Aegis
A Project Change Supervisor

HOWTO

Peter Miller
pmiller@opensource.org.au

.

This document describes Aegis version 4.25
and was prepared 4 December 2012.

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/>.

Table of Contents

1. Introduction

This manual contains a series of brief lessons or “How To” guides for using Aegis. Each is arranged to cover two pages, so that when the manual lies open on the desk, the whole subject is easily visible in front of you.

When printing this manual, it is essential to print it double sided, or the “subject at once” effect will not occur.

The table of contents will be printed last. Insert it (there should be two pages, on one sheet of paper) before this page.

1.1. Assumed Knowledge

Many of these sections are written for use by beginners, so there is a fairly low level of assumed knowledge. However, you may want to have *The Aegis User Guide*, and *The Aegis Reference Manual* very close by, as all of the material conveyed here is available in a more expended or detailed form on those manuals.

1.2. Howto Install Aegis

The description of how to build, test and install Aegis may be found in the Reference Manual, under the heading *The BUILDING File*, which reproduces the `BUILDING` file included in the Aegis source distribution.

If you installed Aegis using a RedHat or Debian package, this will not be at all relevant to you, simply ignore it.

1.3. Howto Contribute

If you would like to see other “How To” subjects, please drop me an e-mail. Better yet, write one and e-mail it to me.

2. Cheat Sheet

This page is a quick reference into the common Aegis commands.

- Usually, "`man` *command_name*" can be used to get more details on a particular command.
- See also the official Aegis quick reference in the User Guide, page 88.
- The "`-p` *name*" option is used to specify the project name.
- The "`-c` *number*" option is used to specify the change number.
- The "`-l`" (or "`-List`") option can often be used to list subjects for the given command (eg. change numbers or projects) or simply to list rather than edit (*e.g.* a file or change attributes).

2.1. Common Commands

`ae_p` *project-name.branch-number*
: Set current project number for all following Aegis commands. The `ae_p` command with no arguments will 'unset' this forced default.

`ae_c` *number*
: Set current change number for all following Aegis commands. The `ae_c` command with no arguments will 'unset' this forced default.

`aecd [ -bl ]`
: Change directory [change to baseline].

`aeb` Aegis build − used by developers, reviewers and/ or integrators.

`aet` Run tests − used by developers.

`aed` Difference of change files with baseline.

`aedless`
: View difference files generated with `aed`.

`ael cd`
: List change details.

`aeca [ -l ]`
: Edit [list] change attributes.

`tkaenc`
: Create a new change (see *aenc*(1) for details), using a GUI interface. This makes it a damn sight easier to type in the description field.

`tkaeca`
: Edit change attributes (see *aeca*(1) for details), using a GUI interface. This makes it a damn sight easier to edit the description field.

`ael ll`
: List all of the lists (there are a lot).

`ael c` List all of the changes for a project (branch).

`ael cf`
: List all of the files in a change.

aeuconf(5)
: This is a man page documenting the `~/.aegisrc` file format.

2.2. Developer Commands

Procedure: `ael cd` → `aedb` → *do stuff* → `aeb` → `aet` → `aed` → `aedless` [→ `aeclean`] → `aede`

`aedb[u]`
: Develop begin [undo].

`aede[u]`
: Develop end [undo].

`aeclean`
: This will remove all files in the development directory which are not in the change as new files or copied files. This may delete files that you want, so be careful.

The *aeclean*(1) command uses Aegis' knowledge of what is *supposed* to be in the change. You are meant to tell Aegis about all source files you create in a development directory. If you have forgotten to do this, it is highly likely that the integration would fail in any case.

If you are importing files from elsewhere, use "*aenf .*" and all of the files in the directory tree below dot (the current directory) will be added to the change (make sure there are no object files when you do this).

`aecp`
: Prepares a file in the project for editing within the change; *i.e.* copy file into change from baseline. Remove symlink if necessary, etc.

`aecpu`
: Reverse the effects of the above.

`aecpu -unch`
: Will check all files in your change to see if any have not been modified, and perform an *aecpu* on them. This will stop an unnecessary version number increment for files that have not changed. (And also improves test correlations.)

