Table CommitLog (Id Integer Identity Primary Key Not Null,
  SessionId Integer,
  Directory nvarchar(256),
  Message text,
  Type char(1),
  Filename nvarchar(256),
  Tag nvarchar(64),
  BugId nvarchar(64),
  OldRev nvarchar(64),
  NewRev nvarchar(64),
  Added Integer,
  Removed Integer,
  Diff text);
Create Index Commit_SessionId On CommitLog(SessionId);
Create Table HistoryLog (Id Integer Identity Primary Key Not Null,
  SessionId Integer,
  Type char(1),
  WorkDir nvarchar(256),
  Revs nvarchar(64),
  Name nvarchar(256),
  BugId nvarchar(64),
  Message text);
Create Index History_SessionId on HistoryLog(SessionId);
Create Table TagLog (Id Integer Identity Primary Key Not Null,
  SessionId Integer,
  Directory nvarchar(256),
  Filename nvarchar(256),
  Tag nvarchar(64),
  Revision nvarchar(64),
  Message text,
  Action nvarchar(32),
  Type char(1));
Create Index Tag_SessionId on TagLog(SessionId);
```

Oracle 9/10

```
Create Table SchemaVersion (Version Number);
Insert Into SchemaVersion (Version) Values (3);
Grant Select On SchemaVersion To Public;
Create Table SessionLog (Id Number Not Null,
  Command nvarchar2(32),
  StartTime Timestamp,
  EndTime Timestamp,
  Hostname nvarchar2(256),
  Username nvarchar2(256),
  SessionId varchar(32),
  VirtRepos nvarchar2(256),
  PhysRepos nvarchar2(256),
  Client varchar(64),
  FinalReturnCode Number,
  CONSTRAINT SessionLog PRIMARY KEY(Id));
Grant Select, Insert On SessionLog To Public;
Create Sequence SessionLog_sequence
  Start with 1
  Increment by 1
  Nomaxvalue
  Nocache
  Noorder;
Grant Select On SessionLog_sequence To Public;
Create or Replace Trigger SessionLog_trigger
  Before insert on SessionLog
    For each row
    Begin
      Select SessionLog_sequence.nextval into :new.Id from dual; End;;
Create Table CommitLog (Id Number Not Null,
  SessionId Number,
  Directory nvarchar2(256),
  Message nclob,
  Type char(1),
  Filename nvarchar2(256),
  Tag varchar(64),
  BugId varchar(64),
  OldRev varchar(64),
  NewRev varchar(64),
  Added Number,
  Removed Number,
  Diff nclob,
  CONSTRAINT CommitLog PRIMARY KEY(Id));
Grant Select, Insert On CommitLog To Public;
Create Sequence CommitLog_sequence
  Start with 1
  Increment by 1
  Nomaxvalue
  Nocache
  Noorder;
Grant Select On CommitLog_sequence To Public;
```

```
Create Or Replace Trigger CommitLog_trigger
  Before insert on CommitLog
    For each row
    Begin
      Select CommitLog_sequence.nextval into :new.Id from dual; End;;
Create Index Commit_SessionId On CommitLog(SessionId);
Create Table HistoryLog (Id Number  Not Null,
  SessionId Number,
  Type char(1),
  WorkDir nvarchar2(256),
  Revs varchar(64),
  Name nvarchar2(256),
  BugId varchar(64),
  Message nclob,
  CONSTRAINT HistoryLog PRIMARY KEY(Id));
Grant Select, Insert On HistoryLog To Public;
Create Sequence HistoryLog_sequence
  Start with 1
  Increment by 1
  Nomaxvalue
  Nocache
  Noorder;
Grant Select On HistoryLog_sequence To Public;
Create Or Replace Trigger HistoryLog_trigger
  Before insert on HistoryLog
    For each row
    Begin
      Select HistoryLog_sequence.nextval into :new.Id from dual; End;;
Create Index History_SessionId on HistoryLog(SessionId);
Create Table TagLog (Id Number  Not Null,
  SessionId Number,
  Directory nvarchar2(256),
  Filename nvarchar2(256),
  Tag varchar(64),
  Revision varchar(64),
  Message nclob,
  Action varchar(32),
  Type char(1),
  CONSTRAINT TagLog PRIMARY KEY(Id));
Grant Select, Insert On TagLog To Public;
Create Sequence TagLog_sequence
  Start with 1
  Increment by 1
  Nomaxvalue
  Nocache
  Noorder;
Grant Select On TagLog_sequence To Public;
Create Trigger TagLog_trigger
  Before insert on TagLog
    For each row
    Begin
      Select TagLog_sequence.nextval into :new.Id from dual; End;;
Create Index Tag_SessionId on TagLog(SessionId);
```

All environment variables which affect CVSNT

This is a complete list of all environment variables that affect CVSNT.

$CVSIGNORE

A whitespace-separated list of file name patterns that CVSNT should ignore.

$CVSWRAPPERS

A whitespace-separated list of file name patterns that CVSNT should treat as wrappers.

$CVSREAD

If this is set, checkout and update will try hard to make the files in your working directory read-only. When this is not set, the default behaviour is to permit modification of your working files.

$CVSUMASK

Controls permissions of files in the repository.

$CVSROOT

Should contain the full pathname to the root of the cvsnt source repository (where the RCS files are kept). This information must be available to CVSNT for most commands to execute; if $CVSROOT is not set, or if you wish to override it for one invocation, you can supply it on the command line: `cvs -d cvsroot cvs_command...` Once you have checked out a working directory, cvsnt stores the appropriate root (in the file `CVS/Root`), so normally you only need to worry about this when initially checking out a working directory.

$EDITOR, $CVSEDITOR, $VISUAL,

Specifies the program to use for recording log messages during commit. $CVSEDITOR overrides $EDITOR.

$PATH

If $rcsBIN is not set, and no path is compiled into CVSNT, it will use $PATH to try to find all programs it uses.

$HOME

$HOMEPATH

$HOMEDRIVE

Used to locate the directory where the `.cvsrc` file, and other such files, are searched. On Unix, CVSNT just checks for HOME. On Windows NT, the system will set HOMEDRIVE, for example to d: and HOMEPATH, for example to \joe. On Windows 95, you'll probably need to set HOMEDRIVE and HOMEPATH yourself.

$CVS_EXT

$CVS_RSH

Specifies the external program which CVSNT connects with, when :ext: access method is specified. This replaces the CVS_RSH environment used in older implementations of CVS.

The CVS_EXT variable parsed as a formatting string specifying the command to pass to invoke the remote server. The default string is: `ssh -l %u %h` The %u parameter is replaced with the username specified in the CVSROOT (or the current username if none is specified) and the %h parameter is replaced with the hostname specified in the CVSROOT.

The CVS_EXT string has the string ' cvs server' appended to it, and this is then passed to the command processor for execution.

$CVS_CLIENT_PORT

Used in client-server mode when accessing the server via Kerberos, GSSAPI, or CVSNT's password authentication if the port is not specified in $CVSROOT.

$CVS_CLIENT_LOG

Used to debug the client server communication. Set this environment variable to the name of a log file. A ".in" and a ".out" log file will be created. Eg: CVS_CLIENT_LOG=log20050622

$CVS_SERVER

Usually CVS clients use the variable CVS_SERVER to specify the location of the server command, eg CVS_SERVER=/Apps/cvs/bin/cvsnt

$CVS_SERVER_LOG

Used for debugging only in client-server mode. If set, everything sent to the server is logged into $CVS_SERVER_LOG.in and everything sent from the server is logged into $CVS_SERVER_LOG.out.

$CVS_SERVER_SLEEP

Used only for debugging the server side in client-server mode. If set, delays the start of the server child process the specified amount of seconds so that you can attach to it with a debugger.

$CVS_DIR

Used by the client to find the *deadlock server* when automatically executing it. If not defined the client looks in the system path.

$CVSLIB

Location of the libraries and protocol DLLs used by CVSNT. Not used on Win32.

$CVSCONF

Location of the global configuration settings file. Not used on Win32.

$COMSPEC

Used under DOS/Windows and OS/2 only. It specifies the name of the command interpreter and defaults to cmd.exe.

$TMPDIR

$TMP

$TEMP

Directory in which temporary files are located. The cvsnt server uses TMPDIR. Some parts of CVSNT will always use `/tmp` (via the tmpnam function provided by the system).

On Windows NT, TMP is used (via the _tempnam function provided by the system).

The patch program which is used by the CVSNT client uses TMPDIR, and if it is not set, uses `/tmp` (at least with GNU patch 2.1). Note that if your server and client are both running CVSNT 1.9.10 or later, CVSNT will not invoke an external patch program.

Part V – Reference

CVSNT Command Reference

Overview

This appendix describes the overall structure of CVSNT commands, and describes some commands in detail.

Some graphical front ends for CVSNT provide a user friendly way to execute some of these commands and options. This reference explains all options available using the command line based CVSNT.

Each command is made up of several components, the first sections deal with the structure of the commands and global options, then each following section describes each command in detail.

Overall structure of CVS commands

The overall format of all cvsnt commands is:

