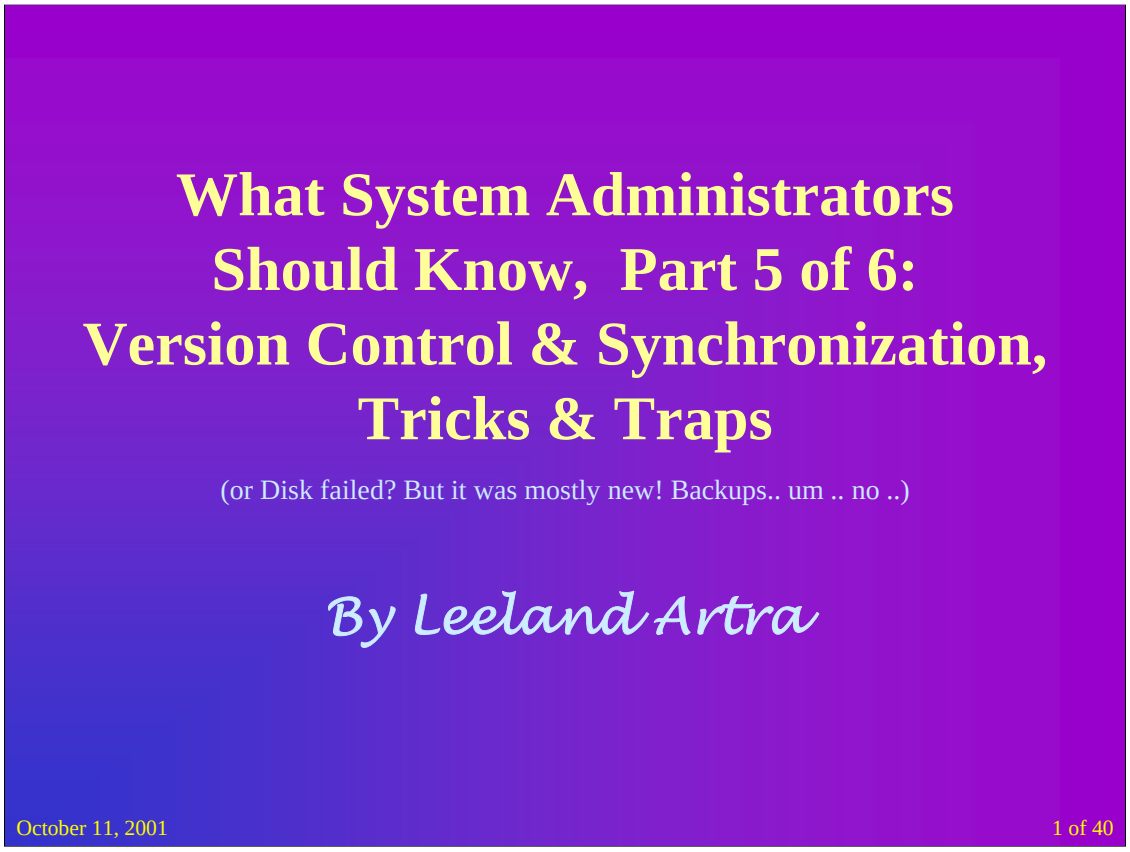


Source code control systems would seem to be a natural tool for systems administration. Yet, most SAs don't use any kind of source code control. After years of not using it myself I started using CVS (Concurrent Versioning System) which is based on the old BSD RCS source code control system. CVS has many unique and key features. There are others (BitKeeper for one) but I keep coming back to CVS.

In the distant past I slapped together a well met short introduction talk to CVS. I dusted that talk off and gave it a poor revamping in less then an hour creating a truly bad talk. But, later Jeremy Mates gave another good talk on CVS. These talks have been requested enough and seem very appropriate for the current "focused series on systems administration" we are doing. So I am dusting off my original (not too bad), the emergency rewrite (pretty poor as it was slapped together in an hour), Jeremy's presentation and adding in all the requested "advanced topics" items to create a new presentation on CVS uses for Systems Administration (and other cool tricks). This will only breifly introduce CVS before moving into examples and HOWTO topics. I have been doing some real (as in actually setting up new repositories and trying things out) research into uses and abuses of CVS.

This presentation will include a brief introduction (it is not a complex tool) to a very good directory synchronization tool that works cross platform, on the same system and remotely through SSH (without requiring root access). This useful little tool is call "unison".

Plus detailed descriptions on how to tie everything together using only SSH and user level access... mmmm and security too...

Version 1.0 – October 11th at 7:00 PM Seattle SAGE Group at the Rosen Building, 960 Republican Street, Seattle

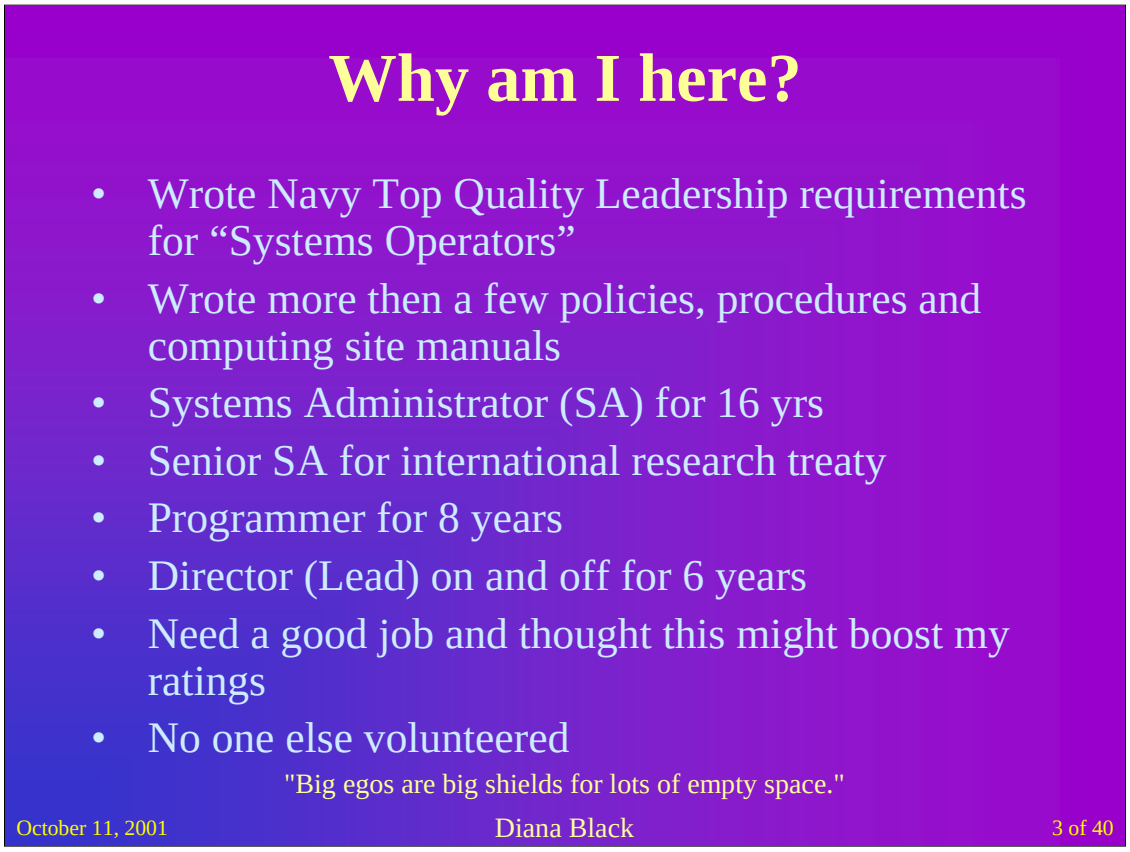
Why Are You Here?

By the end you should:

- Understand what synchronization and version control tools can do for systems administration
- Have an idea of what tools are available
- Understand the need for configuration control
- Know where to go for more detailed information

"We must have strong minds, ready to accept facts as they are."
Harry S Truman (1884-1972)

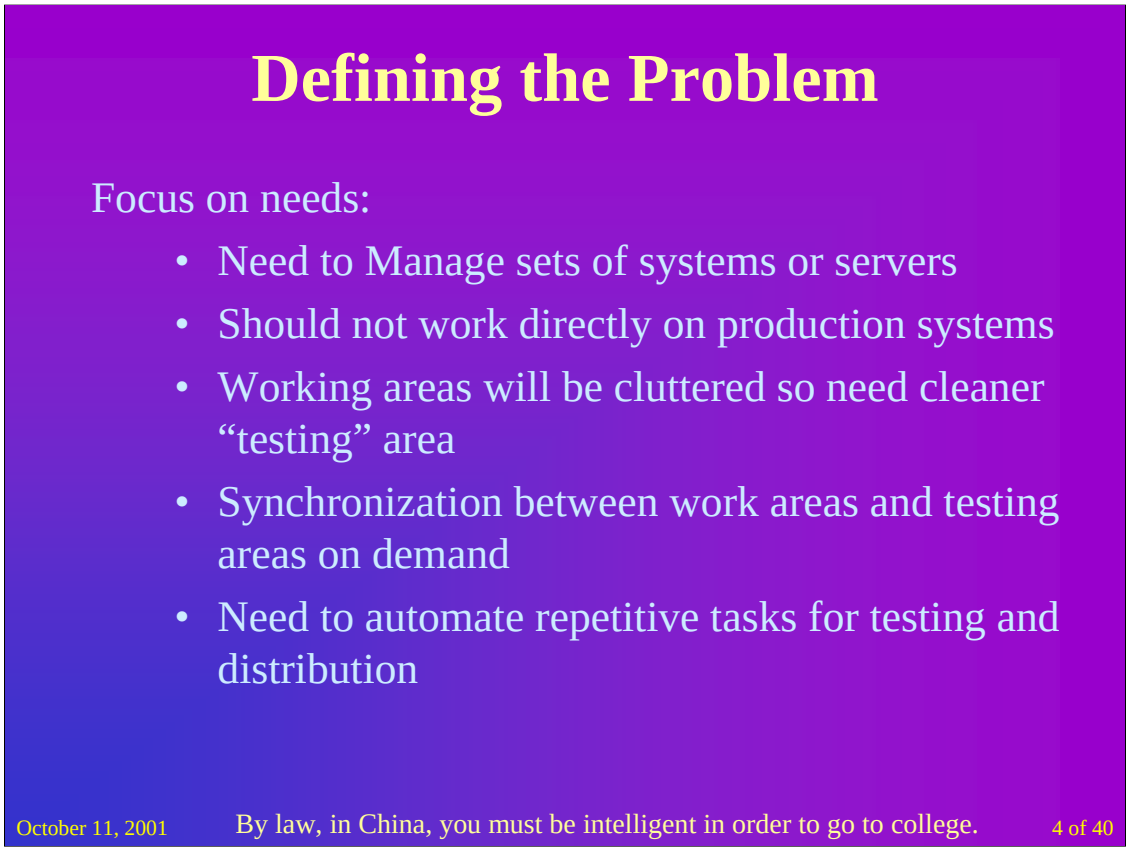
Note: The “Things SAs Should Know” series has spawned a technical white paper that describes in more detail each of the documentation methods discussed. Plus the paper includes examples and bibliographic references.



