Aegis
A Project Change Supervisor

CVS Transition Guide

Peter Miller
pmiller@opensource.org.au

.

This document describes Aegis version 4.25
and was prepared 25 July 2016.

This document describing the Aegis program, and the Aegis program itself, are
Copyright © 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012 Peter Miller

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

1. CVS Comparison

This chapter presents a brief comparison of Aegis and CVS.

1.1. Never Trust A Skinny Chef

This is a comparison of Aegis and CVS, but it's written by an Aegis expert who isn't a CVS expert. This means that, from time to time, information may be biased in Aegis' favour even though I'm trying to be even handed. Worse, there are probably inaccuracies stemming from my lack of familiarity with CVS: if you find any, I'd certainly like to know, so I can correct this document.

1.2. Contributors

In putting this chapter together, I solicited assistance from users on the `aegis-users` mailing list. Where you are reading their comments and not mine, you'll see it as a name at the end of the paragraph.

1.3. The Aegis Process

"*Writing software is hard, improving the way that software is written is harder.*" Bill Pugh

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 impies 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.

1.4. CVS has no Built-in Process

"By not having a process built into CVS, the team must work out a process separately. Since you're not likely to do better than the Aegis process, the likely result is to do worse. Also you will have to expend effort to implement and enforce the process." *Trenton G. Twining <tgt@cycletime.com>*

"Every project I've seen *use* either Aegis or an Aegis approach, so drastically outshines other projects in the same organization, it's clear to me that the Aegis approach is the difference. It works better with some

analysis. It works better with some planning and forethought. It works." *Trenton G. Twining <tgt@cycle-time.com>*

"For one thing, [Aegis] was actually designed, rather than just having accreted like CVS. And if you are looking for code reviews and all that stuff, it's probably easier to do them with Aegis (with CVS it is a roll-your-own type affair)." *Jim Kingdon <kingdon@panix7.panix.com>* (for many years, the chief CVS maintainer)

1.5. The Good, the Bad and the Ugly

This section contains the answers to some frequently asked questions.

1.5.1. Why should I change from CVS to Aegis?

- enforced review − damn important in a company environment *Gus <gus@getsystems.com>*
- mandatory testing (this may be disabled, per project)
- More space efficient for large code trees, and only one copy of the baseline (also makes backup easier) *Gus <gus@getsystems.com>*
- To maintain control over your code repository. The baseline can't even be written to by developers, so the audit trail is more secure. *Gus <gus@getsystems.com>*
- Support for change sets. My main complaint with CVS is that you are unable to associate modified files into a change so once the files are committed to the CVS repository, there is no easy way to back it out or work out which other files were changed as part of a logical set. *Tim Potter <Tim.Potter@anu.edu.au>*
- Separation of the roles of developer, reviewer and integrator. At the moment, typical distributed CVS development happens with people checking in stuff as they develop it with very little integration testing as they go along. It's pretty much up to people "in the know" to manually go through changed files and check to see if something has been broken by a developer. It gets even tricker when there are particular assumptions made that aren't written down. *Tim Potter <Tim.Potter@anu.edu.au>*
- Automated testing support.

1.5.2. What features does CVS have that Aegis hasn't?

- Low change overhead. Developers can omit as much of the process as they feel like.
- Partial checkins. You can commit just a few files from your work area.
- Works with both Windows and Unix. Aegis still struggles with this, and only has weak support for Windows.

1.5.3. How do I do CVS-style remote development?

A comprehensive answer will require quite a bit of understanding how Aegis' development process works. See the "Teamwork" portion of Aegis Howto and the "Geographically Distributed Development" chapter of the Aegis User Guide for more information. Here, however, is a brief summary.

"CVS-style remote development" is a development model whereby there is one master "server" containing an authoritative copy of the repository, and developers run CVS "clients" which contain shadow copies of the repository which are locally updated by developers to add new code. Periodically each local site must "pull" down the latest changes which others have committed to the server (via "cvs update"), and "push" its own updates back to the server to make them visible to others (via "cvs commit"). Merging and conflict resolution occurs at commit time, and no one is responsible for (or able to) ensure the integrity of the master repository, because anyone can commit garbage at any time, thus corrupting the baseline. The corrupted baseline then propagates to all clients.

Aegis's remote development can emulate this model, although Aegis supports a much broader range of distributed development topologies, as well as providing security against a corrupted baseline.