`aem` Merge out-of-date files. See the *−Only-merge* option of the *aed*(1) command.

`aenf[u]`
Create/ add a new file [undo].

`aemv`
Rename (move) files.

`aerm[u]`
This tells Aegis the file is to be removed from the baseline when the change is integrated. Or *aermu* to undo this *before* the change is finished.

2.3. Reviewer Commands

Procedure: `ael cd` → `aecd` → `aedless` → *view output, review source files* → `aerpass`

Remember: the point of reviews is to find problems, not be a rubber stamp. You are expected to fail some reviews.

`aerpass`
Review pass.

`aerpu`
Review pass undo.

`aerfail`
Review fail.

2.4. Integrator Commands

Procedure: `aeib` → `aeb` → `aet` → `aed` → `aeipass`

There is an *aeintegratq*(1) script distributed with Aegis to do this procedure automatically.

`aeib[u]`
Integrate begin [undo].

`aeipass`
Integrate pass.

`aeifail`
Integrate fail.

2.5. Project Administrator Commands

This includes all of the commands that don't fit the categories above.

`aenc[u]`
Create a new change [undo]. See *aecattr*(5) for description of file format, or use *tkaenc*(1) instead.

`aend` and `aerd`
New developer; remove developer.

`aenrv` and `aerrv`
New reviewer; remover reviewer.

`aeni` and `aeri`
New integrator; remove integrator.

`aena` and `aera`
New (project) administrator; remove administrator.

`aepa [-l]`
Edit [list] project attributes (see *aepattr*(5) for file format).

`aeca [-l]`
Edit [list] change attributes (see *aecattr*(5) for file format).

`tkaeca`
Edit change attributes using a GUI. This makes it much easier to type in the description field.

3. How to Start Using Aegis

For the first-time user, Aegis can appear to be very daunting. It has a huge number of configuration alternatives for each project, and it is difficult to know where to begin.

It is assumed that you already have Aegis installed. If you do not, see the section of the *Reference Manual* called *The BUILDING File*. This reproduces the `BUILDING` file included in the Aegis source distribution.

3.1. First, Create The Project

You need to create a new project. Follow the instructions in the *How to Create a New Project* section, and then return here.

3.2. Second, Use a Template Project

The very first time you use Aegis, it will be easiest if you download one of the template projects. This gives you a project which (almost always) works correctly the first time "out of the box".

- The template projects can be found on the Aegis home page: `http://aegis.sourceforge.net/`
- If you are a long-time GNU Make user, you probably want the Make-RCS template, at least to start with.
- Follow the instructions found on the web page and you will have a working Aegis project, complete with self tests.
- From this starting point, create changes (the *tkaenc* command is good for this, as it gives you a simple GUI) and try modifying the calculator, or adding more programs to the project.
- The template projects is intended to be generally useful. Many users have simply retained this format and inserted their own projects into it. (Use a change to delete the calculator and its tests.)

3.3. Second, Copy a Template Project

If this isn't the very first time, you may wish to get more adventurous, and try copying the relevant bits out of a working project. Usually, when sites first try this, the working project will be one of the template projects from the previous section.

- Create a new project. For this exercise, you probably want a single user project.
- Create a new change
- Copy the project *config* file, and and files referenced by it, such as the new file templates and the build configuration file (*Makefile* or *Howto.cook*, depending).
- Copy the sources of the existing project into the development directory. If you have several levels of directories, reproduce this, too.
- Remove files which are not primary sources (*e.g.* the generated C sources of you have yacc input files).
- Using the "*aenf* ." command (yes, that's a dot, meaning "the current directory") you can tell Aegis to add all of the source files in the development directory to the change set.
- You will probably need to modify your build method to meet Aegis' expectations. Rather than do this immediately, change the *build_command* in the project *config* file to read "`build_command = "exit 0";`" and fix it in the next change set.
- Now, build, develop end, review and integrate, as found in the *User Guide* worked example. (Except, of course, there is only one member of staff.)
- Create a second change, and copy the project configuration file (called *aegis.conf* by default), and the build configuration file (probably *Makefile* or *Howto.cook*) into the change.
- This would be a good time to read the *Dependency Maintenance Tool* chapter of the Aegis User Guide, and also *Recursive Make Considered Harmful* (see the author's web site) if you haven't already.
- Edit the build configuration, try the *aeb* command; it will probably fail. Iterate until things build correctly.
- develop end, review and integrate as normal. Your project is now under Aegis.