Leeland Artra is the dead "Director, Computer Systems Technology" walking for the Cell System Initiative (CSI) at the University of Washington. CSI is a combined biological research program and information systems research & development program focused on bringing the power and technologies available in today's computer industry to the biology research laboratory environment. The primary goal of the information systems research component of CSI is to develop an optimal computing environment for collaborating, distributed groups of biologists.

Previously Leeland was the Senior Researcher for the Cellworks Project at the University of Washington (CWP). The CWP has nearly the same goals as CSI and at CWP Leeland was the inventor of numerous database Java technologies. One of which was the Moulage System, which was highly recognized as an important research step in data mining and sharing. The Moulage System was nominated by Oracle for the Computer World Smithsonian 1998 award and became one of the finalists for that award. Leeland was also named an Oracle Futures Development Partner.

Leeland's current efforts are focused on looking for a new job as CSI failed to get the grants necessary to support its development efforts. Previously his efforts were focused on creating new Java based object oriented database technologies and architectures to store, access, mine, and analyze information.



Before we even start thinking about the tools to use, it is important to understand the problem to solve.

Think this through. You have a set of systems or desktops. You need to maintain these computers. Maintaining the systems involves adding new files, editing existing files, and removing (or moving) outdated files.

You probably don't want to be editing files on your "live" production servers. Firstly, this is dangerous, because any mistakes you make are immediately visible to the world. Secondly, if something really goes wrong your whole site might shutdown. So you will need a work area, in which you can edit your site, experiment with new content and new file organization for the site before they go live.

This work area will probably be cluttered with these experimental versions of your site, so you probably also need a test area (or staging area). When you want to test your site you will have to arrange to copy the appropriate files from your work area to the staging area. When you are happy with the contents of the staging area you will need a way to copy files from the staging area to the live site. This might also involve removing files from the live site.

As well as your work area and staging area, you will need somewhere to store old copies of files that have been on your site. This might be older versions of scripts or jobs that you no longer use, or different configuration designs that have been used across your site. Or perhaps you have made some changes on your site that will only last for a short while (maybe there are on site consultants who need the systems set up a particular way for a series of tests) and then you need to set everything back the way it was. We'll call this archive area "the repository".

Finally, you will probably find that there are a lot of repetitive tasks that you carry out when you check your site before making it live. For example, double-checking that all the internal links work, or validating your scripts. I've found those to be the bare minimum for maintaining a site. For small sites you can get away with not having a staging area, and test things in your work area as well. I find that quickly becomes unwieldy, and since small sites have a habit of becoming quite large quite quickly it is better to try and make things manageable from the start.

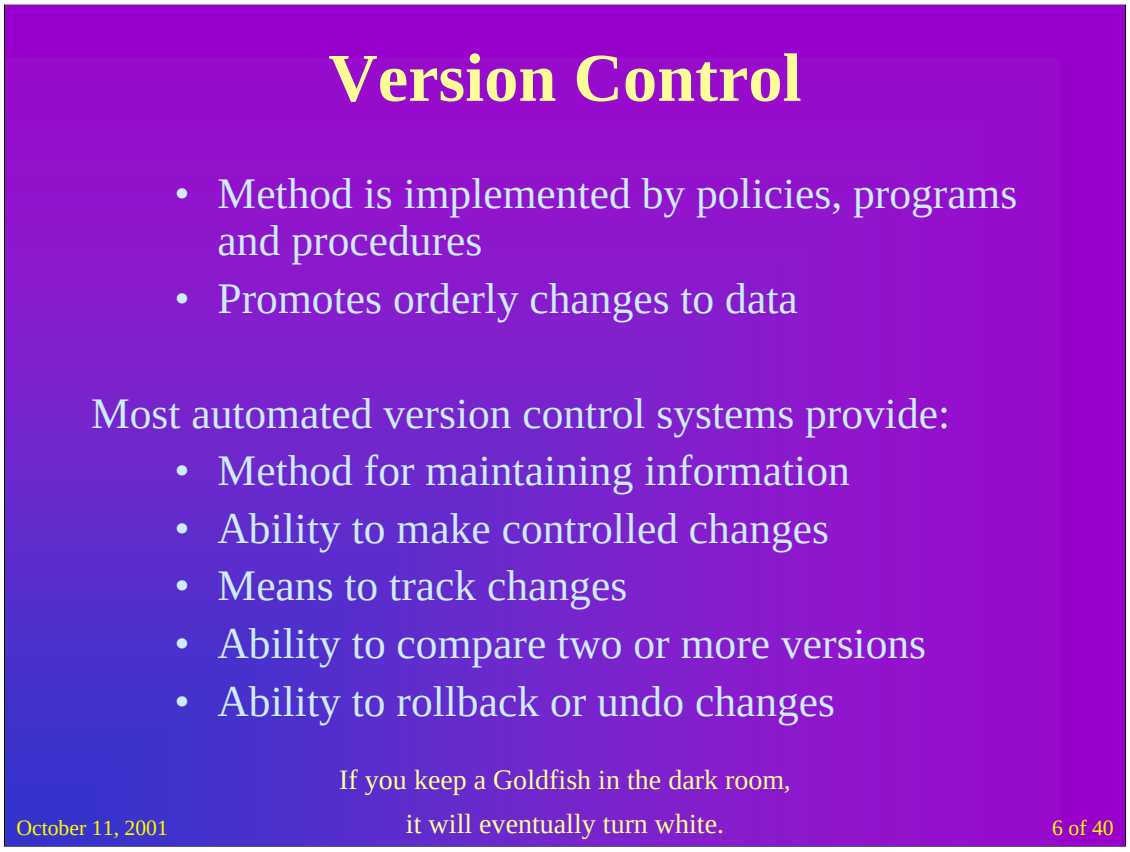
The Requirements

- A repository for all the versions of your files
- A work area, to make changes in
- A staging area, to test in
- A way of copying different versions of files from the repository to the work area
- A way of copying files from the work area to the staging area
- A way of copying files from the staging area to the live site
- A way of automating certain tasks (like checking for misconfigurations)

"One of the symptoms of an approaching nervous breakdown is the belief that one's work is terribly important."

Bertrand Russell (1872-1970)

So the actual requirements for a configuration control system in a production systems environment are pretty clear. For the most part this constitutes the minimum set of requirements. You might be able to add a few more for site and business specific needs.



Version Control is a method, implimented by policies, programs and procedures, that promotes orderly change of data and programs.

Automated Version Control provides several capabilities:

- A method for maintaining information.
- The ability to make controlled changes to information.
- A way to track what changes were made and by whom.
- Methods for comparing any two versions of a data resource.
- A means to rollback or undo changes.

Most Automated Version Control Systems operate on text files. Which can include:

- html documents,
- script files,
- software documentation,
- program source code,
- plain text documents,
- specially formatted documents, and
- other ASCII files.

It may also allow controled change of non-text files, such as program libraries, binary-formatted text (e.g., a word processing file), and other binary files.

Implementing version control will provide defined methods for an entire team to use; allowing everybody to operate under the same 'ground rules'. This will thereby keep all changes orderly vs. chaotic, saving development time. By providing the ability to track changes accountability will be promoted. Thus, making it easier to find the right person to solve problems in the maintained resource. Automatic maintenance of a list of exact changes made which can be retrieved quickly and easily, making it easier to advise users of the information on how it has changed from version to version. It is easy to 'roll back' to an earlier version of the information, if a serious mistake was made during a change.

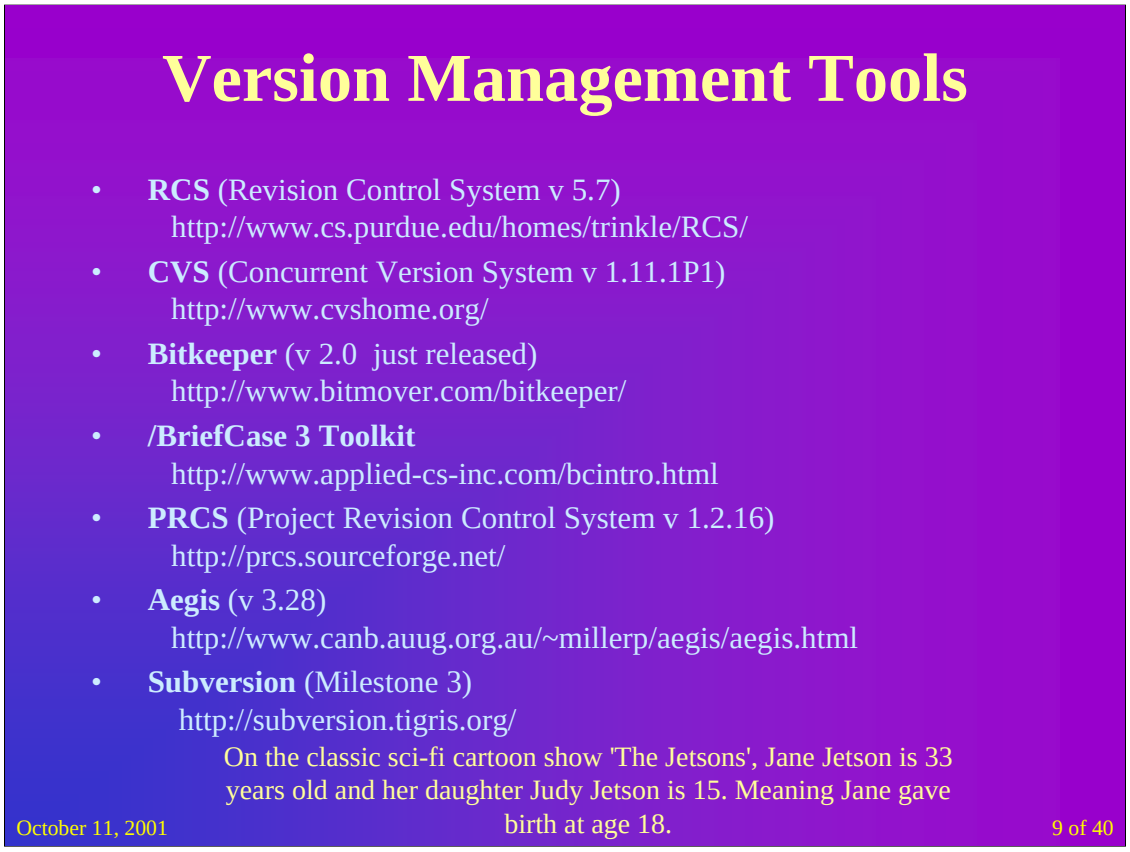
Now Lets Talk Tools

Types:

1. Version Management
   - Track changes to files
2. Synchronization
   - Push or Pull to create mirrors
   - Usually unidirectional
3. Home Grown
   - Does just what you need and no more

"Money can't buy happiness; it can, however, rent it."
Anonymous

Version control tools help multiple users to make simultaneous changes to a collection of documents/files, without clobbering each others' work or resulting in version confusion. Version control is essential for software development, and is often used for managing web sites, documentation, engineering drawings, corporate legal and business documents, and other documentation which must be archived and controlled.