Here is one way of doing CVS-style remote development using Aegis. You need: one server machine running Aegis, many client machines running Aegis, and optionally but recommended a web server (this can be the same machine as the web server, or a separate machine). Briefly, you create an Aegis project on

a designated "server" site, create a change to populate the server repository with project files, then distribute this change with aedist (via email, WWW, or any other transport mechanism) to the client sites.

Each client site sets up a local Aegis project, receives the first aedist change set from the server, which populates the client's repository. Then each client does local development against the local repository via changes, and submits these changes as they are integrated back to the server via aedist change sets (email, WWW, or any other transport mechanism).

The server receives these aedist change sets from clients, integrates them into the server repository, and republishes them (again via aedist) for all clients to receive (the originating client will receive the same change set again, but aedist is smart enough to recognize if the change already exists, so it does not get applied a second time at the originating client site).

This differs from the CVS model because no one can directly update the server repository via "cvs commit" analogue; clients can only submit a change, but the server must integrate it before it is accepted by the server and published for consumption by other clients. Similarly, the server cannot directly update a client's repository via "cvs update"; the server can only inform the client of new changes, but the client must integrate the changes before they are accepted into the client's local repository. Notice that server and client operations are identical: both publish changes, and both accept changes from external sources.

Merging and conflict resolution occur at the time of receipt of the change (either by the server when it receives a local update from a client, or by the client when it receives a "master update" from the server). Aegis changes are automatically distributed as patch files and complete source file copies, and when Aegis receives the changes, it first attempts to use the patch file, thus doing merging automatically. Should the merge produce a conflict with the local (and hence updated) version of the file, the development directory for the received change will contain the copy of the file provided in the change, which naturally cannot reflect any changes to the file made locally. The developer is given ample opportunity to notice this merge conflict because Aegis requires that the received change must build, that it must pass its tests, that the developer generate difference files to see the exact effect of the change, that it be reviewed, and that it be properly integrated.

Note that if for any reason a client gets badly out of sync with the server (for instance if the client loses track of which aedist change sets it has already applied), the client can always download an aedist change set containing the server's entire project baseline, and apply this to the client's local repository (using the aedist −receive −nopatch option, which forces the received files to be applied as-is and not to try to patch the files). This will effectively synchronize the client's repository with the server's baseline, at which point the client can again download and apply incremental aedist change sets from the server. Also note that this situation can be avoided by keeping server and client well synchronized; one way of doing this is via e-mail automation, whereby the server automatically sends out aedist change sets via e-mail to all interested clients when the server integrates such a changeset. The clients could have a procmail filter to automatically pipe such messages through aedist −receive, at which point the change sets from the server are automatically unpacked and built, thus requiring only that they be reviewed and integrated.

Finally, notice that the server is not "authoritative" in any technical sense. The server publishes changes which clients are encouraged but not required to accept; the final authority for acceptance of a change into any repository is the integrity of the repository itself, enforced by the Aegis process.

1.5.3.1. Loose command equivalents

Here are some command which provide similar features to what CVS users may be familiar with.

A. To access a remote server:		
Copy of the remote project repository to a local project repository.	Fetch the latest complete version from a web server running the Aegis CGI interface. Process this downloaded ".ae" file with *aedist −receive*, building and integrating with the usual Aegis process.	Analogue of first-time remote *cvs co*.

Receive the latest change sets from the remote server	Fetch the latest complete version *or* the individual change sets of interest, from a web server runnjing the Aegis CGI interface. Process this downloaded ".ae" file with *aedist −receive*, building and integrating with the usual Aegis process.	Analogue of remote *cvs update*.
Send a change set to a remote server	Use the *aedist −send* command, and send the to the project maintainer. This is usually done by email.	Analogue of remote *cvs commit*.
B. To set up a server for others to access:	The Aegis CGI interface may be installed using the *aeget.instal* command. You need to be running Apache.	CVSup, *etc*
Receive and integrate changesets from contributors	The project maintainer uses the *aedist −receive* command on change sets received via email, or *aepatch −receive* for ordinary patches, or *aetar −receive* for tarballs. Automatic processing via procmail is also possible, but authentication (*e.g.* GnuPG) is essential.	Much like the usual *patch* command.

2. Aegis Commands for unreconstructed CVS users

This section compares Aegis commands and CVS commands. There can be no exact correspondence, because they are each actually attempting to achieve something different.