```
cvs [ cvs_options ] cvs_command [ command_options ] [ command_args ]
```

cvs

The name of the cvsnt program.

cvs_options

Some options that affect all sub-commands of cvsnt. These are described below.

cvs_command

One of several different sub-commands. Some of the commands have aliases that can be used instead; those aliases are noted in the reference manual for that command. There are only two situations where you may omit cvs_command: cvs -H elicits a list of available commands, and cvs -v displays version information on cvsnt itself.

command_options

Options that are specific for the command.

command_args

Arguments to the commands.

There is unfortunately some confusion between cvs_options and command_options. -l, when given as a cvs_option, only affects some of the commands. When it is given as a command_option is has a different meaning, and is accepted by more commands. In other words, do not take the above categorization too seriously. Look at the documentation instead.

CVS's exit status

cvsnt can indicate to the calling environment whether it succeeded or failed by setting its exit status. The exact way of testing the exit status will vary from one operating system to another. For example in a unix shell script the $? variable will be 0 if the last command returned a successful exit status, or greater than 0 if the exit status indicated failure.

If cvsnt is successful, it returns a successful status; if there is an error, it prints an error message and returns a failure status. The one exception to this is the cvs diff command. It will return a successful status if it found no differences, or a failure status if there were differences or if there was an error. Because this behavior provides no good way to detect errors, in the future it is possible that cvs diff will be changed to behave like the other cvsnt commands.

Default options and the ~/.cvsrc and CVSROOT/cvsrc files

There are some command_options that are used so often that you might have set up an alias or some other means to make sure you always specify that option. One example (the one that drove the implementation of the .cvsrc support, actually) is that many people find the default output of the diff command to be very hard to read, and that either context diffs or unidiffs are much easier to understand.

The ~/.cvsrc file is a way that you can add default options to cvs_commands within cvs, instead of relying on aliases or other shell scripts.

The format of the ~/.cvsrc file is simple. The file is searched for a line that begins with the same name as the cvs_command being executed. If a match is found, then the remainder of the line is split up (at whitespace characters) into separate options and added to the command arguments *before* any options from the command line.

If a command has two names (e.g., checkout and co), the official name, not necessarily the one used on the command line, will be used to match against the file. So if this is the contents of the user's ~/.cvsrc file:

```
log -N
diff -u
update -P
checkout -P
```

the command cvs checkout foo would have the -P option added to the arguments, as well as cvs co foo.

With the example file above, the output from cvs diff foobar will be in unidiff format. cvs diff -c foobar will provide context diffs, as usual. Getting "old" format diffs would be slightly more complicated, because diff doesn't have an option to specify use of the "old" format, so you would need cvs -f diff foobar.

In place of the command name you can use cvsnt to specify global options. For example the following line in .cvsrc

```
cvs -z6
```

causes cvsnt to use compression level 6.

The CVSROOT/cvsrc file on the server contains the default .cvsrc file that is used by all compatible clients. This is merged with the local .cvsrc file and the result behaves as normal.

The CVSROOT/cvsrc file cannot contain global options as it is parsed after the server has started.

Older cvsnt clients and Unix cvs clients will not use the global cvsrc.

Global options

The available cvs_options (that are given to the left of cvs_command) are:

-allow-root=rootdir

Specify legal cvsroot directory. Obsolete.

-a, --authenticate,

Authenticate all communication between the client and the server. Only has an effect on the cvsnt client. Authentication prevents certain sorts of attacks involving hijacking the active tcp connection. Enabling authentication does not enable encryption.

-b bindir

In CVS 1.9.18 and older, this specified that rcs programs are in the bindir directory. Current versions of cvsnt do not run rcs programs; for compatibility this option is accepted, but it does nothing.

-T tempdir

Use tempdir as the directory where temporary files are located. Overrides the setting of the $TMPDIR environment variable and any precompiled directory. This parameter should be specified as an absolute pathname.

-d cvs_root_directory

Use cvs_root_directory as the root directory pathname of the repository. Overrides the setting of the $CVSROOT environment variable.

-e editor

Use editor to enter revision log information. Overrides the setting of the $CVSEDITOR and $EDITOR environment variables.

-f

Do not read the ~/.cvsrc file. This option is most often used because of the non-orthogonality of the cvsnt option set. For example, the cvs log option -N (turn off display of tag names) does not have a corresponding option to turn the display on. So if you have -N in the ~/.cvsrc entry for log, you may need to use -f to show the tag names.

-F file

Read the contents of file and append it to the supplied command line. Arguments are separated by whitespace, and follow normal quoting rules.

-H, -help,

Display usage information about the specified cvs_command (but do not actually execute the command). If you don't specify a command name, cvs -H displays overall help for cvsnt, including a list of other help options.

-l

Do not log the cvs_command in the command history (but execute it anyway).

-n

Do not change any files. Attempt to execute the cvs_command, but only to issue reports; do not remove, update, or merge any existing files, or create any new files.

This option has a long history and is not guaranteed to actually leave the sandbox in the same state that it started with.

It is supported only for the *checkout* command to an empty directory, which is required for historical purposes (certain graphical user interfaces use this function).

cvsnt has other commands which replace the functionality of this option - see *status -q* and *ls* commands.

-N

Enable :local: access to a network share. Normally this is explicitly prohibited to discourage its use. It is recommended that you setup a proper server instead, as problems encountered using network shares will not normally be supported.

-o[locale]

Where supported by the server (CVSNT 2.0.57+), try to convert the character set of the server to that of the client. This allows you to store exended characters such an umlauts in the repository even if your machine is set to a different codepage/language to the server.

For Win32, the codepage used is the current ANSI codepage. This may not render correctly in the OEM codepage used by the command line processor. To verify that CVSNT is doing the correct conversion look at the filename in Windows Explorer.

As of CVSNT 2.0.59 this is the default behaviour.

-O

Disable client/server locale translation. If the client and server are not in the same locale then care must be taken not to use characters outside US-ASCII codepage if this option is used.

-Q

Cause the command to be really quiet; the command will only generate output for serious problems.

-q

Cause the command to be somewhat quiet; informational messages, such as reports of recursion through subdirectories, are suppressed.

-r

Make new working files read-only. Same effect as if the $CVSREAD environment variable is set (apc.html). The default is to make working files writable, unless watches are on.

--readonly

Make all users readonly. This is used for read only mirror servers.

-s variable=value

Set a user variable.

-t

Trace program execution; display messages showing the steps of cvsnt activity. Particularly useful with -n to explore the potential impact of an unfamiliar command. More instances of -t increase verbosity.

-v, --version,

Display version and copyright information for cvsnt.

-w

Make new working files read-write. Overrides the setting of the $CVSREAD environment variable. Files are created read-write by default, unless $CVSREAD is set or -r is given.

-x, --encrypt,

Encrypt all communication between the client and the server. Only has an effect on the cvsnt client. Enabling encryption implies that message traffic is also authenticated.

-z gzip-level

Set the compression level. Valid levels are 1 (high speed, low compression) to 9 (low speed, high compression), or 0 to disable compression (the default). Only has an effect on the cvsnt client.

Common command options

This section describes the command_options that are available across several cvsnt commands. These options are always given to the right of cvs_command. Not all commands support all of these options; each option is only supported for commands where it makes sense. However, when a command has one of these options you can almost always count on the same behavior of the option as in other commands. (Other command options, which are listed with the individual commands, may have different behavior from one cvsnt command to the other).

Warning: the history command is an exception; it supports many options that conflict even with these standard options.

-D date_spec

Use the most recent revision no later than date_spec. date_spec is a single argument, a date description specifying a date in the past.

The specification is sticky when you use it to make a private copy of a source file; that is, when you get a working file using -D, cvsnt records the date you specified, so that further updates in the same directory will use the same date.

-D is available with the checkout, diff, export, history, rdiff, rtag, and update commands.

A wide variety of date formats are supported by cvsnt. The most standard ones are ISO8601 (from the International Standards Organization) and the Internet e-mail standard (specified in RFC822 as amended by RFC1123).

ISO8601 dates have many variants but a few examples are:

```
1972-09-24
1972-09-24 20:05
```

There are a lot more ISO8601 date formats, and cvsnt accepts many of them, but you probably don't want to hear the *whole* long story :-).

In addition to the dates allowed in Internet e-mail itself, cvsnt also allows some of the fields to be omitted. For example:

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

The date is interpreted as being in the local timezone, unless a specific timezone is specified.

These two date formats are preferred. However, cvsnt currently accepts a wide variety of other date formats. They are intentionally not documented here in any detail, and future versions of cvsnt might not accept all of them.

One such format is month/day/year. This may confuse people who are accustomed to having the month and day in the other order; 1/4/96 is January 4, not April 1.

Remember to quote the argument to the -D flag so that your shell doesn't interpret spaces as argument separators. A command using the -D flag can look like this:

```
$ cvs diff -D "1 hour ago" cvs.texinfo
```

-f

When you specify a particular date or tag to cvsnt commands, they normally ignore files that do not contain the tag (or did not exist prior to the date) that you specified. Use the -f option if you want files retrieved even when there is no match for the tag or date. (The most recent revision of the file will be used).

Note that even with -f, a tag that you specify must exist (that is, in some file, not necessary in every file). This is so that cvsnt will continue to give an error if you mistype a tag name.

-f is available with these commands: annotate, checkout, export, rdiff, rtag, and update.

Warning: The commit and remove commands also have a -f option, but it has a different behavior for those commands.

-k kflag

Alter the default processing of keywords. Your kflag specification is sticky when you use it to create a private copy of a source file; that is, when you use this option with the checkout or update commands, cvsnt associates your selected kflag with the file, and continues to use it with future update commands on the same file until you specify otherwise.

The -k option is available with the add, checkout, diff, import and update commands.

-l

Local; run only in current working directory, rather than recursing through subdirectories.

Warning: this is not the same as the overall cvs -l option, which you can specify to the left of a cvs command!

Available with the following commands: annotate, checkout, commit, diff, edit, editors, export, log, rdiff, remove, rtag, status, tag, unedit, update, watch, and watchers.

-m message

Use message as log information, instead of invoking an editor.

Available with the following commands: add, commit and import.

-n

Do not run any checkout/commit/tag program. (A program can be specified to run on each of these activities, in the modules database this option bypasses it).

Warning: this is not the same as the overall cvs -n option, which you can specify to the left of a cvs command!

Available with the checkout, commit, export, and rtag commands.

-P

Prune empty directories.

-p

Pipe the files retrieved from the repository to standard output, rather than writing them in the current directory. Available with the checkout and update commands.

-R

Process directories recursively. This is on by default.

Available with the following commands: annotate, checkout, commit, diff, edit, editors, export, rdiff, remove, rtag, status, tag, unedit, update, watch, and watchers.

-r tag

Use the revision specified by the tag argument instead of the default head revision. As well as arbitrary tags defined with the tag or rtag command, two special tags are always available: HEAD refers to the most recent version available in the repository, and BASE refers to the revision you last checked out into the current working directory.

The tag specification is sticky when you use this with checkout or update to make your own copy of a file: cvsnt remembers the tag and continues to use it on future update commands, until you specify otherwise (for more information on sticky tags/dates.

The tag can be either a symbolic or numeric tag, or the name of a branch.

Specifying the -q global option along with the -r command option is often useful, to suppress the warning messages when the rcs file does not contain the specified tag.

Warning: this is not the same as the overall cvs -r option, which you can specify to the left of a cvsnt command!

-r is available with the checkout, commit, diff, history, export, rdiff, rtag, and update commands.

-W

Specify file names that should be filtered. You can use this option repeatedly. The spec can be a file name pattern of the same type that you can specify in the .cvswrappers file. Available with the following commands: import, and update.

add--Add files to repository

- Requires: repository, working directory.
- Changes: repository.
- Synonym: ad,new

This adds new files to the existing working directory. Before any commands which operate on sandbox files can be used, they must be added to the list of cvs controlled files using this command.

Directories are added immediately, and exist on all branches. Ordinary files must be committed before other users are able to use them.

add options

-b bugid

Mark the newly added file with a bug identifier.

-k kflag

Override the default expansion option for the file.

-m message

Use "message" for the message creation log.

-r branch

Add the new file onto a different branch (default is the same branch as the directory).

admin--Administration

- Requires: repository, working directory.
- Changes: repository.
- Synonym: adm,rcs

This is the cvsnt interface to assorted administrative facilities. Some of them have questionable usefulness for cvsnt but exist for historical purposes. Some of the questionable options are likely to disappear in the future. This command *does* work recursively, so extreme care should be used.

Do not use this command unless you know what you are doing. Some of the admin commands can have unexpected consequences.

On unix, if there is a group named cvsadmin, only members of that group can run cvs admin. This group should exist on the server, or any system running the non-client/server cvsnt. To disallow cvs admin for all users, create a group with no users in it. On NT, server administrators are able to use the admin command.

admin options

-ksubst

This option is provided as legacy support for older servers and has no function under CVSNT.

-l[rev]

Lock the revision with number rev. If a branch is given, lock the latest revision on that branch. If rev is omitted, lock the latest revision on the default branch. There can be no space between -l and its argument.

This command is depreciated in favour of the 'edit -c' command, which gives pseudo reserved checkouts.

-mrev:msg

Replace the log message of revision rev with msg.

-orange

Deletes (outdates) the revisions given by range.

Note that this command can be quite dangerous unless you know *exactly* what you are doing (for example see the warnings below about how the rev1:rev2 syntax is confusing).

If you are short on disc this option might help you. But think twice before using it--there is no way short of restoring the latest backup to undo this command! If you delete different revisions than you planned, either due to carelessness or (heaven forbid) a cvsnt bug, there is no opportunity to correct the error before the revisions are deleted. It probably would be a good idea to experiment on a copy of the repository first.

Specify range in one of the following ways:

rev1::rev2

Collapse all revisions between rev1 and rev2, so that cvsnt only stores the differences associated with going from rev1 to rev2, not intermediate steps. For example, after -o 1.3::1.5 one can retrieve revision 1.3, revision 1.5, or the differences to get from 1.3 to 1.5, but not the revision 1.4, or the differences between 1.3 and 1.4. Other examples: -o 1.3::1.4 and -o 1.3::1.3 have no effect, because there are no intermediate revisions to remove.

::rev

Collapse revisions between the beginning of the branch containing rev and rev itself. The branchpoint and rev are left intact. For example, -o ::1.3.2.6 deletes revision 1.3.2.1, revision 1.3.2.5, and everything in between, but leaves 1.3 and 1.3.2.6 intact.

rev::

Collapse revisions between rev and the end of the branch containing rev. Revision rev is left intact but the head revision is deleted.

rev

Delete the revision rev. For example, -o 1.3 is equivalent to -o 1.2::1.4.

rev1:rev2

Delete the revisions from rev1 to rev2, inclusive, on the same branch. One will not be able to retrieve rev1 or rev2 or any of the revisions in between. For example, the command cvs admin -oR_1_01:R_1_02 . is rarely useful. It means to delete revisions up to, and including, the tag R_1_02. But beware! If there are files that have not changed between R_1_02 and R_1_03 the file will have *the same* numerical revision number assigned to the tags R_1_02 and R_1_03. So not only will it be impossible to retrieve R_1_02; R_1_03 will also have to be restored from the tapes! In most cases you want to specify rev1::rev2 instead.

:rev

Delete revisions from the beginning of the branch containing rev up to and including rev.

rev:

Delete revisions from revision rev, including rev itself, to the end of the branch containing rev.

None of the revisions to be deleted may have branches or locks.

If any of the revisions to be deleted have symbolic names, and one specifies one of the :: syntaxes, then cvsnt will give an error and not delete any revisions. If you really want to delete both the symbolic names and the revisions, first delete the symbolic names with cvs tag -d, then run cvs admin -o. If one specifies the non-:: syntaxes, then cvsnt will delete the revisions but leave the symbolic names pointing to nonexistent revisions. This behavior is preserved for compatibility with previous versions of cvsnt, but because it isn't very useful, in the future it may change to be like the :: case.

Due to the way cvsnt handles branches rev cannot be specified symbolically if it is a branch.

Make sure that no-one has checked out a copy of the revision you outdate. Strange things will happen if he starts to edit it and tries to check it back in. For this reason, this option is not a good way to take back a bogus commit; commit a new revision undoing the bogus change instead.

-q

Run quietly; do not print diagnostics.

-t[file]

Useful with cvsnt. Write descriptive text from the contents of the named file into the rcs file, deleting the existing text. The file pathname may not begin with -. The descriptive text can be seen in the output from cvs log. There can be no space between -t and its argument.

If file is omitted, obtain the text from standard input, terminated by end-of-file or by a line containing . by itself. Prompt for the text if interaction is possible; see -I.

-t-string

Similar to -tfile. Write descriptive text from the string into the rcs file, deleting the existing text. There can be no space between -t and its argument.

annotate--find out who made changes to the files

- Requires: repository, working directory.
- Changes: nothing.
- Synonyms: ann

Annotate is used to discover who made changes to specific lines within files. The output to annotate gives the username, date and version number of the change.

The output to annotate is similar to checkout, for example:

```
1.54          (tmh      28-Aug-02):      char host[NI_MAXHOST];
1.54          (tmh      28-Aug-02):
1.71          (tmh      26-Mar-03):      if(!getnameinfo((struct
sockaddr*)&ss,ss_len,host,NI_MAXHOST,NULL,0,flags))
1.54          (tmh      28-Aug-02):          remote_host_name = xstrdup(host);
```

This information is usually enough to assign blame (or credit!) when tracing bugs.

annotate options

-l

Local directory only, no recursion

-R

Process directories recursively (default).

-f

Use head revision if tag is not found.

-r rev

Annotate files for specific revision or tag.

-D date

Annotate files for specific date

chacl--Change access control lists

- Requires: repository, working directory.
- Changes: repository.
- Synonyms: setacl, setperm

Modify the access control list for a file or directory.

chacl options

-a

Add access control entry - any combination of read, write, create, tag, control. Any of these may be prefixed by 'no' to deny access. Also special access all or none for setting all permissions.

-d

Delete access control entry

-j branch

Entry applies when merging from branch.

-m message

Show customised error message when access is blocked due to this entry.

-n

Stop entry from being inherited by subdirectories.

-p priority

Modify the priority that this entry has. This is an advanced option - the internal prioritisation is designed to work correctly in most circumstances.

-r branch

This entry applies to a single branch only.

-R

Recurse into subdirectories. Note that because entries are by default recursive this option is not normally required.

-u user

This entry applies to a single user (or group) only.

checkout--Check out sources for editing

- Synopsis: checkout [options] modules...
- Requires: repository.
- Changes: working directory.
- Synonyms: co, get

Create or update a working directory containing copies of the source files specified by modules. You must execute checkout before using most of the other cvsnt commands, since most of them operate on your working directory.

The modules are either symbolic names for some collection of source directories and files, or paths to directories or files in the repository. The symbolic names are defined in the modules file.

Depending on the modules you specify, checkout may recursively create directories and populate them with the appropriate source files. You can then edit these source files at any time (regardless of whether other software developers are editing their own copies of the sources); update them to include new changes applied by others to the source repository; or commit your work as a permanent change to the source repository.

Note that checkout is used to create directories. The top-level directory created is always added to the directory where checkout is invoked, and usually has the same name as the specified module. In the case of a module alias, the created sub-directory may have a different name, but you can be sure that it will be a sub-directory, and that checkout will show the relative path leading to each file as it is extracted into your private work area (unless you specify the -Q global option).

The files created by checkout are created read-write, unless the -r option to cvsnt is specified, the CVSREAD environment variable is specified or a watch is in effect for that file.

Note that running checkout on a directory that was already built by a prior checkout is also permitted. This is similar to specifying the -d option to the update command in the sense that new directories that have been created in the repository will appear in your work area. However, checkout takes a module name whereas update takes a directory name. Also to use checkout this way it must be run from the top level directory (where you originally ran checkout from), so before you run checkout to update an existing directory, don't forget to change your directory to the top level directory.

checkout options

These standard options are supported by checkout:

-D date

Use the most recent revision no later than date. This option is sticky, and implies -P.

-f

Only useful with the -D date or -r tag flags. If no matching revision is found, retrieve the most recent revision (instead of ignoring the file).

-k kflag

Process keywords according to kflag. This option is sticky; future updates of this file in this working directory will use the same kflag. The status command can be viewed to see the sticky options.

-l

Local; run only in current working directory.

-n

Do not run any checkout program (as specified with the -o option in the modules file).

-P

Prune empty directories.

-p

Pipe files to the standard output.

-R

Checkout directories recursively. This option is on by default.

-r tag

Use revision tag. This option is sticky, and implies -P.

In addition to those, you can use these special command options with checkout:

-A

Reset any sticky tags, dates, or -k options.

-c

Copy the module file, sorted, to the standard output, instead of creating or modifying any files or directories in your working directory.

-d dir

Create a directory called dir for the working files, instead of using the module name. In general, using this flag is equivalent to using mkdir dir; cd dir followed by the checkout command without the -d flag.

There is an important exception, however. It is very convenient when checking out a single item to have the output appear in a directory that doesn't contain empty intermediate directories. In this case *only*, cvsnt tries to "shorten" pathnames to avoid those empty directories.

For example, given a module foo that contains the file bar.c, the command cvs co -d dir foo will create directory dir and place bar.c inside. Similarly, given a module bar which has subdirectory baz wherein there is a file quux.c, the command cvs -d dir co bar/baz will create directory dir and place quux.c inside.

Using the -N flag will defeat this behavior. Given the same module definitions above, cvs co -N -d dir foo will create directories dir/foo and place bar.c inside, while cvs co -N -d dir bar/baz will create directories dir/bar/baz and place quux.c inside.

-j tag

With two -j options, merge changes from the revision specified with the first -j option to the revision specified with the second j option, into the working directory.

With one -j option, merge changes from the ancestor revision to the revision specified with the -j option, into the working directory. The ancestor revision is the common ancestor of the revision which the working directory is based on, and the revision specified in the -j option.

In addition, each -j option can contain an optional date specification which, when used with branches, can limit the chosen revision to one within a specific date. An optional date is specified by adding a colon (:) to the tag: -jSymbolic_Tag:Date_Specifier.

-b

Perform the -j merge from the branchpoint not the last mergepoint. This can be useful to re-merge changes that have been merged before, however it likely to produce a lot of conflicts.

-m

Perform the -j merge from the last recorded mergepoint. This is the default.

-N

Only useful together with -d dir. With this option, cvsnt will not "shorten" module paths in your working directory when you check out a single module. See the -d flag for examples and a discussion.

-s

Like -c, but include the status of all modules, and sort it by the status string.

-3

Produce 3-way conflict differences, containing the old and new files from the server and the edited files.

-S

Select between conflicting case-sensitive names on a case-insensitive client. This provides limited support for checking out repositories with such conflicts - the problem should really be fixed in the repository.

-t

Update using the last checkin time not the current time. This can cause issues with build systems so it is not recommended that this be used unless you are fully aware of the side-effects.

checkout examples

Get a copy of the module tc:

```
$ cvs checkout tc
```

Get a copy of the module tc as it looked one day ago:

```
$ cvs checkout -D yesterday tc
```

chown--Change directory owner

- Requires: working directory, repository
- Changes: repository.
- Synonyms: setowner

Change the owner of a directory. The owner has administration rights over files within that directory, in addition to the standard cvsnt administrators.

chown options

-R

Change directory owner recursively.

commit--Check files into the repository

- Synopsis: commit [-lnRf] [-m 'log_message' | -F file] [-r revision] [files...]
- Requires: working directory, repository.
- Changes: repository.
- Synonym: ci

Use commit when you want to incorporate changes from your working source files into the source repository.

If you don't specify particular files to commit, all of the files in your working current directory are examined. commit is careful to change in the repository only those files that you have really changed. By default (or if you explicitly specify the -R option), files in subdirectories are also examined and committed if they have changed; you can use the -l option to limit commit to the current directory only.

Commit verifies that the selected files are up to date with the current revisions in the source repository; it will notify you, and exit without committing, if any of the specified files must be made current first with update. Commit does not call the update command for you, but rather leaves that for you to do when the time is right.

When all is well, an editor is invoked to allow you to enter a log message that will be written to one or more logging programs and placed in the RCS file inside the repository. This log message can be retrieved with the log command. You can specify the log message on the command line with the -m message option, and thus avoid the editor invocation, or use the -F file option to specify that the argument file contains the log message.

commit options

These standard options are supported by commit, for a complete description of them):

-D

Assume all datestamps are different and send all files to the server for checking.

-l

Local; run only in current working directory.

-n

Do not run any module program.

-R

Commit directories recursively. This is on by default.

commit also supports these options:

-F file

Read the log message from file, instead of invoking an editor.

-f

Note that this is not the standard behaviour of the -f option as defined for common command options.

Force cvsnt to commit a new revision even if you haven't made any changes to the file. If the current revision of file is 1.7, then the following two commands are equivalent:

```
$ cvs commit -f file
$ cvs commit -r 1.8 file
```

The -f option disables recursion (i.e., it implies -l). To force cvsnt to commit a new revision for all files in all subdirectories, you must use -f -R.

This command is also used when changing -k expansion options. Unless -f is specified modified options will not be propogated back to the server.

-m message

Use message as the log message, instead of invoking an editor.

-c

Check for a valid edit on the file before committing. See 'cvs edit'.

-b bugid

Only commit files that have been edited and marked with bug bugid.

-B bugid

Mark committed files as belonging to bug bugid.

-e

Keep files edited after commit - supresses the automatic unedit that normally follows a commit.

-T

Attempt to move branches rather than create branch revisions where possible. Where a branch has no revisions, this option will compare what you are trying to commit with the parent branch head, and if they are identical it will move the branch rather than create a new revision.

This option has detrimental affects on reproducability, for example:

In branch A, a developer is working with files a (version 1.1) and b (version 1.1.2.1). He merges from HEAD, then commits using this option, moving the branch A in file a to version 1.2, and committing a new file b version 1.1.2.2

Meanwhile manager B notices that branch A is no longer builds (is broken), and asks the developer to fix it. The developer knows that the revsion 1.1.2.1 file works, so he wants to test with the older version. He can't. Since the environment that the older file was written in no longer exists (as the branchpoint has moved), it is impossible to roll back.

For this reason it is recommended that this option only be used where absolutely necessary, and on unrelated files only.

commit examples

Committing to a branch

You can commit to a branch revision (one that has an even number of dots) with the -r option. To create a branch revision, use the -b option of the rtag or tag commands. Then, either checkout or update can be used to base your sources on the newly created branch. From that point on, all commit changes made within these working sources will be automatically added to a branch revision, thereby not disturbing main-line development in any way. For example, if you had to create a patch to the 1.2 version of the product, even though the 2.0 version is already under development, you might do:

```
$ cvs rtag -b -r FCS1_2 FCS1_2_Patch product_module
$ cvs checkout -r FCS1_2_Patch product_module
$ cd product_module
[[ hack away ]]
$ cvs commit
```

This works automatically since the -r option is sticky.

Creating the branch after editing

Say you have been working on some extremely experimental software, based on whatever revision you happened to checkout last week. If others in your group would like to work on this software with you, but without disturbing main-line development, you could commit your change to a new branch. Others can then checkout your experimental stuff and utilize the full benefit of cvsnt conflict resolution. The scenario might look like:

```
[[ hacked sources are present ]]
$ cvs tag -b EXPR1
$ cvs update -r EXPR1
$ cvs commit
```

The update command will make the -r EXPR1 option sticky on all files. Note that your changes to the files will never be removed by the update command. The commit will automatically commit to the correct branch, because the -r is sticky.

To work with you on the experimental change, others would simply do

```
$ cvs checkout -r EXPR1 whatever_module
```

diff--Show differences between revisions

- Synopsis: diff [-lR] [format_options] [[-r rev1 | -D date1] [-r rev2 | -D date2]] [files...]
- Requires: working directory, repository.
- Changes: nothing.

The diff command is used to compare different revisions of files. The default action is to compare your working files with the revisions they were based on, and report any differences that are found.

If any file names are given, only those files are compared. If any directories are given, all files under them will be compared.

The exit status for diff is different than for other cvsnt commands.

diff options

These standard options are supported by diff:

-D date

Use the most recent revision no later than date. See -r for how this affects the comparison.

-k *kflag*

Process keywords according to *kflag*.

This option is for use when diffing two repository revisions - it will probably not do what you expect when diffing against a sandbox file.

-l

Local; run only in current working directory.

-R

Examine directories recursively. This option is on by default.

-r *tag*

Compare with revision tag. Zero, one or two -r options can be present. With no -r option, the working file will be compared with the revision it was based on. With one -r, that revision will be compared to your current working file. With two -r options those two revisions will be compared (and your working file will not affect the outcome in any way).

One or both -r options can be replaced by a -D date option, described above.

The following options specify the format of the output. They have the same meaning as in GNU diff.

```
-0 -1 -2 -3 -4 -5 -6 -7 -8 -9
--binary
--brief
--changed-group-format=arg
-c
```

```
  -C nlines
  --context[=lines]
-e --ed
-t --expand-tabs
-f --forward-ed
--horizon-lines=arg
--ifdef=arg
-w --ignore-all-space
-B --ignore-blank-lines
-i --ignore-case
-I regexp
   --ignore-matching-lines=regexp
-h
-b --ignore-space-change
-T --initial-tab
-L label
  --label=label
--left-column
-d --minimal
-N --new-file
--new-line-format=arg
--old-line-format=arg
--paginate
-n --rcs
-s --report-identical-files
-p
--show-c-function
-y --side-by-side
-F regexp
--show-function-line=regexp
-H --speed-large-files
--suppress-common-lines
-a --text
--unchanged-group-format=arg
-u
  -U nlines
  --unified[=lines]
-V arg
-W columns
  --width=columns
```

diff examples

The following line produces a Unidiff (-u flag) between revision 1.14 and 1.19 of backend.c. Due to the -kk flag no keywords are substituted, so differences that only depend on keyword substitution are ignored.