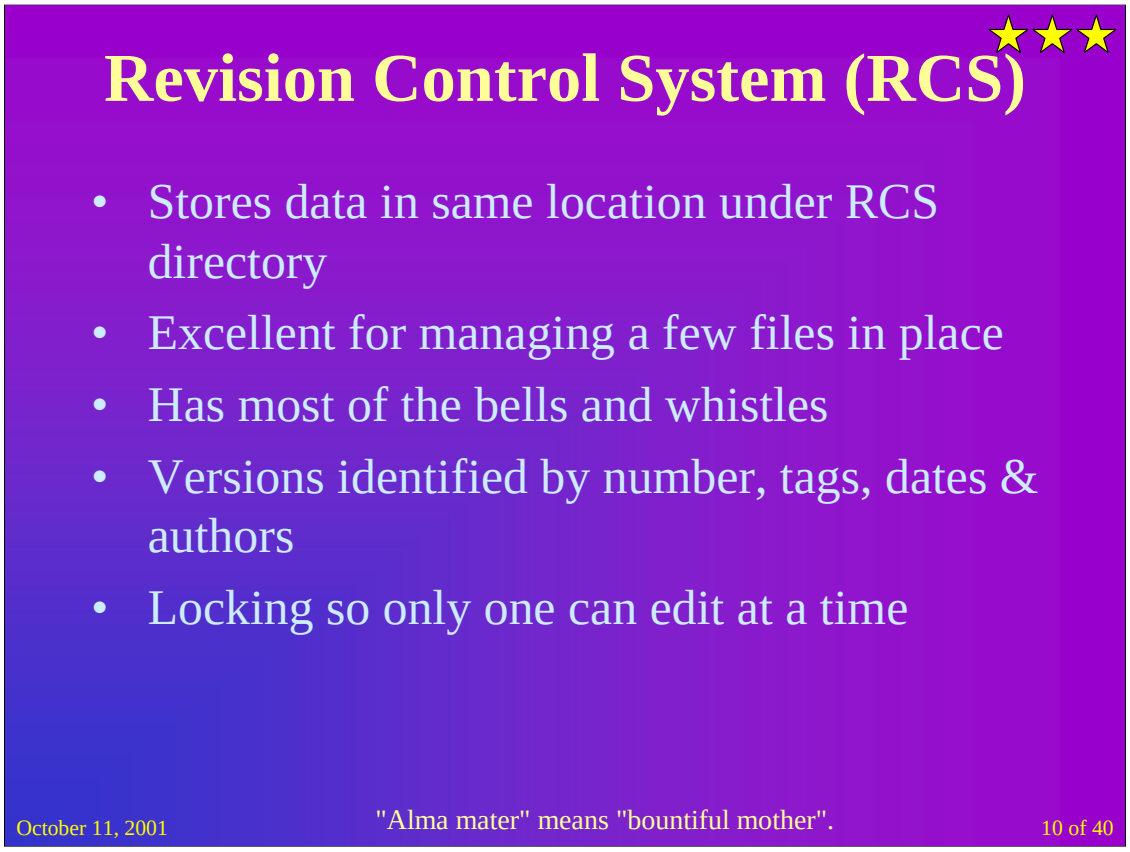
CVS is the Concurrent Versions System, the dominant open-source network-transparent version control system. The Revision Control System (**RCS**) manages multiple revisions of files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, e.g., programs, documentation, graphics, papers, and form letters.

CVS is useful for everyone from individual developers to large, distributed teams: its client-server access method lets developers access the latest code from anywhere there's an Internet connection; Its unreserved check-out model to version control avoids artificial conflicts common with the exclusive check-out model; Its client tools are available on most platforms.

BitKeeper is a scalable configuration management system that runs on most Linux and Unix varieties, Windows98/NT/W2K/ME and supports: Geographically distributed development; Scalable to 1000's of developers; Disconnected operation; Compressed repositories; Excellent merging tools; Change sets; Multi-protocol (FS/RSH/SSH/HTTP/BKD/SMTP) connectivity; and Named lines of development (available Q4 2001).

PRCS, the Project Revision Control System, is the front end to a set of tools that (like CVS) provide a way to deal with sets of files and directories as an entity, preserving coherent versions of the entire set. PRCS was designed primarily by Paul N. Hilfinger, with input and modifications by Luigi Semenzato and Joshua MacDonald. PRCS is written and maintained by Joshua MacDonald. Its purpose is similar to that of SCCS, RCS, and CVS, but (according to its authors, at least), it is much simpler than any of those systems.

Aegis is a transaction-based software configuration management system. It provides a framework within which a team of developers may work on many changes to a program independently, and Aegis coordinates integrating these changes back into the master source of the program, with as little disruption as possible.



Uses a method to save revision and data in a space efficient way.

Changes do not destroy the original.

Revisions can be retrieved according to ranges of revision numbers, symbolic names, dates, authors, and states.

One user restriction for modification. However, RCS allows read access when two individuals request same file. Can notify the individuals of who is working on a resource.

Control releases and configurations. Revisions can be assigned symbolic names and marked as released, stable, experimental, etc. With these facilities, configurations of modules can be described simply and directly.

RCS needs little extra space for the revisions (only the differences). If intermediate revisions are deleted, the corresponding changes are compressed accordingly.

RCS Basic Commands:

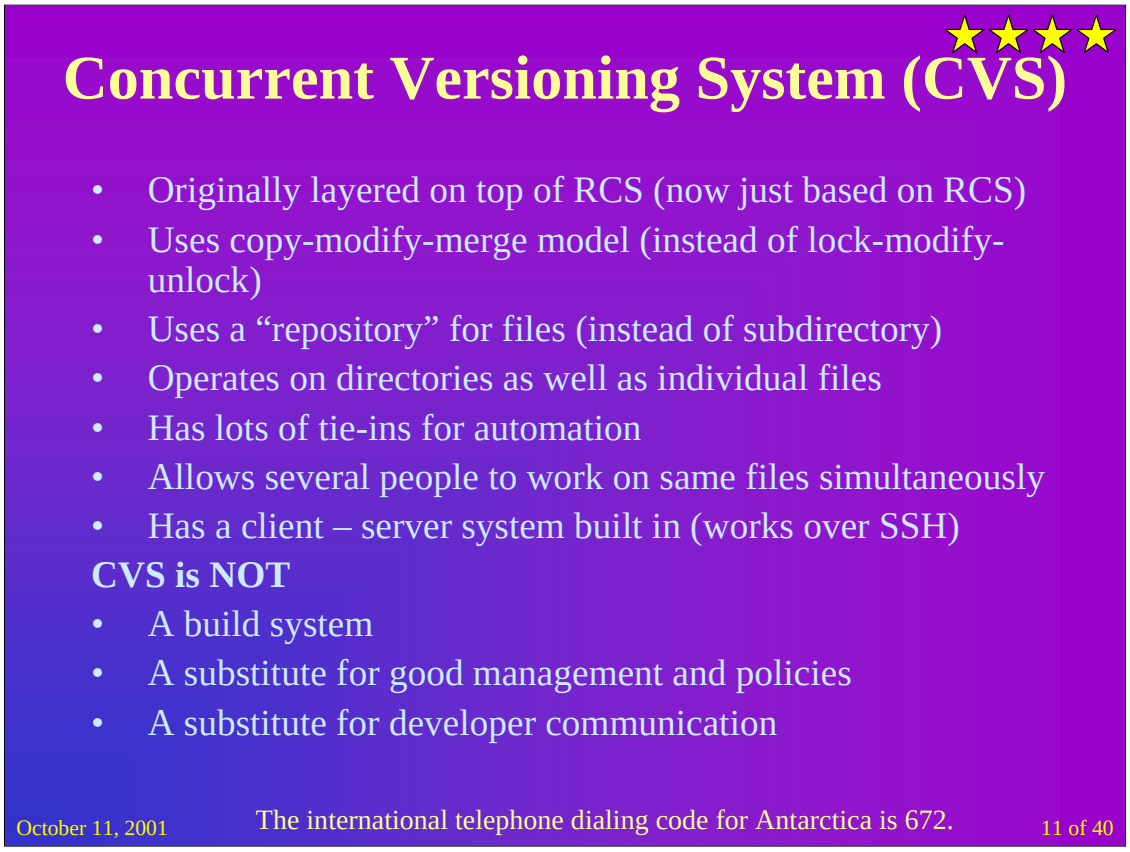
•**ci file** - Check in a file under RCS.

•**co -l file** - Check out a file for editing

•**co file** - Check out a file for read-only access

•**rcsdiff file** - Show differences between a file and reference (repository) copy

•**rcs -l file** - Lock changes to a file (if checked out) or lock the file so only you can check it out (if it is not already checked out)



CVS is layered on top of RCS. RCS or SCCS use a lock-modify-unlock model CVS uses a copy-modify-merge model. Uses a repository to hold resources and revision histories. Operates on an individual file or directory or a directory tree. More than one user may change the same document at the same time, but the CVS program will alert each reviser when their changes conflict with changes made by other concurrent revisers of the same document. All functions of of Revision Control System (RCS) are retained, but locking and multiple updates are dealt with gracefully. CVS allows several developers to work on sources at once. CVS can work as a client-server system.

CVS is usually implemented as a friendlier interface to RCS.

CVS is a large and complex system that is very powerful. To be able to use all of its capabilities well, you should take time to read the manual. This presentation provides only a basic overview of CVS and how to use it. Consult the manual (and the man pages) for more extensive information.

One of the strong points about CVS is that it not only lets you retrieve old versions of specific files, you can collect files (or directories of files) into "modules" and operate on an entire module at once. The RCS history files of all modules are kept at a central place in the file system hierarchy. When someone wants to work an a certain module he just types cvs checkout malloc which causes the directory `malloc' to be created and populated with the files that make up the malloc project.

With cvs tag malloc-1.0 you can give the symbolic tag malloc-1.0 to all the versions of the file in the malloc module. Later on, you can do cvs checkout -r malloc-1.0 malloc to retrieve the files that make up the 1.0 release of malloc. You can even do things like cvs diff -c -r malloc-1.0 -r malloc-1.5 to get a context diff of all files that have changed between release 1.0 and release 1.5!