Create a new project. (In both cases, you then have to add content.)	`aenpr`	`cvs init`
Create a new work area.	`aenc;aedb` *or* `tkaenc`	`mkdir`
Copy files into a work area	`aecp`	`cvs checkout`
Add a new file	`aenf` *filename*	`cvs add` *filename*
Add a whole directory tree as new files.	`aenf .`	`cvs import`
Remove a file	`aerm` *filename*	`cvs remove` *filename*
Build in a work area.	`aeb`	`make`
Bring a work area up-to-date with the repository.	`aem` *(however, merging (the "m" in aem) is required much less often)*	`cvs update`
Finish development of a change.	`aede` *(and then it needs to be reviewed and integrated)*	`cvs commit`
Rename a source file	`aemv` *old-name new-name*	`mv; cvs remove; cvs add`
Show the differences between the work area and the repository.	`aediff`	`cvs diff`
Partial commits are forbidden in Aegis. However, it is possible to do something very similar.	`aeclone` *This clones the whole complete change. Now use* `aecpu` *etc, in the cloned change to get rid of the files you don't want to commit yet.*	`cvs commit` *filename*
How do I browse the sources? Please note that Aegis has a copy sitting there already. With CVS you have to create one first.	`aecd -bl` *now look around*	`mkdir` *somewhere* `cd` *somewhere* `cvs checkout` *now look around*
How do I find stuff in source files?	`aefind | grep`	`find | grep`
Clean out everything but primary source files from a work area.	`aeclean`	`make clean`
Make a separate copy of sources in the repository.	`aecp -independent`	`cvs export`

2.1. How do I use a Remote Server?

There is no direct equivalent, but see the section in this document titled "How do I do CVS-style remote development?" for an explanation of how to simulate this development model using Aegis distributed development tools. For more information on Aegis' distributed development model and tools, see *aedist*(1) or the "Teamwork" section of the HOWTO, or the "Geographically Distributed Development" chapter of

the User Guide.

2.2. How do I synchronize an Aegis project with a CVS project?

This situation might occur if you are using Aegis to track your source code changes to a project which is primarily managed in CVS. For this discussion we assume you have a CVS work directory in which you checked out the latest CVS sources. Further we assume that these source files have been imported and integrated into an Aegis project. Then, some time passes, and you have updated your Aegis sources via changes, and the CVS tree has also been updated via commits. The question is how can to synchronize the two source trees. The answer is to use patch files.

To import the latest CVS changes, create a copy of the CVS tree. Assume the old CVS tree is called cvs/ and the new copy is called new_cvs/. In the new_cvs/ directory, run "cvs update" to get the newest sources. Then run "diff −Nur cvs/ new_cvs/ −x Entries.Log > patchfile" to generate a patchfile showing the differences between the old and new CVS trees. Use *aepatch* to import this patchfile into an Aegis change, then follow the usual Aegis change procedure (build, test, diff, review, integrate). (The *aetar*(1) command is another possibility.)

To export your changes into CVS, again use *aepatch* to export your Aegis changes as patchfiles. Apply this patch to your local CVS tree (you might want to make a copy first), then execute "cvs commit" to submit your changes. Or, if you don’t have write access to the CVS repository, send the patchfile via other means (*e.g.* e-mail) to the project CVS maintainers.

3. Importing and Exporting

It is possible to import a CVS repository into Aegis, while preserving all of your history information.

Because Aegis uses change sets and CVS does not, it is necessary for Aegis to "synthesize" the change sets during the import process. See the *aeimport*(1) command page for a complete description of how this is done. Once the change sets have been discovered, the *aeimport*(1) command then synthesizes an Aegis change for each one, and enters it into the Aegis database as if it had taken place using Aegis.

Once this has occurred, you may then access all of the previous versions of the files as you would for any other Aegis project. See *aecp*(1) for more information, particularly the **−delta** option. All of the various lists and reports are also available (see *ael*(1) and *aer*(1) for more information). The usual web browsing is also available.

3.1. Importing from CVS

The command which does the importing is simple enough, but there is some setup required first.

- It is usual to have a separate account which "owns" an Aegis project. This means that accidental file writes while browsing sources can't happen, among other reasons. Whether you have a separate account per project, or a hold-all source account doesn't bother Aegis (the more security conscious among you, however, will want one account per project). If you have one account per project, name it after the project (there's less to remember that way). This example will call the both the account and the Aegis project "`example`".
- It is also possible to have a separate Unix group for each project. This means that you can have fine-grained control over who has read access to your repository. (Or, as above, you may want a simple source group.) Note that write access to the repository is controlled via a different mechanism in Aegis; the repository itself will be read-only to *all* staff, including authorized developers and integrators. (See the User Guide for how your changes actually reach the repository.) If you decide to have a group per project, use the same name as the project as, again, there is less to remember.
- Login as the `example` user.
- You need to decide where your Aegis project is going to live. It could live below the `example` user's home directory, but it does not have to. (It is impractical for it to actually *be* the `example` user's home directory.) Wherever you put the project directory, it must be accessible from all workstations and/or servers upon which development *and* code reviews are to be carried out. (See the "Teamwork" chapter of the HOWTO for more information.) In this example, we are going to put it in `/projects/example`; again, named for the project.
- If you have a single project under your $CVSROOT directory tree, you use the commands

  ```
  unset AEGIS_PATH
  aeimport $CVSROOT −project example −dir /projects/example −verbose
  ```