4. How to Recreate Old Versions

It is possible to recreate old versions of your project. This is done using the delta number assigned to every completed change.

4.1. aecp

Recreating the sources is usually done to recreate a bug. To this end, it is also usually done from within an existing change. The *aecp*(1) command is used to copy histprical file versions into a change.

The *aecp*(1) command has some options which are used to perform the source recreation:

−DELta *number*
: This option tells *aecp*(1) to extract an historical version of the files, rather than the head revision (the one visable in the baseline). You need to know the *delta* number of the change, assigned at integration time, not the change number.

−BRanch *number*
: If the historical version is on a different branch than the one the current change is on, use this option. The branch number is to the left of the "D" in version strings.

−DELta-From-Change *number*
: This option tells *aecp*(1) to extract an historical version of the files, rather than the head revision (the one visable in the baseline). You only need the change number to use this option.

−DELta-Date "*string*"
: **This option tells *aecp*(1) to extract an historical version of the files, rather than the head revision (the one visable in the baseline). You only need the date the change was integrated to use this option. It understands many forms of written (English) dates, but try to avoid ambiguous month numbering (it can be confused by some European vs. American numeric formats, use month names instead).**

4.2. Finding Delta Numbers

You can find delta numbers in a number of ways:

• The "*ael change-details*" command will list change details. If changes are completed, their delta number will appear at the top of the listing.

• The "*ael project-history*" command lists all integration for a project, including their change numbers and delta numbers.

• The *aeannotate*(1) command lists the file source, annotationg each line with the developer, the date and the version. To the right of the "D" in the version is the delta number.

• The *#{version}* substitution (see *aesub*(5) for more information) is covered in the next section.

In addition, you may need to use the **−BRanch** option, if the historical version is on a different branch than the one the current change is on. The branch number is to the left of the "D" in version strings.

4.3. ${version}

The *build_command* field in the project *config* file may be given the *${version}* substitution, which you may use to embed the version string into your executables. You could, for example, have this string printed when yoiur program is given the **−version** command line option. For example:

```
% aegis −version
aegis version 4.15.D012
%
```

Armed with this version string, you can recreate the sources for the version being executed. The command

```
% aecp −change=4.15.D012 .
%
```

would be issued from inside a suitable change. This form of the *aecp −change* option combines the **−BRanch** and **−DELta** options into a single command line option.

4.4. Out Of Date

Once you have recreated your sources and rebuilt your project, and presumably fixed the bug you were hunting, there are a couple more steps.

• The first is to remove unchanged sources. Do this with the

```
% aecpu −unchanged
%
```

command. This removes from your change all files which were not changed by this change. This cuts down on the clutter and makes the next step much easier.

• The next step is to merge the files. Because you are working with historical versions of the files, Aegis will think they are out-of-date and want you to merge them. Do this is the usual way (using the *aem*(1) command). Remember that Aegis will stash a backup copy with a ",B" suffix before merging.

You may find the following command

```
% ael cf | grep '('
%
```

useful for finding out-of-date files.

• Once Aegis thinks all the files are up-to-date you then need to rebuild and retest before completing development.

5. How to Create a New Project

Before you can do anything else with Aegis, you need a project to do it to. The advantage of this is that each project is administered and configured independently.

If this is your first time using Aegis, you probably want a single-user project. You can change the number of users later, if you ever want to add more staff to the project.

You need to select the name with some care, as changing the project name later is tedious. Adding aliases, however, is simple.

5.1. Single User Project

A single suer project is one where all of the different staff roles are filled by the same person, and a number of interlocks are disabled, as you will see in a moment.

Unfortunately, there is no Tcl/Tk GUI for this, yet, which makes this documentation bigger then it needs to be.

Don't do anything yet! Read through all of the steps first.