CVS Basic Commands:

•cvs add [filename]: Adds a new file/directory to the repository

•cvs admin filename: Administration front end for rcs

•cvs checkout filename: Checkout sources for editing

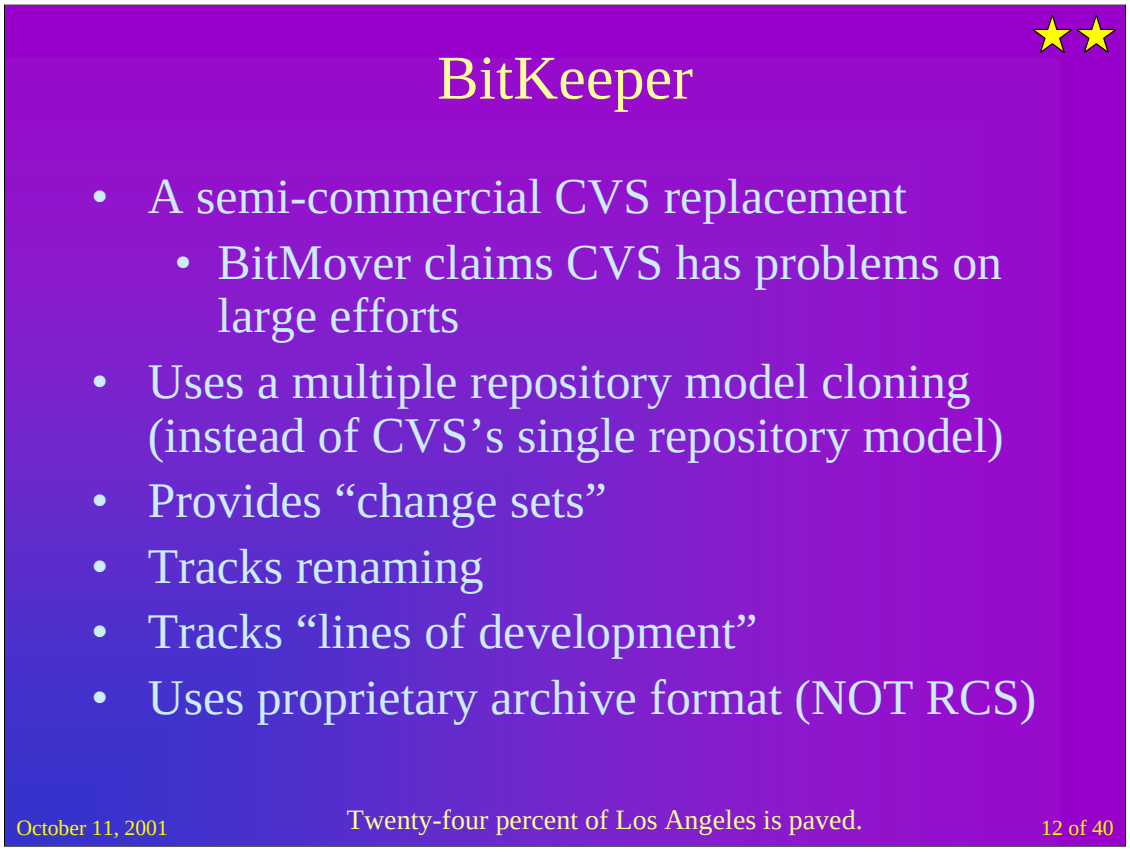
•cvs commit [filename]: Checks files into the repository

•cvs diff filename: Runs diffs between revisions

•cvs history [filename]: Shows status of files and users

•cvs status [filename]: Status info on the revisions

•cvs release [filename]: Status info on the revisions



The semi-commercial model means that you can use it if you let it report back to BitMover.

BitMover claims CVS has problems as the development effort gets large. But CVS is used successfully for the entire BSD and Linux developments.

CVS has a single repository model. Each work area is clear text only which means no revision control in the work area during development.

BitKeeper provides staging areas. You can mimic CVS by having one master repository and several work areas. You can also extend that to have one master and several staging areas with several work areas below each staging area. This allows people working on related projects to merge amongst themselves before merging into the master. Anyone who has lived through a change that broke the build can see the value of staging areas.

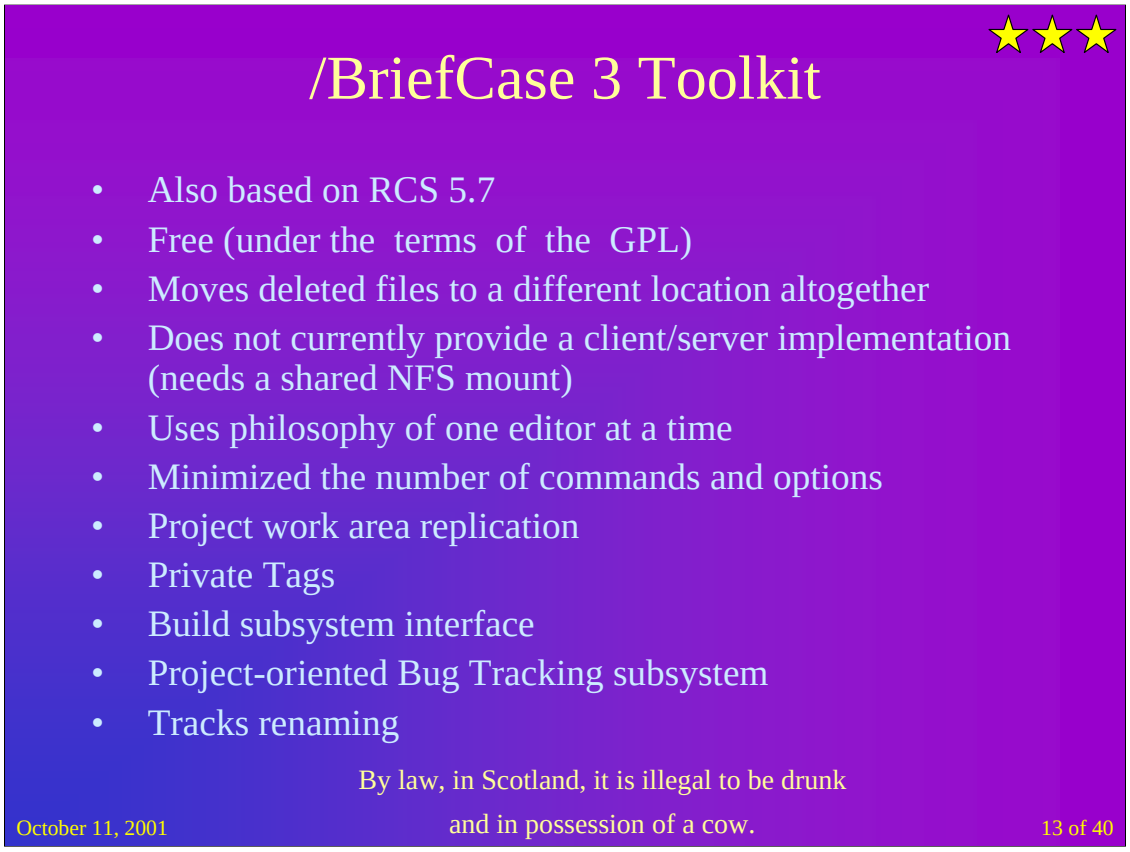
CVS has no change sets.

CVS has no rename support.

CVS has no lines of development.

CVS was based on RCS and still has RCS' limitations.

On the plus side, CVS is free, works well enough for some development projects, and CVS repositories are easily converted to BitKeeper. If you can't afford a good source management product, use CVS, we'll help you migrate off of it when the time comes.

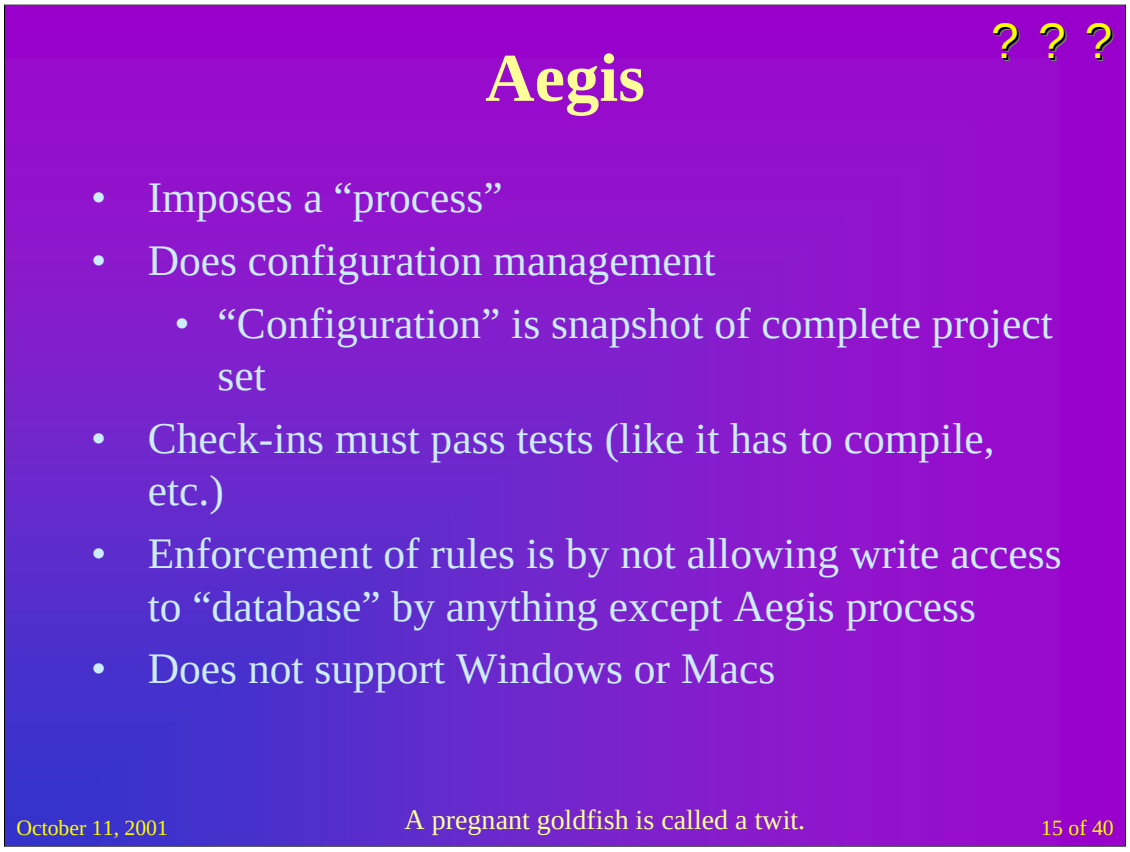


/BriefCase client scripts run their server-side counterparts with rsh and use a combination of pipes and an NFS mounted (rw) staging area (/Stage) to move information between the client and server hosts.

Project Revision Control System (PRCS) ? ? ? ?

- Also based on RCS 5.7 (loosely)
- Free
- Does not currently provide a client/server implementation
- Minimized the number of commands and options
- Execution subsystem interface
- Tracks renaming and moving
- Atomic operations on project versions
- Does not support Windows or Macs

PRCS, the Project Revision Control System, is the front end to a set of tools that (like CVS) provide a way to deal with sets of files and directories as an entity, preserving coherent versions of the entire set. PRCS was designed primarily by Paul N. Hilfinger, with input and modifications by Luigi Semenzato and Joshua MacDonald. PRCS is written and maintained by Joshua MacDonald. Its purpose is similar to that of SCCS, RCS, and CVS, but (according to its authors, at least), it is much simpler than any of those systems. This page is where information on the latest developments in PRCS can be found. The current release of PRCS is 1.2.16. Version 1.0 was released on September 3, 1996.



Aegis is a transaction-based software configuration management system. It provides a framework within which a team of developers may work on many changes to a program independently, and Aegis coordinates integrating these changes back into the master source of the program, with as little disruption as possible.

•Aegis sets out to perform configuration management. This is very different to version control, but to see why you’ll need to understand what Aegis thinks a configuration is. Also that ‘‘management’’ word implies a degree of authorization and reporting, which Aegis also provides.