  Depending on how big your project is, this may take some time. You will be informed of progress, though it will not mean very much if you are new to Aegis.
- If you have several modules under your $CVSROOT directory tree, and the module you wish to import is contained under a single directory, use a command such as this:

  ```
  unset AEGIS_PATH
  aeimport $CVSROOT/example −project example −dir /projects/example −verbose
  ```

  Again this may take some time.
- If you have a more complex module structure below $CVSROOT, it may be necessary to duplicate and manually rearrange the directories until you have a single directory tree you can point the *aeimport*(1) command at. There is no support for reading the `$CVSROOT/CVSROOT/modules` file at this time.
- If something goes wrong part way through (*e.g.* running out of disk space) you will be left with a partial Aegis project that isn't very useful. Use the

  ```
  aermpr example
  ```

  command to get rid of it.

- Once the *aeimport*(1) command completes successfully, you need to use the command

```
ae_p example.1.0
ael project_files
```

  to examine the names of the files Aegis has imported. If they don't match your expectations (*e.g.* if the directory tree is not the shape you need) remove the project (*aermpr*) and re-run *aeimport*(1) with the appropriate arguments.
- The above commands create a project called "`example`", with branches "`1`" and "`1.0`" (see the Branching chapter of the User Guide for how branching works in Aegis). To access branch 1.0, you use the project name "`example.1.0`". If you wanted no branches at all (just a trunk) then add "`-version -`" to the above *import*(1) commands. If you are new to Aegis, and don't understand what this paragraph is talking about, *do not* change anything.
- The last thing you do as the `example` user is to add your normal account as a project administrator.

```
aena yourlogin
```

  Now logoff the `example` account, and login to your normal account.
- Aegis has populated the development roles with staff found during the import. Use the *project staff* report to examine and verify the resulting settings.

```
ae_p example.1.0
aer projstaff
```

  If there are staff in inappropriate roles, use the following commands to rearrange them.

	Add	Remove
Developer	*aend*(1)	*aerd*(1)
Reviewer	*aenrv*(1)	*aerrv*(1)
Integrator	*aeni*(1)	*aeri*(1)
Administrator	*aena*(1)	*aera*(1)

  See the User Guide for more information about the various roles within Aegis.
- You need to customize the `aegis.conf` file for your project. In order to do this, you will need to create a change set to modify the file. The worked example in the User guide will be useful in understanding how this is done. It is essential that you use a change set for this; editing the baseline directly is a *big* no-no. See also the configuration examples directory, `/usr/local/share/config.example`, for some pre-built configuration settings. (It's probably a Bad Idea to mess with the history command settings, but the rest are fair game.)

  In particular, your need to set the build command. The *aeimport*(1) command has set it to "`exit 0`". See the Dependency Maintenance Tool of the User Guide for more information.

You project should now be ready to use. Happy developing!

If you have questions, it is worth subscribing to the `aegis-users` mailing list. See `http://aegis.sourceforge.net/` for how to do this.

3.2. Exporting to CVS

If it becomes necessary to export your project from Aegis into CVS, this may be done relatively simply.

- This method only works if you have been using RCS as your history tool.
- Backup your CVS repository. The next step will change it.
- Copy the `,v` files from the Aegis history tree into the CVS root tree (the Aegis project in this example is called "`example`".

```
ae_p example.1.0
aecd -bl ../history
tar cf - . | ( cd $CVSROOT ; tar xf - )
```

  Note that this assumes that the two directory trees are the same shape. It is highly likely that you have a module called something like "`example`", which would change the last line above to look like

```
tar cf - . | ( cd $CVSROOT/example ; tar xf - )
```

More baroque directory structures will require more manual mangling during the copy.

Note that Aegis' expectations of locking, access lists, keyword expansion, *etc*, are rather different to CVS, so it may be necessary to use *find*(1) and *rcs*(1) to set things the way you want.

. .

Table of Contents