- You may want to read the *aenpr*(1) man page for more information.
- The command "*aenpr* name *−version −*" will create the project with no branches. This will automatically make you (that is, the executing user) the project administrator and the project owner. The *umask* is remembered, too.
- The root of the project directory will be in your home directory, named after the project name. If you want something else, use the *aenpr −directory* option.
- The default group at the time of execution determines the file group of the project. Make sure the account created for the project has the correct group. (Even if you don't understand this, your system administrator should.)
- The *umask* at the time of execution determines the group access to the project. Even if you usually work with a restrictive *umask*, set it to the right one for the project before running this *aenpr* command.
- For additional security, it is often *very* useful to create a UNIX user for each project. You may need to consult your system administrator for assistance with this. It is usual to name the user and the project the same thing, to avoid confusion. Log in as this user to execute the initial project creation commands; once completed **no one** will ever login to this account again.
- Add the staff to the project: the "*aena* `your-normal-login`" command adds your normal account as a project administrator. You need this if you are using a special project account, so that your normal self can administer the project.
- At this point, log out of the special project account. Ask the system administrator to permanently disable it.
- Add the rest of the staff: the "*aend* `your-login`" command makes you a developer, the "*aenrv* `your-login`" command makes you a reviewer and the "*aeni* `your-login`" command makes you an integrator.
- You need to edit the project attributes next. The "*aepa −edit*" command does this. You will be placed into a text editor, and will see something similar to this:

```
description = "The \"example\" project";
developer_may_review = false;
developer_may_integrate = false;
reviewer_may_integrate = false;
developers_may_create_changes = false;
umask = 022;
```

  Ignore any extra stuff, you should not change it at the moment. To get a single user project, edit the field to read

```
developer_may_review = true;
developer_may_integrate = true;
reviewer_may_integrate = true;
developers_may_create_changes = true;
```

  Be extra careful to preserve the semicolons! You may also want to change the description at this time, too. Don't forget the closing double-quote *and* semicolon.
- Create the first branch now. They inherit all staff and attributes at creation time, which is why we worked on the trunk project first. The command "*aenbr* `name` *1*" followed by followed by "*aenbr* `name`*.1 0*" will give you a branch called *name*.1.0 for use wherever Aegis wants a project name. (See the branching chapter of the User Guide for more information.)

5.2. Two User Project

Everything is done as above, except you want to project attributes to look like this:

```
developer_may_review = false;
developer_may_integrate = true;
reviewer_may_integrate = true;
developers_may_create_changes = true;
```

This says that developers can't review their own work.

You will need to add the other person to the developer, reviewer and integrator roles, too.

Converting a single user project to a two person project is simply editing the project attributes to look like this later. *Remember:* each branch inherited its attributes when it was created − you need to edit the ancestor branches' project attributes too.

5.3. Multi User Project

Everything is done as above, except you want to project attributes to look like this:

```
developer_may_review = false;
developer_may_integrate = false;
reviewer_may_integrate = false;
developers_may_create_changes = true;
```

This says that developers can't review their own work, and reviewers can't integrate their own reviews. This ensures the maximum number of eyes validate each change.

You will need to add the other staff to the appropriate developer, reviewer and integrator roles. Staff need to always be permitted all roles: it is common for junior staff, for example, *not* to be authorized as reviewers.

Converting a single user project to a multi-person project is simply editing the project attributes to look like this later. *Remember:* each branch inherited its attributes when it was created − you need to edit the ancestor branches' project attributes too.

5.4. Warning

The */usr/local/com/aegis/state* file contains pointers to "system" projects. *Pointers*. Users may add their own project pointers (to their own projects) by putting a search path into the *AEGIS_PATH* environment variable. The system part is always automatically appended by *Aegis*. The default, already set by the */usr/local/lib/aegis/cshrc* file, is *$HOME/lib/aegis*. Do not create this directory, *Aegis* is finicky and wants to do this itself.

Where projects reside is completely flexible, be they system projects or user projects. They are not kept under the */usr/local/com/aegis* directory, this directory only contains pointers.