•A configuration is a snapshot of the source files, all source files in the project, immediately after a commit. A single identifier can be used to recreate all source files for any one commit, and it isn’t a date. (It’s like a CVS tag, but different. It carries more information and exists long before the commit.)

•The complete audit path for every file is available from this single identifier.It includes who and when, with change set comments, not simply comments attached to each file which just happen to be the same.

•Implicit in this, is that a commit is performed for a set of files, not one file at a time.

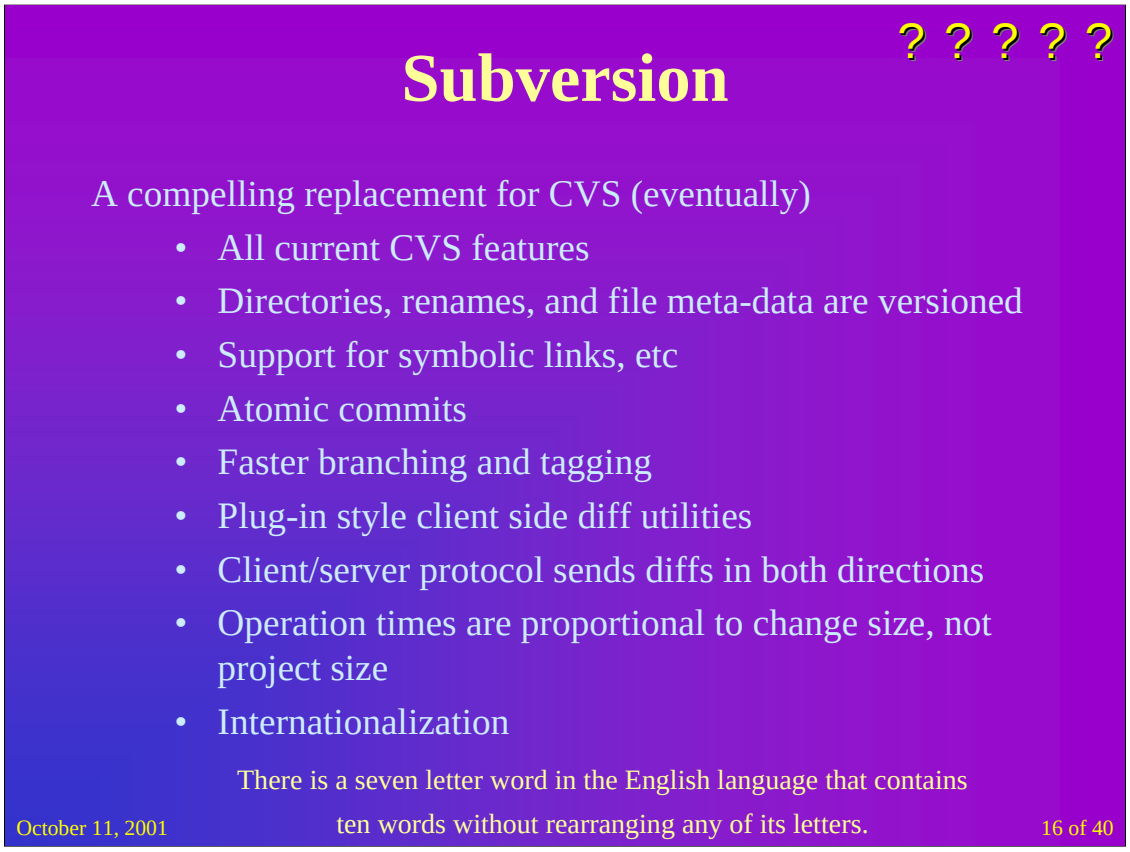
•For each commit, the files are not simply plonked into the repository.They must also compile successfully, they must also pass tests. Just as some database systems validate transactions before allowing a commit, so does Aegis require validation.

•In order to enforce this validation, no-one on the project has write permission to the repository. Not even the staff authorized to commit changes.

•The only method of changing the repository is through Aegis, and this is always validated, and always peer reviewed, and it always leaves a record.

•To support all of this, and private work areas too, Aegis imposes a process. It is said that all software development uses a process, but it usually isn’t written down. Aegis provides a process, but it is very flexible and can be easily adapted for your project’s needs, from single person projects to huge team projects.

Version control is an important part of this process, but in Aegis it is like the foundations of a house: essential, but usually unattractive and largely out of sight. Aegis delegates as much as possible to other tools, both because they already exist and don’t need re-writing, and also to provide developers with maximum choice. One of the easiest ways to think of Aegis, to misquote the X11 folks, is Aegis is about rules, not tools, because the rules (the process) is the ‘‘missing bit’’ Aegis was written to provide. The "management" part of software configuration management.



The goal of the Subversion project is to build a version control system that is a compelling replacement for CVS in the open source community. The software is released under an Apache/BSD-style open source license, and will have the following features:

•All current CVS features - CVS is good, as far as it goes, so we want to keep feature-compatibility: versioning, folding of non-conflicting changes, detection of conflicting changes, branching, merging, historical diffs, log messages, line-by-line history (cvs annotate), etc.
Generally, Subversion's conceptual interface to a particular feature will be as similar to CVS's as possible, except where there's a compelling reason to do otherwise.
Directories, renames, and file meta-data are versioned - Lack of these features is the most common complaint against CVS -- basically, CVS only versions file contents. Subversion will handle directory changes, file renames, and permission and other meta-data changes as well.

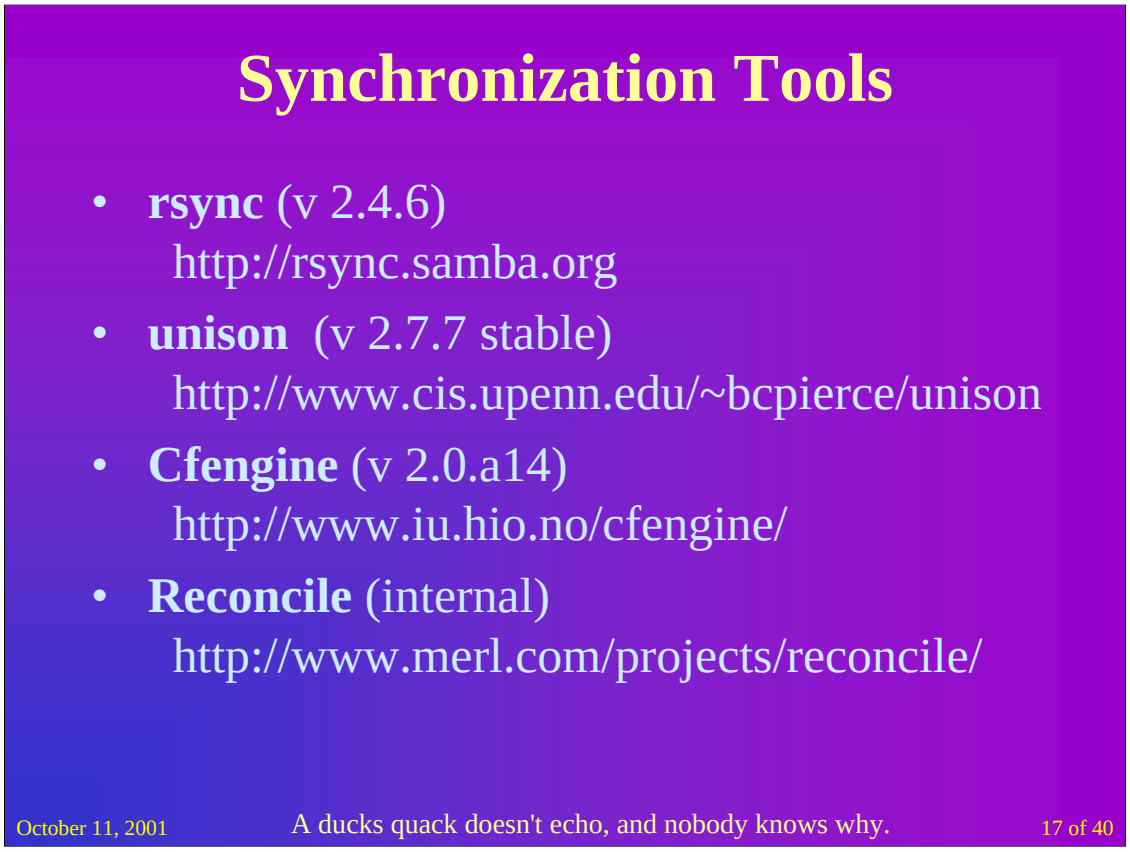
•Symbolic links, etc, are supported - Subversion will handle symbolic links ("shortcuts"), multiple hard links, and other special file types as long as their semantics are compatible with version control.

•Commits are truly atomic - No part of a commit takes effect until the entire commit has succeeded. Revision numbers are per-commit, not per-file.

•Branching and tagging are cheap (constant time) operations - There is no reason for these operations to be expensive, so they aren't. Branches and tags will both be implemented in terms of an underlying "clone" operation. A clone is just an alias, optionally within the project's namespace, pointing at a specific revision of an existing project. An clone takes up a small, constant amount of space. All clones are tags; if you start committing on one, then it's a branch as well. (This does away with CVS's "branch-point tagging", by removing the artificial distinction that made branch-point tags necessary in the first place.)

•Repeated merges are handled gracefully - Subversion will have a way of remembering what has been merged, so that repeated merges from the same source do not require careful human calculation to avoid spurious conflicts (anyone who's done repeated CVS merges knows what we're talking about). (There are some theoretical problems with remembering merge sources -- knowing where the merged data came from implies some sort of universal repository registry. However, our first goal is to make sure that multiple merges from branches made in the same repository as the original project compound gracefully. Remembering merges from remote sources is more difficult, due to the difficulty of distinguishing remote sources, but there are good "90%" solutions that will work in practice).