```
$ cvs diff -kk -u -r 1.14 -r 1.19 backend.c
```

Suppose the experimental branch EXPR1 was based on a set of files tagged RELEASE_1_0. To see what has happened on that branch, the following can be used:

```
$ cvs diff -r RELEASE_1_0 -r EXPR1
```

A command like this can be used to produce a context diff between two releases:

```
$ cvs diff -c -r RELEASE_1_0 -r RELEASE_1_1 > diffs
```

If you are maintaining ChangeLogs, a command like the following just before you commit your changes may help you write the ChangeLog entry. All local modifications that have not yet been committed will be printed.

```
$ cvs diff -u | less
```

edit--Mark files for editing

- Requires: repository, sandbox.
- Changes: Current directory.
- Synonyms:

This is command is used to mark files for editing in a reserved or semi-reserved scenario. When used with bug identifiers it also marks which users are currently working on which bugs, and which files are affected by those bugs.

cvs is primarily designed as a cooperative system, as experience has shown that this is the most productive way for teams of developers to work. The reserved models implemented by this command do not replace proper configuration management processes - the correct model to use should be decided after due consideration of the advantages and disadvantages of each.

The default working model is a cooperative multiple-editors model. Any number of people may be editing a file at any time and anyone may commit changes. This is very similar to the standard cooperative model except that the server keeps track of the editors.

Using the edit -c option creates a semi-reserved or cooperative reserved system. The edit command checks that there are no editors before editing the file, however its cooperative nature does not prevent other users editing if they wish to.

For a stricter model the -kc and -kx expansion options create a mandatory reserved system on individual files. Users are prevented from editing or committing a file unless they are the only editor of that file.

edit options

-A

Modify filesystem ACL on edited file (Win32 only, Experimental).

In shared sandbox scenarios this command is designed to stop other users from being able to modify the edited file - it create an access control list that disallows write access to anyone but the editor.

-a

Specify actions for the temporary watch that is created during an edit. This may be one of edit,unedit,commit,all,none.

-b bugid

Mark the edited file with a bug identifier. This marks the editor as not only using the file, but also working on a particular bug on that file. Specify multiple -b options for multiple bugs.

-c

Check that working files are unedited before appying the edit.

-f

Edit even if working files are already edited by someone else (default).

-l

Do not recurse into subdirectories.

-m message

Specify a reason for this edit. This message is made available as an option to the trigger libraries and notify script.

-R

Process directories recursively (default).

-w

Edit the whole file, not just the current branch. Normally edits only apply to the branch that is current at the time of the edit. If you wish to stop anyone changing other branches then this option allow this.

-x

Exclusive edit. Attempt to stop other users editing the file even if they do not use the -c option.

-z

Edit creates copies of the original files in the CVS/Base directory. With this option those copies are gzip compressed, which saves disk space and stops the copies being found by text searches of the sandbox.

editors--Find out who is editing a file

- Requires: repository, sandbox.
- Changes: nothing.
- Synonyms:

This command queries the server for everyone editing a file or group of files. It can also optionally show which bugs are being worked on.

The server only knows the last reported state of each client. In a controlled environment this is very likely to be accurate, however it is possible to leave edits on the server and not on the client (for example my deleting the sandbox without using commit/unedit).

The normal editors output is as follows:

```
components.dir/TEST.xml   tmh   Fri Dec  3 16:15:00 2004 GMT   tucker
c:\temp\repos\components.dir
rep/version_no.h          tmh   Thu Nov  4 17:37:43 2004 GMT   tucker   D:\t\test\rep
```

If you list edits for all branches and bug identifiers you get an extra columns. The full output is as follows:

```
components.dir/TEST.xml   tmh   Fri Dec  3 16:15:00 2004 GMT   tucker
c:\temp\repos\components.dir   3465   HEAD
rep/version_no.h          tmh   Thu Nov  4 17:37:43 2004 GMT   tucker   D:\t\test\rep
HEAD
```

The first output is designed to be compatible with older cvs versions that did not support the full cvsnt feature set.

editors options

-a

Show all branches, not just the current branch.

-c

Check whether edit on the selected files would actually succeed. This can be used by frontends to verify an edit without actually performing it.

-l

Process this directory only

-R

Process directories recursively (default).

-v

Show active bugs within the output.

export--Export sources from CVS, similar to checkout

- Synopsis: export [-flNnR] [-r rev|-D date] [-k subst] [-d dir] module...
- Requires: repository.
- Changes: current directory.

This command is a variant of checkout; use it when you want a copy of the source for module without the cvsnt administrative directories. For example, you might use export to prepare source for shipment off-site. This command requires that you specify a date or tag (with -D or -r), so that you can count on reproducing the source you ship to others (and thus it always prunes empty directories).

One often would like to use -kv with cvs export. This causes any keywords to be expanded such that an import done at some other site will not lose the keyword revision information. But be aware that doesn't handle an export containing binary files correctly. Also be aware that after having used -kv, one can no longer use the ident command (which is part of the rcs suite--see ident(1)) which looks for keyword strings. If you want to be able to use ident you must not use -kv.

export options

These standard options are supported by export:

-D date

Use the most recent revision no later than date.

-f

If no matching revision is found, retrieve the most recent revision (instead of ignoring the file).

-l

Local; run only in current working directory.

-n

Do not run any checkout program.

-R

Export directories recursively. This is on by default.

-r tag

Use revision tag.

In addition, these options (that are common to checkout and export) are also supported:

-d dir

Create a directory called dir for the working files, instead of using the module name.

-k subst

Set keyword expansion mode.

-N

Only useful together with -d dir.

history--Show status of files and users

- Synopsis: history [-report] [-flags] [-options args] [files...]
- Requires: the file $CVSROOT/CVSROOT/history
- Changes: nothing.

cvsnt can keep a history file that tracks each use of the checkout, commit, rtag, update, and release commands. You can use history to display this information in various formats.

Logging must be enabled by creating the file $CVSROOT/CVSROOT/history.

Warning: history uses -f, -l, -n, and -p in ways that conflict with the normal use inside cvsnt.

history options

Several options (shown above as -report) control what kind of report is generated:

-c

Report on each time commit was used (i.e., each time the repository was modified).

-e

Everything (all record types). Equivalent to specifying -x with all record types. Of course, -e will also include record types which are added in a future version of cvsnt; if you are writing a script which can only handle certain record types, you'll want to specify -x.

-m module

Report on a particular module. (You can meaningfully use -m more than once on the command line.)

-o

Report on checked-out modules. This is the default report type.

-T

Report on all tags.

-x type

Extract a particular set of record types type from the cvsnt history. The types are indicated by single letters, which you may specify in combination.

Certain commands have a single record type:

F release

O checkout

E export

T rtag

One of four record types may result from an update:

C A merge was necessary but collisions were detected (requiring manual merging).

G A merge was necessary and it succeeded.

U A working file was copied from the repository.

W The working copy of a file was deleted during update (because it was gone from the repository).

One of three record types results from commit:

A A file was added for the first time.

M A file was modified.

R A file was removed.

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

-a

Show data for all users (the default is to show data only for the user executing history).

-l

Show last modification only.

-w

Show only the records for modifications done from the same working directory where history is executing.

The options shown as -options args constrain the report based on an argument:

-b str

Show data back to a record containing the string str in either the module name, the file name, or the repository path.

-D date

Show data since date. This is slightly different from the normal use of -D date, which selects the newest revision older than date.

-f file

Show data for a particular file (you can specify several -f options on the same command line). This is equivalent to specifying the file on the command line.

-n module

Show data for a particular module (you can specify several -n options on the same command line).

-p repository

Show data for a particular source repository (you can specify several -p options on the same command line).

-r rev

Show records referring to revisions since the revision or tag named rev appears in individual rcs files. Each rcs file is searched for the revision or tag.

-t tag

Show records since tag tag was last added to the history file. This differs from the -r flag above in that it reads only the history file, not the rcs files, and is much faster.

-u name

Show records for user name.

-z timezone

Show times in the selected records using the specified time zone instead of UTC.

import--Import sources into CVS, using vendor branches

- Synopsis: import [-options] repository vendortag releasetag...
- Requires: Repository, source distribution directory.
- Changes: repository.

Use import to incorporate an entire source distribution from an outside source (e.g., a source vendor) into your source repository directory. You can use this command both for initial creation of a repository, and for wholesale updates to the module from the outside source.

The repository argument gives a directory name (or a path to a directory) under the cvsnt root directory for repositories; if the directory did not exist, import creates it.

When you use import for updates to source that has been modified in your source repository (since a prior import), it will notify you of any files that conflict in the two branches of development; use checkout -j to reconcile the differences, as import instructs you to do.

If cvsnt decides a file should be ignored, it does not import it and prints I followed by the filename, for a complete description of the output).

If the file $CVSROOT/CVSROOT/cvswrappers exists, any file whose names match the specifications in that file will be treated as packages and the appropriate filtering will be performed on the file/directory before being imported.

The outside source is saved in a first-level branch, by default 1.1.1. Updates are leaves of this branch; for example, files from the first imported collection of source will be revision 1.1.1.1, then files from the first imported update will be revision 1.1.1.2, and so on.

At least one argument is required. repository is needed to identify the collection of source. Normally also two other arguments are supplied - vendortag is a tag for the entire branch (e.g., for 1.1.1). You must also specify at least one releasetag to identify the files at the leaves created each time you execute import.

Note that by default import does *not* change the directory in which you invoke it. In particular, it does not set up that directory as a cvsnt working directory. For initial imports the -C option will achieve this, but for vendor source imports you need to import them first and then check them out into a different directory.

import options

This standard option is supported by import:

-m message

Use message as log information, instead of invoking an editor.

There are the following additional special options.

-b branch

Branch.

-k subst

Indicate the keyword expansion mode desired. This setting will apply to all files created during the import, but not to any files that previously existed in the repository.

-I name

Specify file names that should be ignored during import. You can use this option repeatedly. To avoid ignoring any files at all (even those ignored by default), specify `-I !'.

name can be a file name pattern of the same type that you can specify in the .cvsignore file.

If you specify '-I @' the contents of .cvsigore files are ignored for the import.

-W spec

Specify file names that should be filtered during import. You can use this option repeatedly. To override all the default wrappers specify '-W !'.

spec can be a file name pattern of the same type that you can specify in the .cvswrappers file.

-C

Create CVS directories during initial import. This provides simplified setup of a sandbox, however as it does not contact the server is not suitable for vendor updates - for this a proper import/checkout sequence should be used.

-i

Ignore files that have illegal windows filenames (: * ? " < > |).

-d

Use the modification time of the file as the import time instead of the current time.

-f

Overwrite any duplicate release tags within imported files.

-n

Do not require vendor or release tags. This is used for initial imports only, and creates a repository without a vendor branch. If you are not planning to use vendor source imports then using this option simplifies the import process.

import output

import keeps you informed of its progress by printing a line for each file, preceded by one character indicating the status of the file:

U file

The file already exists in the repository and has not been locally modified; a new revision has been created (if necessary).

N file

The file is a new file which has been added to the repository.

C file

The file already exists in the repository but has been locally modified; you will have to merge the changes.

I file

The file is being ignored.

L file

The file is a symbolic link; cvs import ignores symbolic links. People periodically suggest that this behaviour should be changed, but if there is a consensus on what it should be changed to, it doesn't seem to be apparent. (Various options in the modules file can be used to recreate symbolic links on checkout, update, etc.

init--Initialise a new repository

- Requires: local access.
- Changes: repository.
- Synonyms:

Initialises a new repository for use. This command can only be issued locally on the server, not remotely.

init options

Init can be called successfully without any options.

-a alias

Define the repository alias for the new repository.

-d description

Set the repository description

-f

Force overwrite of an existing repository. This doesn't normally make sense, so be careful with this option.

-r repository

Remote repository creation. To successfully use this you must have an admin account on an existing repository, remote init must be initialized on the server, and the server needs the access rights to modify its global configuration.

-n

Do not attempt repository registration

-u

Unregister an existing repository. For this to succeed remotely the same conditions must exist as in -r.

info--Get information about the client and server

- Requires: nothing. (Repository required for server functions).
- Changes: nothing.
- Synonyms: inf

Return information about available protocols on the client or server. Also list the current cvsignore and cvswrappers settings.

Without any parameters this lists available protocols:

```
$ cvs info
Available protocols:
--
Available protocols:

local               (internal)
ext                 ext 2.8.01 (Soolin) Build 9329
gserver             gserver 2.8.01 (Soolin) Build 9329 (Active Directory)
pserver             pserver 2.8.01 (Soolin) Build 9329
server              server 2.8.01 (Soolin) Build 9329
sserver             sserver 2.8.01 (Soolin) Build 9329
ssh                 ssh 2.8.01 (Soolin) Build 9329
sspi                sspi 2.8.01 (Soolin) Build 9329
```

For information about an individual protocol specify the prototocol name on the command line.

```
$ cvs info pserver
Name:               pserver
Version:            pserver 2.8.01 (Soolin) Build 9329
Syntax:             :pserver[;keyword=value...]:[username[:password]@]host[:port][:]/path
  Username:         Optional
  Password:         Optional
  Hostname:         Required
  Port:             Optional
Client:             Yes
Server:             Yes
Login:              Yes
Encryption:         No
Impersonation:      CVS Builtin

Keywords available:

proxy               Proxy server
proxyport           Proxy server port (alias: proxy_port)
tunnel              Proxy protocol (aliases: proxyprotocol,proxy_protocol)
proxyuser           Proxy user (alias: proxy_user)
proxypassword       Proxy passsord (alias: proxy_password)
```

The format is designed to be easily parsed by frontends. Its layout does not change between cvs versions, however lines may be added or deleted from the output.

Specifying cvswrappers or cvsignore dumps out the internal state of these files. It is possible to have duplicates, as the list is built up of both the client and server contents. When parsed however the client always takes precedence over the server setting.

info options

-c

Return client-side information. Returns all protocols available to the client

-s

Return server-side information. Returns protocols that a client can use to communicate with the server. This does not include local or external protocols.

-b

(Where supported) list available cvsnt servers on the local network. This currently requires mdns support on the client.

-r server

Find out as much as possible about a remote cvsnt server. For this command to succeed the remote server must support the cvsnt enumeration protocol.

```
$ cvs info -r cvs.cvsnt.org
Server: cvs.cvsnt.org
Version: Concurrent Versions System (CVSNT) 2.8.01 (Soolin) Build 9329

Protocols:
  gserver
  pserver
  sserver
  sync

Repositories:
  /usr/local/cvs                cvsnt main repository

Anonymous username: cvs
Anonymous protocol: pserver
Default repository: /usr/local/cvs

Anonymous login: :pserver:cvs@cvs.cvsnt.org:/usr/local/cvs
Default login: :sserver:cvs.cvsnt.org:/usr/local/cvs
```

The layout will remain the same as much as possible in future revisions to facilitate automatic parsing. Parsers should ignore elements that they do not understand.

log--Print out log information for files

- Synopsis: log [options] [files...]
- Requires: repository, working directory.
- Changes: nothing.

Display log information for files. log used to call the rcs utility rlog. Although this is no longer true in the current sources, this history determines the format of the output and the options, which are not quite in the style of the other cvsnt commands.

The output includes the location of the rcs file, the head revision (the latest revision on the trunk), all symbolic names (tags) and some other things. For each revision, the revision number, the author, the number of lines added/deleted and the log message are printed. All times are displayed in Coordinated Universal Time (UTC). (Other parts of cvsnt print times in the local timezone).

Warning: log uses -R in a way that conflicts with the normal use inside cvsnt.

log options

By default, log prints all information that is available. All other options restrict the output.

-B bugid

Only select revisions which are related to a single bug.

-b

Print information about the revisions on the default branch, normally the highest branch on the trunk.

-d dates

Print information about revisions with a checkin date/time in the range given by the semicolon-separated list of dates. The date formats accepted are those accepted by the -D option to many other cvsnt commands. Dates can be combined into ranges as follows:

d1<d2, d2>d1,

Select the revisions that were deposited between d1 and d2.

<d, d>,

Select all revisions dated d or earlier.

d<, >d,

Select all revisions dated d or later.

d

Select the single, latest revision dated d or earlier.

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

Note that the separator is a semicolon (;).

-h

Print only the name of the rcs file, name of the file in the working directory, head, default branch, access list, locks, symbolic names, and suffix.

-l

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

-N

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

-R

Print only the name of the rcs file.

-rrevisions

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

rev1:rev2

Revisions rev1 to rev2 (which must be on the same branch).

rev1::rev2

Revisions between, but not including, rev1 and rev2.

:rev

Revisions from the beginning of the branch up to and including rev.

::rev

Revisions from the beginning of the branch up to, but not including, rev.

rev:

Revisions starting with rev to the end of the branch containing rev.

rev::

Revisions starting just after rev to the end of the branch containing rev.

branch

An argument that is a branch means all revisions on that branch.

branch1:branch2, branch1::branch2,

A range of branches means all revisions on the branches in that range.

branch.

The latest revision in branch.

A bare -r with no revisions means the latest revision on the default branch, normally the trunk. There can be no space between the -r option and its argument.

-S

Suppress log output when no revisions are selected within a file.

-s states

Print information about revisions whose state attributes match one of the states given in the comma-separated list states.

-T

Display dates and times in the log output in Local time rather than GMT.

-t

Print the same as -h, plus the descriptive text.

-w[logins]

Print information about revisions checked in by users with login names appearing in the comma-separated list logins. If logins is omitted, the user's login is assumed. There can be no space between the -w option and its argument.

-X

Suppress extended information generated only by CVSNT servers. This can be useful with some frontends that cannot parse the extra output.

-x

Generate full output. This is the default unless configured otherwise on the server.

log prints the intersection of the revisions selected with the options -B,-d, -s, and -w, intersected with the union of the revisions selected by -b and -r.

login--Cache a client password locally

- Requires: repository.
- Changes: local password cache.
- Synonyms: logon, lgn

Cache the password required for the client locally. Not all protocols require this, and some do not even support it. If you are using a protocol that does not require this then for security reasons it is better not to use it, since the local cache is relatively easy to find and decrypt if your local account/machine is compromised.

Do not make any assumptions about the storage of passwords in the local cache. In particular do not attempt to manipulate it manually - its format may change without warning.

login options

-p password

Specify the password to use (the default is to prompt).

logout--Remove the cached entry for a password

- Requires: repository.
- Changes: local password cache.
- Synonyms:

  Destroy the password cache entry for the current connection.

  logout options

  none.

ls--list modules, files and directories in the repository

- Requires: repository.
- Changes: nothing.
- Synonyms: dir,list,rls

Lists the contents of the repository, and optionally the latest version information from files within the repository.

Used without any parameters, it lists the toplevel directories (modules) in the repository. This includes directories created using the modules2 file.

The list is followed by the contents of the modules file, if available.

ls options

-D date

Show files current on a particular date.

-e

Display in CVS/Entries format:

```
$ cvs ls -e CVSROOT
Listing module: CVSROOT

/checkoutlist/1.9/Wed Jan 26 19:08:06 2005/-kkv/
/commitinfo/1.10/Tue Jan 11 01:25:34 2005/-kkv/
/config/1.15/Sun Jan 23 02:15:57 2005/-kkv/
```

-l

Display all details. Note that the usage of the -l option differs from other cvs commands. This is for consistency with the unix-style ls command.