5.4.1. Creating Projects

When you create a new project, the *first* element of the AEGIS_PATH is used as the place to remember the project *pointer*. This means the project will not show up in the global project list if you have set AEGIS_PATH to include private projects.

There are two ways to make sure that you are creating a global project. Either "*unset AEGIS_PATH*" immediately before using the "*aenpr*" command, or use the "`-library` */usr/local/com/aegis*" option of the *aenpr* command.

5.4.2. Web Visibility

If you have a Web server, you may like to install the Aegis web interface. You do this by copying the *aeget* program from */usr/local/bin/aeget* into your web server's *cgi-bin* directory. There is a *aeget.instal* helper script, if you don't know where your web server's *cgi-bin* directory is.

You may prefer to use a symbolic link, as this will be more stable across Aegis upgrades. However, this requires a corresponding *follow-symlinks* setting in your web server's configuration file. (Use the *aeget.instal −s* option.)

If you have a Web server, and `aeget` was installed, you can use a wrapper script to set the *AEGIS_PATH* environment variable, if you want it to be able to see more projects than just the global projects.

5.5. Changing The Project Owner

Typically, when folks try Aegis for the first time, they don't worry about having a separate user for their projects. However, once things are ticking along, it is less and less attractive to toss it all and start again cleanly. So, now you need to change the project owner from the user who started the Aegis evaluation to the unique project user account.

1. You need to be *root* to perform this procedure.
2. Create the user account. It doesn't need to work to login, so the password can be disabled. You probably want to arrange to have this user's email forwarded somewhere sensible (maybe see the Distributed Development chapter of the User Guide).
3. The owner of the project is taken from the owner of the project directory tree, so this is

what needs to be changed. Go to the root of the project tree − the directory which appears in the "*ael projects*" listing. This isn't the trunk baseline, but the directory above it (you will see *info*, *history* and *baseline* sub-directories).

4. Use the command

```
chown -R username .
```

to change the ownership of this directory, and all files and sub-directories below it. Insert the username of the account you created in step 2. (You need the **dot** on the end of the command, its not mere punctuation.)

There is no need to change the owner of any active changes, or any other change attributes.

6. How to Move a Project

By "move a project", you may wish to change the project's name but leave the project files in the same location, or you may wish to change a projects directory location and leave it with the same name. This section covers both.

There are two ways to move a project. One is from within Aegis, and one is from outside Aegis. Each section below covers both methods.

6.1. Relocating a Project

This section deals with moving a project's files from one file system location to another.

6.1.1. From within Aegis

This works best when you are moving a project from one machine to another. It is a *very* good idea if there are no active changes on *any* branch.

Step 1: You need to know where in the file system the project currently resides. Take a look in the projects list (*ael p*) and see the directory reported for the trunk of the project. Ignore any active branches.

Step 2: Usually, when you remove a project, Aegis deleted all of the project files. However the aermpr −keep option tells Aegis to remove the project name, but keep all of the project files.

Step 3: Move the files to their new location, you need *all* of the files below the directory tree you found in step 1. This may be a simple file move, or may involve copying the files to tape, and then unpacking on a new machine. Remember to make sure the file ownerships are set the way you want (usually, this means "preserved exactly").

Step 4: Tell Aegis where the project is. To do this, use the −dir and −keep options of the *aenpr*(1) command.

6.1.2. From outside Aegis

This works best of the project is staying on the same machine, or the same NFS network.

Step 1: You need to know where in the file system the project currently resides. Take a look in the projects list (ael p) and see the directory reported for the trunk of the project. Ignore any active branches.

Step 2: Move the files to the new location.

Step 3: Edit the /usr/local/com/aegis/state file and edit the path appropriately to tell Aegis where you moved the files to. You will need to be root for this step.

6.2. Renaming a Project

This section deals with changing a project's name without moving is files.

6.2.1. From within Aegis

Step 1: You need to know where in the file system the project currently resides. Take a look in the projects list (*ael p*) and see the directory reported for the trunk of the project. Ignore any active branches.

Step 2: Usually, when you remove a project, Aegis deletes all of the project files. However the aermpr −keep option tells Aegis to remove the project name, but keep all of the project files. (The *aenbru −keep* command is the equivalent for branches.)