•Support for plug-in client side diff programs - Subversion knows how to show diffs for text files, and also gives the user the option to plug in external diff programs for any kind of file. The external program need merely conform to some simple

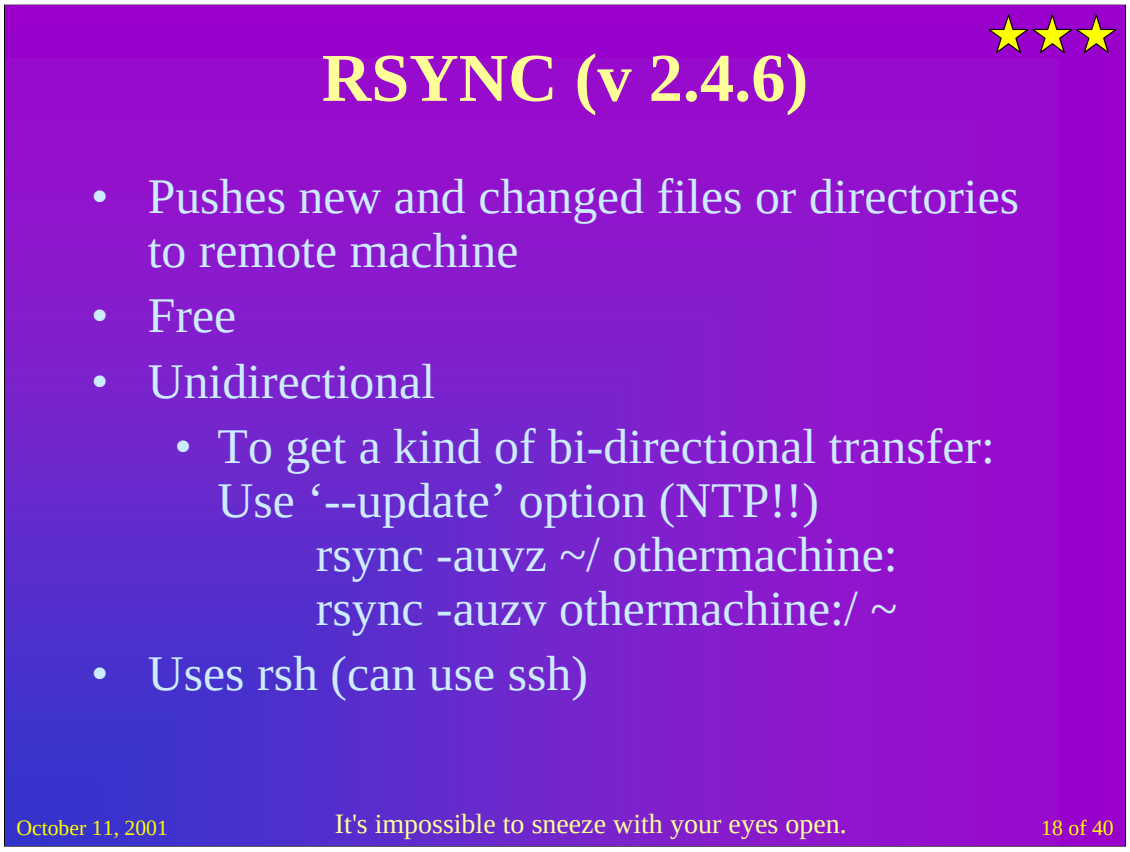


Cfengine, or the configuration engine is an agent/software robot and a high level policy language for building expert systems to administrate and configure large computer networks. Cfengine uses the idea of classes and a primitive intelligence to define and automate the configuration and maintenance of system state, for small to huge configurations. Cfengine is designed to be a part of a computer immune system, and can be thought of as a gaming agent.

rsync is an open source utility that provides fast incremental file transfer. rsync is freely available under the GNU General Public License.

Unison is a file-synchronization tool for Unix and Windows. It allows two replicas of a collection of files and directories to be stored on different hosts (or different disks on the same host), modified separately, and then brought up to date by propagating the changes in each replica to the other. Unison shares a number of features with tools such as configuration management packages (CVS, PRCS, etc.) distributed filesystems (Coda, etc.) uni-directional mirroring utilities (rsync, etc.) and other synchronizers (Intellisync, Reconcile, etc).

Reconcile maintains consistent copies of files on different computers such as laptop computers, desktop workstations, and servers. By keeping a complete set of all files at each site and cross-updating only on demand, it gains several major advantages over other distributed file systems: users can work disconnected for days or weeks, then reconnect and safely update all their files; connectivity is required only when reconciling; network and server loads are minimized; users control when updates happen; and any computer can be a server or backup.



rsync uses the "rsync algorithm" which provides a very fast method for bringing remote files into sync. It does this by sending just the differences in the files across the link, without requiring that both sets of files are present at one of the ends of the link beforehand. At first glance this may seem impossible because the calculation of diffs between two files normally requires local access to both files.

USAGE

Basically you use rsync just like rcp, but rsync has many additional options.

Here is a brief description of rsync usage:

Usage: rsync [OPTION]... SRC [SRC]... [USER@]HOST:DEST

or rsync [OPTION]... [USER@]HOST:SRC DEST

or rsync [OPTION]... SRC [SRC]... DEST

or rsync [OPTION]... [USER@]HOST::SRC [DEST]

or rsync [OPTION]... SRC [SRC]... [USER@]HOST::DEST

or rsync [OPTION]... rsync://[USER@]HOST[:PORT]/SRC [DEST]

SRC on single-colon remote HOST will be expanded by remote shell

SRC on server remote HOST may contain shell wildcards or multiple sources separated by space as long as they have same top-level

To do a transfer in each direction, giving the --update option so that newer files are not overwritten. (If you rely on this, then you'll want to run NTP or something similar to make sure the machine clocks are reasonably correct.)

rsync -auvz ~/ othermachine:

rsync -auzv othermachine:/ ~

RSYNC over SSH

- Use ‘-e’ option to specify rsh replacement:
  rsync -avz -e ssh ~/ othermachine:/
- Or set RSYNC_RSH
  - Can include arguments like:
    ssh -i ~/.ssh/rsyncU

"therein": the, there, he, in, rein, her, here, ere, therein, herein.

If you just one to copy files to or from the local machine, then you need something like this:

rsync -avz -e ssh directory userb@hostb:/other/dir

rsync always copies either to or from the remote machine; it can't have both the source and destination remote. However, you can send the whole command to a remote machine to execute there:

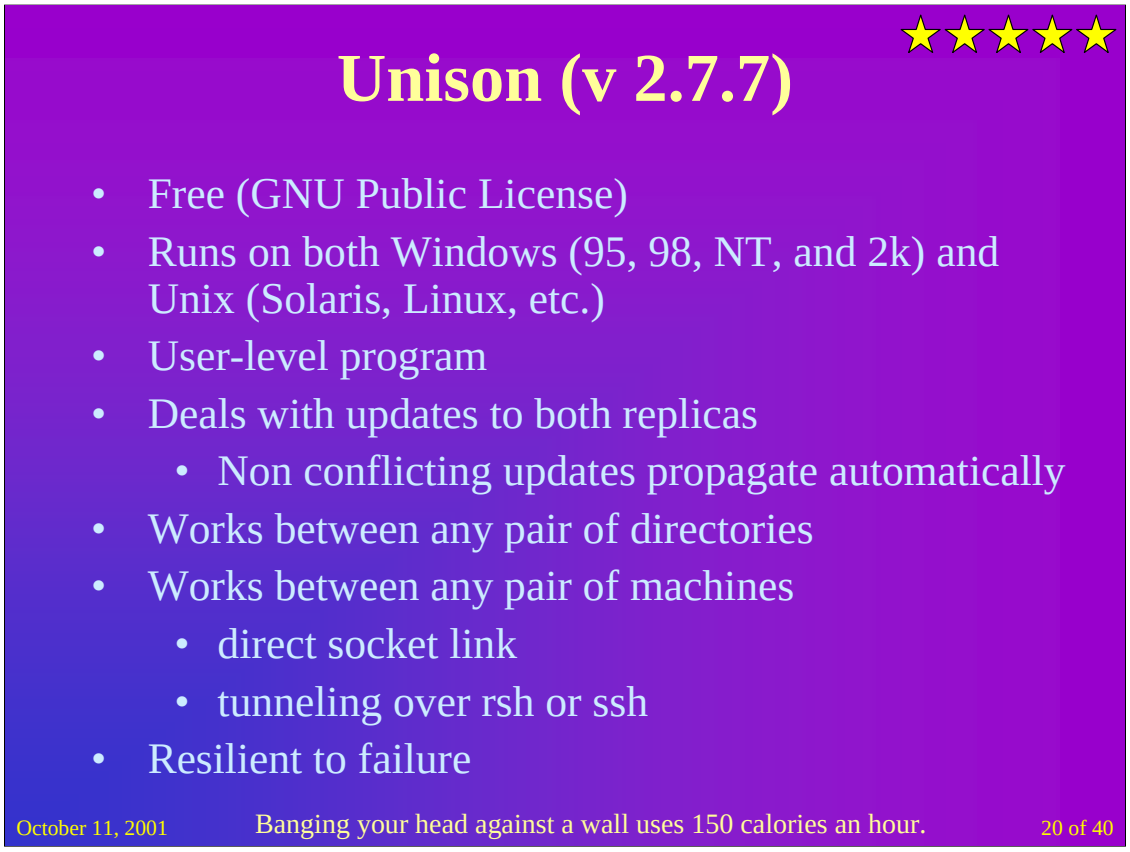
ssh usera@hosta rsync -avz -e ssh directory userb@hostb:/other/dir

rsync is uses an environment variable RSYNC_RSH that you can set to anything. It will use this program and any parameters provided to its commands. For example, say you're behind a firewall that doesn't let anything out except for Port 53. What you can do is setup your SSH server on the outside to also listen to 53 (as long as you don't have bind running). Then setup your environment on the intranet machine like so:

RSYNC_RSH='ssh -p 53'

Then run your command:

rsync -avz /a/bunch/of/stuff external.server.com:/dest/dir



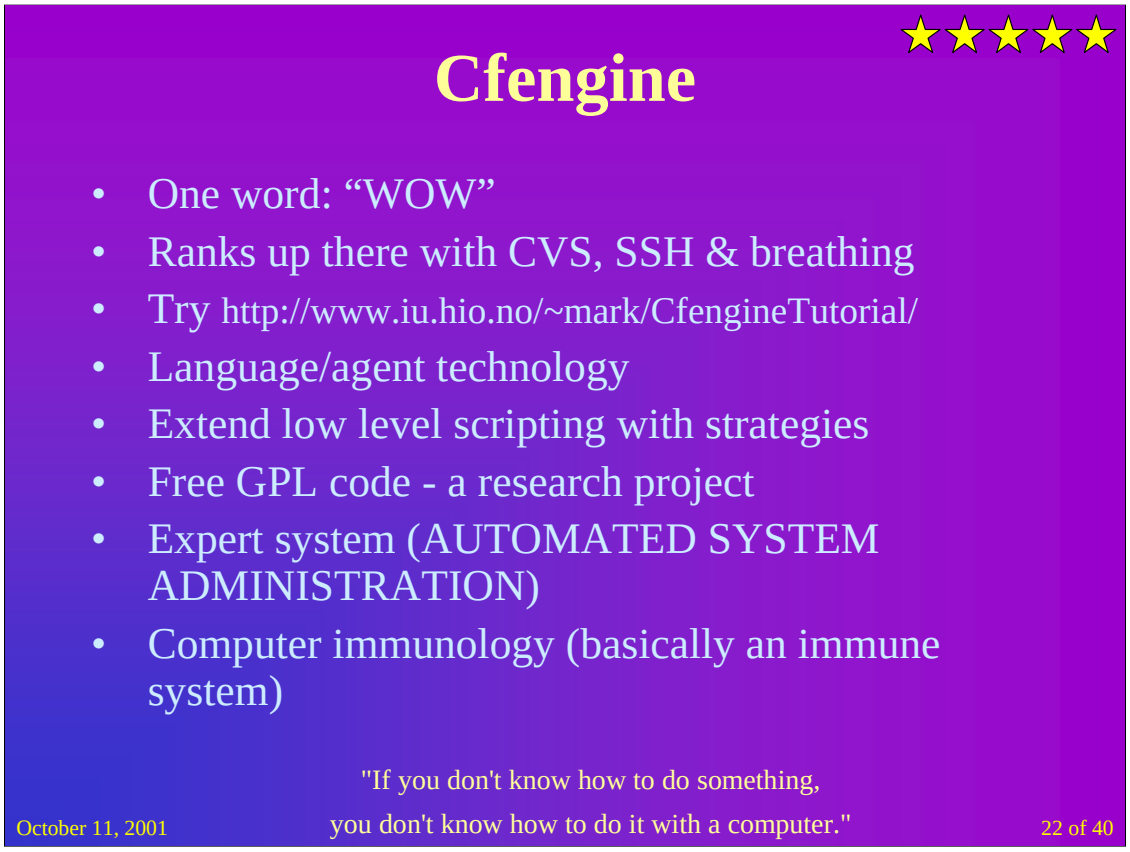
Unison is a file-synchronization tool for Unix and Windows. It allows two replicas of a collection of files and directories to be stored on different hosts (or different disks on the same host), modified separately, and then brought up to date by propagating the changes in each replica to the other. Unison shares a number of features with tools such as configuration management packages (CVS, PRCS, etc.) distributed filesystems (Coda, etc.) uni-directional mirroring utilities (rsync, etc.) and other synchronizers (Intellisync, Reconcile, etc). However, there are a number of points where it differs: Unison runs on both Windows (95, 98, NT, and 2k) and Unix (Solaris, Linux, etc.) systems (unison works across platforms, allowing you to synchronize a Windows laptop with a Unix server, for example); unlike a distributed filesystem, Unison is a user-level program: there is no need to hack (or own!) the kernel, or to have superuser privileges on either host; unlike simple mirroring or backup utilities, Unison can deal with updates to both replicas of a distributed directory structure. Updates that do not conflict are propagated automatically. Conflicting updates are detected and displayed; unison works between any pair of machines connected to the internet, communicating over either a direct socket link or tunneling over an rsh or an encrypted ssh connection. It is careful with network bandwidth, and runs well over slow links such as PPP connections; Transfers of small updates to large files are optimized using a compression protocol similar to rsync; Unison has a clear and precise specification; Unison is resilient to failure (it is careful to leave the replicas and its own private structures in a sensible state at all times, even in case of abnormal termination or communication failures); and