```
$ cvs ls -l CVSROOT
Listing module: CVSROOT

checkoutlist                    1.9     Wed Jan 26 19:08:06 2005 -kkv
commitinfo                      1.10    Tue Jan 11 01:25:34 2005 -kkv
config                          1.15    Sun Jan 23 02:15:57 2005 -kkv
```

-P

Ignore (Prune) empty directories.

-q

Quieter output. Do not print extraneous human-readable prompts.

-R

Recurse into subdirectories.

-r tag

Show files with the specified revision, tag or branch.

-T

Show timestamps in local time instead of GMT.

lsacl--Show file/directory permissions

- Requires: repository, sandbox.
- Changes: nothing.
- Synonyms: lsattr,listperm

List the access control lists entries for files and directories within the current sandbox. For directories also shows the owner.

Permissions on a file or directory are also supplemented by parent directories, so the lack of mention of a user does not imply or deny access. For compatibility with older cvs versions the default is to grant permissions unless explicitly denied. This can be changed by putting an inheritable default deny permission in the repository root.

lsacl options

-d

List only directories, not files. By default both file and directory permisisons are listed in the output.

-R

Recursively list permissions in all subdirectories.

rlsacl--Show remote file/directory permissions

- Requires: repository
- Changes: nothing.
- Synonyms: rlsattr,rlistperm

List permissions remotely, without reference to a local sandbox.

passwd--Modify a user's password or create a user

- Requires: repository.
- Changes: remote password file.
- Synonyms: password, setpass.

Change the username/password information for a user. This command is only useful for those protocols which do not use system passwords (such as pserver). It does not affect the real system password of the user.

Ordninary users are only able to change their own cvs password. Repository administrators can use the full funcitonality of this command.

If invoked without a username, the current username is used. If invoked with a username, the repository administrator can change the details of another user.

passwd options

-a

Add user. Adds a new user entry to the password file.

-x

Disable user. Changes the password so that the user cannot log in.

-X

Delete user. Remove the user entry from the password file.

-r user

Alias username to real system user. Before a virtual (pserver) user can log in the system needs to know which user account to use for that user.

-R

Remove system alias for user.

-D domain

(Win32 only) Use the users' domain password instead of a separate password. For security reasons this is not recommended.

rannotate--Show who made changes to remote files

- Requires: repository.
- Changes: nothing.
- Synonyms: rann, ra

Show changes on remote files within a repository. Does not require a sandbox.

rchacl--Change remote access control lists

- Requires: repository.
- Changes: repository.
- Synonyms: rsetacl, rsetperm

Change an access control list on a remote file or directory. Does not require a sandbox.

rchown--Change owner of a remote directory

- Requires: repository.
- Changes: repository.
- Synonyms: rsetowner

Change the owner of a remote directory. Does not require a sandbox.

rdiff--'patch' format diffs between releases

- rdiff [-flags] [-V vn] [-r t|-D d [-r t2|-D d2]] modules...
- Requires: repository.
- Changes: nothing.
- Synonyms: patch, pa

Builds a Larry Wall format patch(1) file between two releases, that can be fed directly into the patch program to bring an old release up-to-date with the new release. (This is one of the few cvsnt commands that operates directly from the repository, and doesn't require a prior checkout.) The diff output is sent to the standard output device.

You can specify (using the standard -r and -D options) any combination of one or two revisions or dates. If only one revision or date is specified, the patch file reflects differences between that revision or date and the current head revisions in the rcs file.

Note that if the software release affected is contained in more than one directory, then it may be necessary to specify the -p option to the patch command when patching the old sources, so that patch is able to find the files that are located in other directories.

rdiff options

These standard options are supported by rdiff:

-D date

Use the most recent revision no later than date.

-f

If no matching revision is found, retrieve the most recent revision (instead of ignoring the file).

-l

Local; don't descend subdirectories.

-R

Examine directories recursively. This option is on by default.

-r tag

Use revision tag.

In addition to the above, these options are available:

-c

Use the context diff format. This is the default format.

-s

Create a summary change report instead of a patch. The summary includes information about files that were changed or added between the releases. It is sent to the standard output device. This is useful for finding out, for example, which files have changed between two dates or revisions.

-t

A diff of the top two revisions is sent to the standard output device. This is most useful for seeing what the last change to a file was.

-u

Use the unidiff format for the context diffs. Remember that old versions of the patch program can't handle the unidiff format, so if you plan to post this patch to the net you should probably not use -u.

-V vn

Expand keywords according to the rules current in rcs version vn (the expansion format changed with rcs version 5). Note that this option is no longer accepted. cvsnt will always expand keywords the way that rcs version 5 does.

rdiff examples

Suppose you receive mail from foo@example.net asking for an update from release 1.2 to 1.4 of the tc compiler. You have no such patches on hand, but with cvsnt that can easily be fixed with a command such as this:

```
$ cvs rdiff -c -r FOO1_2 -r FOO1_4 tc | \
$$ Mail -s 'The patches you asked for' foo@example.net
```

Suppose you have made release 1.3, and forked a branch called R_1_3fix for bugfixes. R_1_3_1 corresponds to release 1.3.1, which was made some time ago. Now, you want to see how much development has been done on the branch. This command can be used:

```
$ cvs patch -s -r R_1_3_1 -r R_1_3fix module-name
cvs rdiff: Diffing module-name
File ChangeLog,v changed from revision 1.52.2.5 to 1.52.2.6
File foo.c,v changed from revision 1.52.2.3 to 1.52.2.4
File bar.h,v changed from revision 1.29.2.1 to 1.2
```

release--Indicate that a Module is no longer in use

- release [-d [-f]] [-e] [-y] directories...
- Requires: Working directory.
- Changes: Working directory, history log.
- Synonyms: re, rel

This command is meant to safely cancel the effect of cvs checkout. Since cvsnt doesn't lock files, it isn't strictly necessary to use this command. You can always simply delete your working directory, if you like; but you risk losing changes you may have forgotten, and you leave no trace in the cvsnt history file that you've abandoned your checkout.

Use cvs release to avoid these problems. This command checks that no uncommitted changes are present; that you are executing it from immediately above a cvsnt working directory; and that the repository recorded for your files is the same as the repository defined in the module database.

If all these conditions are true, cvs release leaves a record of its execution (attesting to your intentionally abandoning your checkout) in the cvsnt history log.

release options

The release command supports the following command options:

-d

Delete your working copy of the file if the release succeeds. If this flag is not given your files will remain in your working directory.

-f

Must be specified with -f, above. Force the the deletion of the directory even if non-cvs files are present.

-e

Don't delete any files, just delete the cvsnt administrative directories. The directory is then left in a state as if it had just been exported.

-y

Automatically assume 'yes' to any confirmation prompts.

release output

Before release releases your sources it will print a one-line message if any file that is not up-to-date.

release examples

Release the tc directory, and delete your local working copy of the files.

```
$ cd ..          # You must stand immediately above the
                 # sources when you issue cvs release.
```

```
$ cvs release -d tc
You have [5] altered files in this repository.
Are you sure you want to release (and delete) directory `tc': y
$
```

remove--Remove files from the working directory

- Requires: working directory, repository.
- Changes: working directory.
- Synonyms: rm, delete

Remove a file from the working directory, marking the file as 'dead' which comes into effect after the next commit.

Files are never actually removed from the repository, only ever flagged as deleted. You can recover such a removed file by using a combinaiton of add and commit. As a safety measure this command will not do anything unless the physical file is already deleted or you use the -f option.

remove options

-f

Delete the physical file as well. Remove will not complete unless the file has already been deleted or this option is given.

-l

Process this directory only.

-R

Process directories recursively.

rename--Rename files in the repository

- Synopsis: rename [-q] source target
- Requires: working directory, repository.
- Changes: repository.
- Synonyms: ren,mv

Use the rename to rename or move a file within the sandbox, whilst keeping the history intact.

Rename functions depend on the capabilities of the server:

CVS Suite 2008:	limited support for rename, move not supported
CVS Suite 2009R2:	rename supported, move not supported
CVS Suite 2.8.02:	rename supported, move available but not supported
CM Suite 2008:	rename and move supported
CVSNT Community Edition 2.5.03, 2.0.51 etc:	due to serious bugs, rename and move not supported
CVS 1.11.x, 1.12.x etc:	rename and move not available

Complex renames/move (across directories) should be avoided. If you must rename across directories then ensure that the 'destination' directory gets committed at the same time as the 'source' directory - otherwise the file can vanish completely.

Rename information is held at the directory level, so the rename/move is not committed to the repository until cvs commit is called on the directory containing the file.

If another user has the file checked out they will continue to use the file under its old name until they issue a cvs update at the directory level. CVSNT has no problems with this and both users can continue to merge each others' changes.

rlog--Return log history of remote file

- Requires: repository.
- Changes: nothing.
- Synonyms: rl

Return the log history of a remote file or group of files. Does not require a sandbox.

rtag--Mark a single revision over multiple files

- Requires: repository.
- Changes: nothing.
- Synonyms: rfreeze

Set a tag on a group of files in a repository. Does not require a sandbox.

status--Display the state of a file in the working directory

- Requires: repository, working directory.
- Changes: nothing.
- Synonyms: st,stat

Display the status of a file within the working directory. This includes any expansion options, its version, and whether it is modified or may require updating.

The normal output from the status command is as follows:

```
$ cvs status cvs.dbk
===================================================================
File: cvs.dbk           Status: Up-to-date

   Working revision:    1.1.2.36
   Repository revision: 1.1.2.36        /usr/local/cvs/cvsnt/doc/cvs.dbk,v
   Expansion option:    o
   Commit Identifier:   75a042064840566c
   Sticky Tag:          CVSNT_2_0_x (branch: 1.1.2)
   Sticky Date:         (none)
   Sticky Options:      -ko
   Merge From:          (none)
```

The layout of this output will remain the same across versions, although information may be added or removed.

A more terse form of status is produced by using the -q option, in which case only the checkout status is displayed:

```
$ cvs status -q cvs.dbk
File: cvs.dbk           Status: Up-to-date
```

status options

-v

Verbose format. Append the tag information for each selected file.

-l

Process this directory only.

-R

Process directories recursively.

-q

Display only a quick summary of the status of each file. Specifying a second -q option reduces the output still further, by supressing output for up to date files.

-X

Display shorter output produced by cvs 1.x. This output may be required for parsing with older tools.

-x

Display full cvsnt status details. Default, unless overridden on the server.

tag--Create a tag or branch

- Requires: repository, working directory.
- Changes: repository.
- Synonyms: ta,freeze

Create or modify a tag in the repository.

A tag is a snapshot of a single moment in time in the repository. Normally a tag would be applied to entire directories, although it is possible to tag individual files if required.

A branch is a unit of parallel development, which may or may not be kept in sync with the main trunk.

Creating a tag or branch does not change the working directory. To create and work with a branch it is also necessary to use the `cvs update` command to move your working directory onto that branch.

tag options

-A

Make an alias of an existing branch (requires -r).

-b

Make a branch tag.

-c

Check that the working files are unmodified before tagging.

-d

Delete the named tag. Deletion of branches is not recommended.

-F

Move the tag if it already exists. Not recommended for branches.

-B

Allow -d and -F to be applied to branch tags. Use of this option is not recommended as it does not affect the revisions within the branch and can result in them being orphaned.

-f

Force a head revision match if the existing branch is not found.

-l

Process local directory only.

-M

Create a floating, or 'magic' branch. A floating branch always points to the head of its parent branch, unless a revision is checked into it. Once a revision is added it becomes a normal fixed branch.

-R

Process directories recursively.

-r rev

Select files based on existing tag/branch/revision.

-D date

Select files current on a specific date.

unedit--Mark edit as finished without committing

- Requires: repository, working directory.
- Changes: working directory.
- Synonyms:

Discard any changes made and finish editing a file without committing. It may also be necessary to run an update command to retrieve the latest version of the file.

Unediting also sends out a notification to other users if the server is configured to do this. It will mark the working directory file as read only.

unedit options

-b bugid

Unedit only files marked as edited with bugid.

-l

Process local directory only.

-m message

Specify the reason for this unedit. The message is sent to the trigger and notify programs on the server.

-r

Revert file only. Do not perform unedit. This merely copies the unedited copy back onto the working copy.

-R

Process directories recursively.

-u username

(repository administrators only) perform an unedit for another user.

Note: this command should be ran on a “clean” sandox / workspace or the *username* workspace. This command will unedit files owned by *username* and also the current user.

-w

Leave working directory file writable after the unedit.

update--Bring work tree in sync with repository

- Requires: repository, working directory.
- Changes: working directory.
- Synonyms: up,upd

After you've run checkout to create your private copy of source from the common repository, other developers will continue changing the central source. From time to time, when it is convenient in your development process, you can use the update command from within your working directory to reconcile your work with any revisions applied to the source repository since your last checkout or update.

It is unwise to let your local working directory become out of sync with others for too long. Depending on your working model it may be necessary to run updates daily or even hourly to keep in step. On the other hand if you are the only developer on a project it may not be necessary to update at all.

If updating is left too long, then conflicts that arise get progressively harder to fix over time as the code diverges. On the other hand frequent updating may mean that there are no conflicts to deal with at all.

update options

These standard options are available with update:

-D date

Use the most recent revision no later than date. This option is sticky, and implies -P.

-f

Only useful with the -D date or -r tag flags. If no matching revision is found, retrieve the most recent revision (instead of ignoring the file).

-F

Write clean repository copy to this filename. This is most often used when you need to perfom a visual side-by-side diff - for which you need two complete copies of the file, but where the 'name' of the file for that point in time may not be known. ie: you haVe the file with *filename* 'blat.htm' in your current directory and you want to compare it to the HEAD revision: but at HEAD it may have a different name (someone has used the `rename` command). You can use the `diff` without knowing the revisions filename, but you can't use the `checkout` unless you know the actual filename for that point in time. So this is the situation where this flag of the udpate command can be used.

-k kflag

Process keywords according to kflag. This option is sticky; future updates of this file in this working directory will use the same kflag. The status command can be viewed to see the sticky options.

-l

Local; run only in current working directory.

-P

Prune empty directories.

-p

Pipe files to the standard output.

-R

Update directories recursively (default).

-r rev

Retrieve revision/tag rev. This option is sticky, and implies -P. These special options are also available with update.

-3

Provide 3-way conflicts.

-A

Reset any sticky tags, dates, or -k options.

-B bugid

Set the boundary of a -j merge to revisions marked with a particular bug. This is used to extract individual bug fixes from one branch to another.

If there are other revisions unrelated to the bug required to merge all the differences, these will also be merged. This option is much more useful in quiet or controlled repositories where this happens infrequently.

-b

Perform the -j merge from the branchpoint, ignoring mergepoints.

-C

Overwrite locally modified files with clean copies from the repository (the modified file is saved in .#file.revision, however).

-c

If the file is edited, update the base revision copy to the latest revision. If this option is not used an unedit will always revert to the same revision that is edited, not the latest revision in the repository.

-d

Create any directories that exist in the repository if they're missing from the working directory. Normally, update acts only on directories and files that were already enrolled in your working directory.

This is useful for updating directories that were created in the repository since the initial checkout; but it has an unfortunate side effect. If you deliberately avoided certain directories in the repository when you created your working directory (either through use of a module name or by listing explicitly the files and directories you wanted on the command line), then updating with -d will create those directories, which may not be what you want.

-I name

Ignore files whose names match name (in your working directory) during the update. You can specify -I more than once on the command line to specify several files to ignore. Use -I ! to avoid ignoring any files at all, for other ways to make cvsnt ignore some files.

-m

Perform the -j merge based on the last mergepoint. This is the default.

-S

Perform limited selection between conflicting case sensitive names on a case insensitive system. This option can be used to checkout files with conflicting names however it is not a solution to the problem - the conflict should be fixed in the repository.

-t

Update using the last checkin time of the file not the current time. Do not use this option if you are using a makefile based system as it will cause problems with the build process. On other systems be aware of any side effects before using this option.

-Wspec

Specify file names that should be filtered during update. You can use this option repeatedly. Use -W ! avoid using the default wrappers. spec can be a file name pattern of the same type that you can specify in the .cvswrappers file.

-jrevision

With two -j options, merge changes from the revision specified with the first -j option to the revision specified with the second j option, into the working directory.

With one -j option, merge changes from the ancestor revision to the revision specified with the -j option, into the working directory. The ancestor revision is the common ancestor of the revision which the working directory is based on, and the revision specified in the -j option.

Note that using a single -j tagname option rather than -j branchname to merge changes from a branch will often not remove files which were removed on the branch.

In addition, each -j option can contain an optional date specification which, when used with branches, can limit the chosen revision to one within a specific date. An optional date is specified by adding a colon (:) to the tag: -jSymbolic_Tag:Date_Specifier.

update output

update and checkout keep you informed of their progress by printing a line for each file, preceded by one character indicating the status of the file:

U file

The file was brought up to date with respect to the repository. This is done for any file that exists in the repository but not in your source, and for files that you haven't changed but are not the most recent versions available in the repository.

P file

Like U, but the cvsnt server sends a patch instead of an entire file. These two things accomplish the same thing.

A file

The file has been added to your private copy of the sources, and will be added to the source repository when you run commit on the file. This is a reminder to you that the file needs to be committed.

R file

The file has been removed from your private copy of the sources, and will be removed from the source repository when you run commit on the file. This is a reminder to you that the file needs to be committed.

M file

The file is modified in your working directory.

M can indicate one of two states for a file you're working on: either there were no modifications to the same file in the repository, so that your file remains as you last saw it; or there were modifications in the repository as well as in your copy, but they were merged successfully, without conflict, in your working directory.

cvsnt will print some messages if it merges your work, and a backup copy of your working file (as it looked before you ran update) will be made. The exact name of that file is printed while update runs.

C file

A conflict was detected while trying to merge your changes to file with changes from the source repository. file (the copy in your working directory) is now the result of attempting to merge the two revisions; an unmodified copy of your file is also in your working directory, with the name .#file.revision where revision is the revision that your modified file started from. Resolve the conflict. (Note that some systems automatically purge files that begin with .# if they have not been accessed for a few days. If you intend to keep a copy of your original file, it is a very good idea to rename it.) Under VMS, the file name starts with __ rather than .#.

? file

file is in your working directory, but does not correspond to anything in the source repository, and is not in the list of files for cvsnt to ignore.

version--Display client and server versions.

- Requires: nothing.
- Changes: nothing.
- Synonyms: ve,ver

Display the version of the client in use. Also displays the version of the remote server if that information is available.

version options

-q

Display only the version number of the local client, not other information.

watch--Watch for changes in a file

- Requires: repository, working directory.
- Changes: repository.
- Synonyms:

Add yourself to the list of watchers for a file. Watchers are notified via the notify script whenever an action they are interested in happens.

watch options

on | off

Enables or disables watches. Enable automatically implies *ro*.

ro

All files "checked out" in future will automatically have the read only attribute set.

rw

Disables automatically setting the read only attribute of checked out files.

add | remove

add or remove notification on actions (must specify -a).

-l (applies to: on/off/ro/rw/add/remove)

Process local directory only.

-R (applies to: on/off/ro/rw/add/remove)

Process directories recursively.

-a (applies to: add/remove)

Specify what actions to watch. One of edit, unedit, commit, all, none.

watchers--list watched files

- Requires: repository, working directory.
- Changes: nothing.
- Synonyms:

Display the list of files that are being watched, and what is being watched about them.

watchers options

-l

Process local directory only.

-R

Process directories recursively.

xdiff--External diff

- Requires: repository, working directory.
- Changes: nothing.
- Sysnonyms: xd

Run an external diff defined by the cvswrappers file on the server. The output and options for this option vary depending on what is run on the server-side diff, however the common options are listed below.

xdiff options

-D date

Diff revision for date against working file. Specifying the -D option twice causes the diff to be against the two dated revisions instead of the working file.

-N

Also diff added and removed files.

-R

Process directories recursively.

-l

Process local directory only.

-o xdiff-options

Pass extra arguments and options to the external xdiff program.

-r rev

Diff revision or tag against working file. Specifying a second -r option causes the diff to be against two specified revisions instead of the working file.