Step 3: Tell Aegis where the project is, using the new name. To do this, use the −dir and −keep options of the *aenpr*(1) command.

6.2.2. From outside Aegis

Step 1: Edit the /usr/local/com/aegis/state file and edit the name appropriately to tell Aegis the new name of the project. You will need to be root for this step.

6.2.3. Project Aliases

You may need some transition time for your developers. Either before or after you rename the project, you may want to consider adding a project alias (see *aenpa*(1) for more information) so that the project has "both" names for a while.

.

7. Working in Teams

Aegis supports teamwork in two basic ways: local development and distributed development. Local development breaks down into a single machine, and networked machines on a common LAN. By building the description a little at a time, this section will show how each of these modes of development are related in the model used by Aegis.

7.1. Local

7.1.1. Single User, Single Machine

The simplest case to understand is the single user. In such an environment, there is a project and the user makes changes to this project in the usual way described in the User Guide and earlier sections of this How-To.

Even in this environment, it is often the case that a single user will be working on more than one thing at once. You could have a large new feature being added, and a series of bug fixes happening in parallel during the course of this development. Or some-one may interrupt you with a more important feature they need to be added. Aegis allows you to simply and rapidly create as many or as few independent changes (and development directories) as required.

By using independent work areas, things which are not yet completed cannot be confused with immediate bug fixes. There is no risk of untested code "contaminating" a bug fix, as would be the case in one large work area.

7.1.2. Multi User, Single Machine

Having multiple developers working on the same project is very little different than having one developer. There are simple many changes all being worked on in parallel. Each has its own independent work area. Each is independently validated before it may be integrated.

One significant difference with multiple developers is that you now have enough people to do real code reviews. This can make a huge difference to code quality.

7.1.3. Multi User, Multi Machine

Aegis assumes that when working on a LAN, you will use a networked file system, of some sort. NFS is adequate for this task, and commonly available. By using NFS, there is very little difference between the single-machine case and the multi-machine case.

There are some system administration constraints imposed by this, however: it is assumed that each machine is apparently "the same", in terms of environment.

7.1.3.1. General Requirements

You need some sort of network file system (*e.g.* NFS, AFS, DFS), but it needs working locks (*i.e.* not CODA at present). I'll assume the ubiquitous NFS for now.

- You need exactly the same */etc/passwd* and */etc/group* file on every machine. This gives a uniform environment, with uniform security. (It also gets the UIDs right, which NFS needs.) Keeping */etc/passwd* and */etc/group* in sync across more than about 3 machines can be time consuming and error prone if done manually − so don't. Use NIS or similar − do sys admin once, automatically takes effect everywhere.
- All of the machines see the same file systems with the same path names as all the others. (Actually, you only need worry about the ones Aegis is interested in.) Again, you can try to keep all those */etc/fstab* files in sync manually, but you are better off using NIS (or NIS+) and the automounter or amd.
- All of the machines need their clocks synchronized. Otherwise tools which use time stamps (obviously *make*(1), but also a few others) will get confused. NTP or XNTP make this simple and automatic. In a pinch, you can use *rdate*(1) from cron every 15 minutes.
- Many sites are worried about the security of NFS. Usually, you need to take the root password away from workstation users; once the environment is uniform across all of them, the need for it usually disappears. It also means they can't abuse NFS, and they can't run packet sniffers, either. By using netgroups (I'm *not* talking about the */etc/groups* file) you can further restrict who NFS will talk to. By using NIS+ and NFSv3 you can quash the last couple of security issues, but unless you have military contracts, it's rarely worth it.

Fortunately, NFS and NIS readily available, both for proprietary systems and open source systems. Large sites use these techniques successfully and securely − and they **don't** have *O(nˆ2)* or even *O(n)* sys admin issues, they get most sys admin tasks down to *O(1)*.

But, *but*, **but!** Many sites are *very* concerned about being able to work when the server(s) are

down. I agree, however I suggest sweet talking your sys admin, not bashing NFS or NIS or Aegis. It is possible to get very high availability from modern systems (and even ancient PCs, using Linux or BSD).