Unison is free (full source code is available under the GNU Public License).

Unison Example

```
# Unison preferences file
batch = true
log = true
times = true
prefer = newer
servercmd = bin/unison
rshargs = -i E:\home\.ssh\identityU
include ignore
root = E:\home\working
root = ssh://leeland@heronetwork.com/working
                    ----- cut here ------
ignore = Name {*~,.*~,.xvpics,*.o,*.tmp,tmp,temp,*.out}
```

A fish has a memory span of 3 seconds,
this explains why they move a lot.



Cfengine is a tool for setting up and maintaining BSD and System-5-like operating system optionally attached to a TCP/IP network. You can think of cfengine as a very high level language--much higher level than Perl or shell: a single statement can result in many hundreds of operations being performed on multiple hosts. Cfengine is good at performing a lot of common system administration tasks, and allows you to build on its strengths with your own scripts. You can also use it as a netwide front-end for cron. Once you have set up cfengine, you'll be free to use your time being like a human being, instead of playing R2-D2 with the system.

The main purpose of cfengine is to allow you to create a single, central system configuration which will define how every host on your network should be configured in an intuitive way. An interpreter runs on every host on your network and parses the master file (or file-set); the configuration of each host is checked against this file and then, if you request it, any deviations from the defined configuration are fixed automatically. You do not have to mention every host specifically by name in order to configure them: instead you can refer to the properties which distinguish hosts from one another. Cfengine uses a flexible system of "classes" which helps you to single out a specific group of hosts with a single statement.

Reconcile

? ?

- Mitsubishi Electric Research Laboratories
- Does everything Unison does (mostly)
- Still in research
- Requires you to become a collaborator to get a testing copy of it

"Any sufficiently advanced bureaucracy
is indistinguishable from molasses."

Reconcile maintains consistent copies of files on different computers such as laptop computers, desktop workstations, and servers. Keeps a complete set of all files at each site and cross-updating only on demand.

make(1) was originally written to help support programming projects. The source code to all but the most trivial of programs is typically stored in more than one file (foo.c, bar.c, and so on for C source code). To build the executable, each file must be compiled to an object file (foo.o, bar.o, ...). Finally, these object files must be linked to form the executable file (the command that you would run).

If you have changed just one .c file, you do not want to have to recompile all of them, and then relink. It would be a great time-saver if you could express in some way “Only recompile the .c files that have been modified since .o file with the same base name[1] was created, and then relink the program.”

This is very similar to the requirements outlined previously, and could be rewritten as “Only compile the services that have been patched since the binary file with the same base name was created.” or “Only signal the deamon with the same base name if the .config files that have been modified."

You express these requirements in rules contained within a text file that make(1) reads when it is run. This file is normally called makefile or Makefile.

Make Example

```
# Sample Makefile to install one file
CONFIGS =  resolv.conf hosts
DESTDIR ?= /etc
SUBDIR = dir1 dir2 dir3
install:
    @for subdir in ${SUBDIR}; do \
        (cd $$subdir; ${MAKE} all); \
    done
    for cfile in ${CONFIGS}; do \
        cp $$cfile ${DESTDIR}/$$cfile; \
        chmod 644 ${DESTDIR}/$$cfile; \
    done
```

The name of all the continents end with the same letter that they start with.

Time For Tricks

- CVS & SSH
- Anonymous CVS via SSH
- Multi-configuration synchronizations with Unison
- Silly Things That Work
- Ideas For What To Do Next

Shakespeare invented the words "assassination" and "bump".

CVS & SSH

Instead of

```
:pserver:leeland@24x7dns.com:/var/cvs
```

use

```
:ext:leeland@heronetwork.com:/var/cvs
```

- “ext” will default to “ssh”, so
- Now set CVS_RSH=cvs_ssh

```
#!/bin/sh
exec /bin/ssh -x -i ~/.ssh/cvsID $@
```

The symbol on the "pound" key (#) is called an octothorpe.

To sift CVS to SSH stop using pserver and instead use “ext”. This will default to ssh. But if you need to get some arguments to ssh (like an alternate key) create a little script and have it run ssh with all the right options.

A batch file version of the above for Windows would look like:

```
SET HOME=C:\HOME\leeland
C:\bin\ssh -x -i %HOME%\.ssh\cvsID %1 %2 %3 %4 %5 %6 %7 %8 %9
```

More CVS & SSH

- If you generate a special RSA key for ssh without a password you can eliminate lots of password typing
- You can use “no password” RSA keys for CVS GUIs (trust me its easier this way)
- On the CVS host make sure the cvs command is the only command that can be issued in authorized_keys:

  command="cvs server",no-pty,no-port-forwarding, no-X11-forwarding 1024 35 1126421….086379

Anonymous CVS via SSH

- Create a anonymous cvs user (acvs)
- Generate a no password ssh keypair
- Create batch ssh file (see notes)
- Edit CVSROOT/config and add a "LockDir=/some/dir“

CVS_RSH=~/bin/sshanoncvs \
cvs -d :ext:acvs@cvshost.net:/cvs/repository get foo

(See http://www.kitenet.net/programs/sshcvs/)

By law, in Orlando, FL, if you tie an elephant to a parking meter you have to feed the meter.

```
#!/bin/sh
# This program implements the client side of anonymous cvs over ssh.
# No more pserver! For details on how to use this and how it works,
# visit http://kitenet.net/programs/perlmoo/sshcvs/
#
# This is copyright 1999, 2001 by Joey Hess under the terms of the GPL.

# Just run ssh, telling it to use this file as the identity file.
# Cvs will tell it what host to connect to as what user and will
# proceed to communicate with the cvs server over the ssh link.
# We do need to fix up the perms of the script to something ssh
# will accept though, and put them back later.
chmod 600 $0
ssh -q -C -i $0 $@
chmod 755 $0
exit

# The remainder of the file is the "private" key that lets users into my
# cvs server. ssh 2 can find this key, ignoring the top part of the file.

-----BEGIN DSA PRIVATE KEY-----
MIIBuwIBAAKBgQCaMAHhVxLgJypYHUf2evLXhrG69LsXRmOwMztSnfA7yt2cruPm
bzd2rW7XOmcTmd1dtW3NKCGgknDCWFat8M/zCGRIuTIw0DDZUkbzLYj33le9qMRI
h9lpMFCctjv04gdjWk2LN5xf94n3+99dhlD97NVjzrZA77TkydGGYRvyFwIVAJ97
```

Multi-configuration synchronizations with Unison

- Figure out and make Unison configurations for which files / directories:
  - Almost never change
  - Change while working
  - Change frequently
  - Need to update when you move
- Create a non password ssh key pair
- Adjust authorized_keys:
  command="bin/unison -server",no-pty,\
  no-port-forwarding,no-X11-forwarding 1024 35 1..3
- Now make some simple scripts like: (umain)
  unison palm
  unison docs
  unison working

Silly Things That Work

- Use the “at” command to spawn a synchronization event on a tag change in CVS (loginfo):

```
^utils   (echo "\n ----- -----\n CVSWEB Repository update";/usr/bin/date;
at –f /www/bin/updateCVS now + 1 min; /usr/bin/cat) >>
$CVSROOT/CVSROOT/updatelog 2>&1
```

- Need input….

The male praying mantis cannot copulate while its head is attached to its body. The female initiates sex by ripping the male's head off.

If you use loginfo make sure your scripts LOCK somehow. You can easily get dozens of scripts trying to get started only the first needs to run so the rest just quietly bail.