The fact is, working in a team requires interaction. Lots of interaction. It is an illusion that you can work independently indefinitely. In the ultimate siege mentality, you need a full and complete private copy of everything in order to pull this off; but expect the other team member to carefully inspect everything you produce this way.

7.1.3.2. Aegis-specific Requirements

There are a couple of things required, once you have the above up and running.

- All of the Aegis distribution can be installed locally for performance, if that's what you need. (Except, see the next item.) Or, you can install it all on an NFS mounted disk, which guarantees everyone is always running exactly the same software revision which can sometimes be important (shortens upgrade times, too.)
- Except the *${prefix}/com/aegis* directory, which must be the one NFS disk mounted by every single machine identically, and must be read write. *I.e.* unique to the whole network (well, all machines using Aegis). This is where the pointer to the projects are kept, and this is where the database locks are kept. If this directory isn't common to every machine, the Aegis database will quickly become corrupted.
- The project directory tree must be on an NFS disk which all machines see, and must be the same absolute path on all machines. This is so that the absolute paths in *${prefix}/com/aegis/state* mean something.
- The development directories need to be on NFS disks every machine can see. Usually, this means a common home directory disk, or a common development directory disk. This can still be a disk local to the workstation, but they must all be exported, and all must appear in the automount maps. This is because Aegis assumes that every workstation has a uniform view of the entire system (so reviews can review your development directory, and integrators can pick up the new files from your development directory).

Large software shops have used these techniques without difficulty.

7.1.4. Known Problems

There is a known problem with the HP/UX NFS clients. If you see persistent "no locks available" error messages when */usr/local/lib/aegis* is NFS mounted, try making the */usr/local/lib/aegis/lockfile* file world writable.

```
chmod 666 /usr/local/lib/aegis/lockfile
```

There is the possibility of a denial of service attack in this mode (which is why the default is 0600) but since you are presently denied service anyway, it's academic.

7.2. Distributed

The distributed functionality of Aegis is designed to be able to operate through corporate firewalls. Corporate firewall administrators, however, take a very dim view of adding holes to the for proprietary protocols. Aegis, as a result, requires none. Instead it uses existing protocols such as e-mail, FTP and HTTP. It will even work with "sneaker net" (hand carried media).

The other aspect of Aegis, which you have probably noticed already, is that it is very keen on security. Security of the "availability, integrity and confidentiality" kind.

Incoming change sets are subject to the same scrutiny as a change set produced locally. It is dropped into a work area, built and tested, before being presented for review. Just like any local change set would be.

7.2.1. Multiple Single-User Sites

In the case of an Open Source project maintainer, this is essential, because incoming contributions are of varying quality, or may interact in unfortunate ways with other received change sets. This careful integration checking is essential. Imaging the chaos which could ensure if change sets were unconditionally dropped into the baseline. (Deliberate malice or sabotage, of course, also being a grim possibility.)

The careful reader will by now be squirming. "How", they wonder, "can the maintainer examine every change every developer makes. Surely it doesn't scale?"

Indeed, it would not. Aegis provides a mechanism for aggregating changes into "super changes". These larger changes can then be shipped around. (See the Branching chapter in the User Guide for more information.)

In the reverse direction, from the maintainer out to the developer, developers in an Open Source project probably aren't going to want to see each and every change set made to the project. Again, they can use an aggregation (*e.g.* grab the latest snapshot when each release is announced) to re-sync in larger chunks, less often. The chances of an intersection are fairly low (otherwise someone is duplicating effort) so the merge is usually quite simple.

7.2.2. Multiple Multi-User Sites

Most distributed large-scale corporate operations are actually similar to Open Source projects, though they usually have more staff. There is usually a "senior" site, and the other sites make their contributions, which are scrutinized carefully before being promoted to full acceptance.

Again, aggregations become essential to the system integration phase of a product. There may even be a hierarchy of concentrators along the way.

Junior corporate sites can sync periodically with the senior site, too, rather than double handle (or worse) every change set.

7.2.3. Telecommuting

One of the most desired cases is that of telecommuting. How do remote worker, who may never make it into the office, develop projects using Aegis?