More Silly Things That Work

- CVS Does not automatically know about MIME types so it treats binaries like text
  - To Fix this add binaries using the command: cvs add –kb file
  - Or Better Yet update the CVSROOT control file “cvswrappers” to be like:

    *.exe -k 'b'

    *.EXE -k 'b'

    *.rtf -k 'b'

    *.RTF -k 'b'

    *.HQX -k 'b'

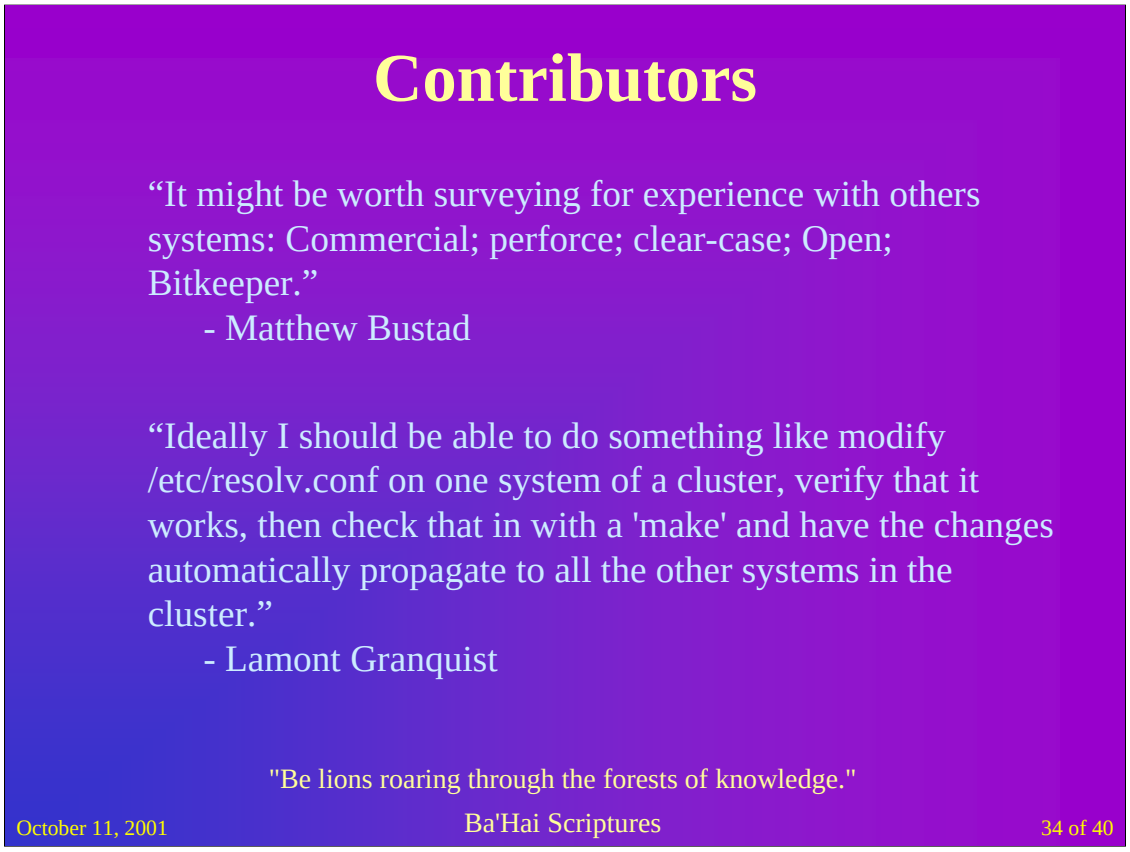
The only 15-letter word that can be spelled without repeating a letter is "uncopyrightable".


```
# This file describes wrappers and other binary files to CVS.
#
# Wrappers are the concept where directories of files are to be
# treated as a single file.  The intended use is to wrap up a wrapper
# into a single tar such that the tar archive can be treated as a
# single binary file in CVS.
#
# To solve the problem effectively, it was also necessary to be able to
# prevent rcsmerge from merging these files.
#
# Format of wrapper file ($CVSROOT/CVSROOT/cvswrappers or .cvswrappers)
#
#  wildcard     [option value][option value]...
#
#  where option is one of
#  -f           from cvs filter         value: path to filter
#  -t           to cvs filter           value: path to filter
#  -m           update methodology      value: MERGE or COPY
#
#  and value is a single-quote delimited value.
#
# For example:
*.exe -k 'b'
*.EXE -k 'b'
```

Ideas For What To Do Next

- Use the /etc/security script from OpenBSD as an example for creating a “configuration sanity checker”
- Build a set of “machine class” templates for cfengine and post them on the SAGE site

If done perfectly, any Rubix Cube combination can be solved in 17 turns.



From: "Matthew Bustad" <matthew.bustad@onebox.com>

Date: Mon, 08 Oct 2001 14:22:12 -0700

It might be worth surveying for experience with others systems.

Comercial; perforce, clear-case.

Open; Bitkeeper

--Matthew.Bustad@onebox.com

----- cut here -----

From: Lamont Granquist <lamont@scriptkiddie.org>

Date: Mon, 8 Oct 2001 22:07:34 -0700 (PDT)

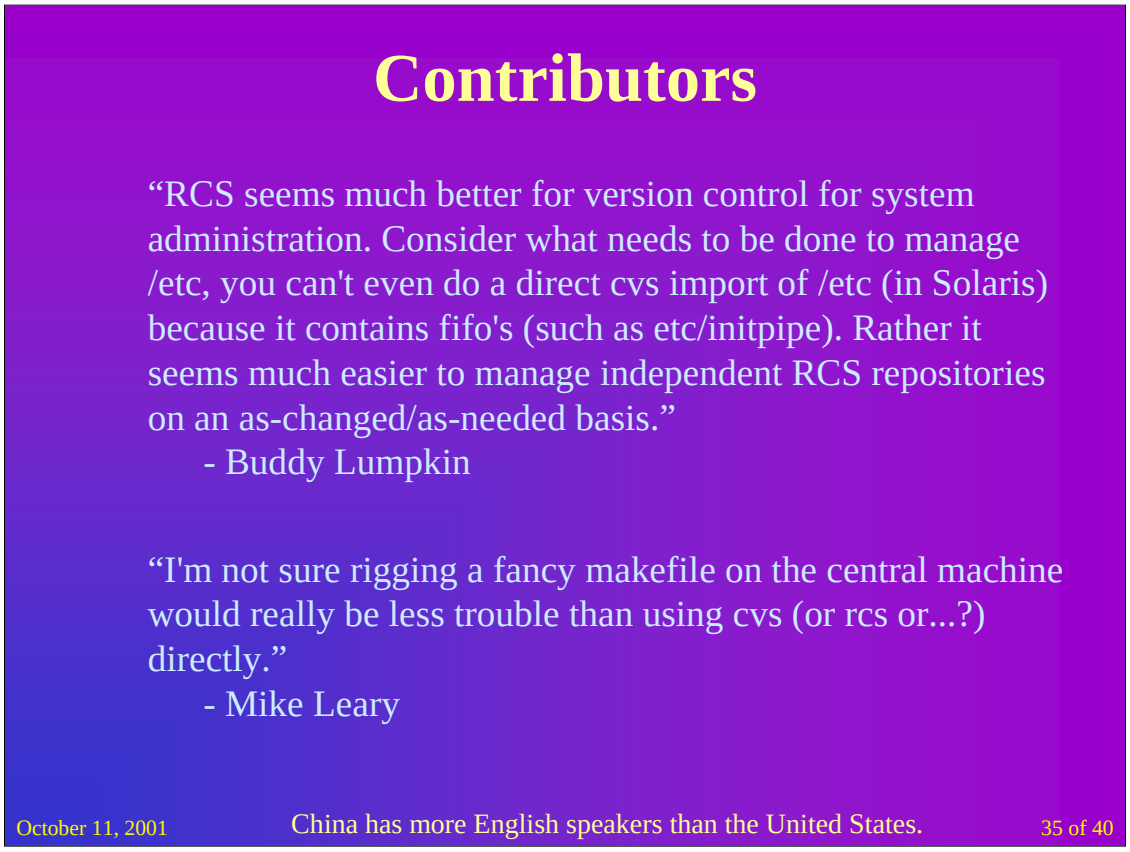
The biggest question I would have is how to use CVS to do change management on a system as a whole, without making it a royal pain in the ass to do updates to the system. At the last system admin job I had, we used CVS for awhile to track changes to system configuration, but the process was that you had to edit the file on the server, then copy it back and check it into the CVS repository. This process routinely got violated since it was too time consuming and we're all lazy^H^H^H^Hbusy system administrators.

So, how do you use CVS to setup a simple system whereby changes to system configuration files can easily be added to a CVS repository with something like a simple 'make' command?

And the corrollary is how can you get it so that changes to the CVS repository done with a checkin can be used to push changes out to other systems? Complete with SIGHUPing the appropriate daemons, and whatever else needs to be done to implement the changes?

Ideally I should be able to do something like modify /etc/resolv.conf on one system of a cluster, verify that it works, then check that in with a 'make' and have the changes automagically propogate to all the other systems in the cluster.

And these questions can be extended to the address the other things I would like to be able to easily do with a configuration management system implemented on top of CVS. I'd like to easily be able to create a file or set of files on a server and then have those files added to the repository and propogated out to other servers. I'd like an easy interface to be able to revert changes (again with propogation to other servers). I'd like to be able to verify the state of a system against the files in the repository. I'd like to easily be able to ingetrate it with systems like



From: "Buddy Lumpkin" <blumpkin@home.net>

Date: Mon, 8 Oct 2001 23:14:14 -0700

The biggest question I would have is how to use CVS to do change management on a system as a whole, without making it a royal pain in the ass to do updates to the system.

----------snip---------

This has been my experience too. RCS seems much better for version control for system administration. Consider what needs to be done to manage /etc, you can't even do a direct cvs import of /etc (in Solaris) because it contains fifo's (such as etc/initpipe). Rather it seems much easier to manage independant RCS repositories on an as-changed/as-needed basis.

Comparing ci and co to cvs login, et al. when hundreds of servers are being

managed. I think I will continue to use CVS for code projects and to manage

versions of docs and images and RCS for system administration change

management.

--Buddy

----- cut here -----

From: Mike Leary <leary@nwlink.com>

Date: Tue, 9 Oct 2001 16:14:35 -0700

I'm just guessing, but I would think a central machine (in addition to the configuration test machine ;) that propagated changed to the remote machines would work.

http://rsync.samba.org/rsync/index.html

http://www.magnicomp.com/rdist/

This was the kind of system in use at a dot-bomb I worked at where the sysads were quite the experts (really!).

Make a magic file(s) for the daemons that need to restart. Maybe a cron job on the remote machines to look

Quotes

- All the quotes came from my personal collection gathered from lots of sources.
- Some places to find more quotes are:

Witty Wisdom Quotes:

http://www.iol.ie/~taeger/wisdomqu/wisdomq1.html

Online Quotes:

http://www.idynamics.com/quotes/

Witty Quotes:

http://www.angelfire.com/ma/hubpoet/pquote.html

Witty, Thought-Provoking, and Humorous:

http://www.tk421.net/essays/wit.shtml

Funny Facts:

http://www.jokersweb.com/

The electric chair was invented by a dentist.

Bibliography

"Using CFENGINE to Maintain Systems and Security", by Mike Lang, http://www.sans.org/infosecFAQ/sysadmin/cfengine.htm, SANS Institute, 2001.

"Day Tutorial on Cfengine", by Mark Burgess, http://www.iu.hio.no/~mark/CfengineTutorial/, 2001.

"Using Rsync With Cfengine," J. Davis, http://www.cs.arizona.edu/people/jdavis/cfengine.html, 2000.

"rsync manual page," http://rsync.samba.org/ftp/rsync/rsync.html, 1999.

Various RCS Documents from the RCS Home Page: http://www.cs.purdue.edu/homes/trinkle/RCS/

Various CVS Documents from the CVS Home: http://www.cvshome.org/

Various BitKeeper Documents from the BitKeeper Home: http://www.bitmover.com/bitkeeper/

The /BriefCase 3 Toolkit Home Page: http://www.applied-cs-inc.com/bcintro.html

PRCS Home Page: http://prcs.sourceforge.net/

Aegis Home Page and FAQ: http://www.canb.auug.org.au/~millerp/aegis/aegis.html

Subversion Home Page: http://subversion.tigris.org/

"Copy from one, it's plagiarism; copy from two, it's research."
Wilson Mizner (1876-1933)

In Conclusion

The average human eats 8 mosquitoes in their lifetime at night.