There are many way to do this, but the simplest is to have a central cite ("the office") with satellite developers.

7.2.3.1. Office to Developer

The office makes available a web interface to Aegis. From this, it is possible to download individual changes, branch updates, or whole projects. All of this is already present in the Aegis distribution.

However, many corporate sites are not going to want to make all of their intimate development details to comprehensively available on the web. For such sites, I would suggest either a direct "behind the firewall" dial-in, or some virtual private networking software (which means users can use a local ISP, and still be treated "as if" they were behind the firewall).

If a VPN won't fly (due to company security policies), then selected encrypted updates could be posted "outside", or perhaps an procmail "change set service" could be arranged.

7.2.3.2. Developer to Office

It is unlikely (though possible) that you would have a web server on the developer's machine − usually you aren't connected, to the office pulling changes sets back is probably not viable.

The simplest mechanism is for the satellite developer to configure their Aegis project so that the trunk tracks the office version. Once a week (or more often if you get notified something significant has happened) pull down the latest version of "the office" as a change set and apply it. This way, the trunk tracks the official version.

The developer works in a sub-branch, with aeipass configured to e-mail branch integrations (but not individual change sets) back to the office. In this way, a work package can be encapsulated in a branch, and sent when finished. You also have the ability to manually send the branch at any earlier state, and it still encapsulates the set of changes you have made to date.

.

8. How to use Aegis with Python

This section describes how to use Aegis to supervise the development of Python programs. Some of the remarks in this section may also be helpful to people who use Aegis to supervise development in other non-compiled languages.

This section is contributed courtesy of Tangible Business Software, `www.tbs.co.za`. Python-specific questions relating to this section may be sent to Pieter Nagel at `<pnagel@tbs.co.za>`.

8.1. Handling Aegis search paths

8.1.1. The Aegis model vs. the Python model

Aegis' view of a project is that it consists of a hierarchy of project baselines. Each baseline consists of only those files that were modified as part of that (sub)project, plus all files that were built by the DMT (see the section of the *User Guide* called *The Dependency Maintenance Tool*). Aegis expects the DMT to be able to collect the entire project into one piece by searching up this baseline search path for all needed files.

This works fine when using statically compiled languages such as C. The build process "projects" source files from various Aegis baselines onto one or more executables. When these are run they do not need to search through the Aegis search path for parts of themselves; they are already complete.

Python programs, however, are never compiled and linked into a single executable. One could say that a Python program is re-linked each time it is run. This means that the Python program will need to be able to find its components at run-time. More importantly, it will need to avoid importing the old versions of files from the baseline when newer versions are present in the development or integration directories.

8.1.2. The solution

The simplest (and only recommended) way to marry Aegis and Python is to configure Aegis to keep all of the project's files visible in a the development and integration directories, at all times. That way Aegis' search path becomes irrelevant to Python.

Use Aegis version 3.23 or later, and set the following in the project *config* file:

```
create_symlinks_before_build
    = true;
remove_symlinks_after_integration_build
    = false;
```

The second directive is not available in earlier versions of Aegis.

If you keep your Python files in a subdirectory of your project, such as *src/python*, you will need that directory's relative in your *PYTHONPATH* whenever Aegis executes your code for testing, i.e. by setting

```
test_command="\
PYTHONPATH=$$PYTHONPATH:src/python \
python …";
```

in your project configuration file (example split across multiple lines for formatting only).

It may seem strange to you that we are not substituting the Aegis *Search_Path* variable into *PYTHONPATH* at all − it does at first seem to be the solution that is called for. The reason why we don't is very simple: *it does not work*. It is worth stressing the following rule:

Never inject any absolute path of any Aegis baseline into the Python search path.

8.1.3. Why setting PYTHONPATH to the Aegis search path will not work

The reason why *PYTHONPATH* does not work as Aegis expects is due to the way Python searches for packages. For a full explanation of what packages are, you can see *Section 6.4* of the *Python Tutorial*, but the crucial point is that a Python package consists of a directory with an *__init__.py* file in which the other files in that directory which should be treated as part of that package are listed.

When Python imports anything from a package, Python first searches for the *__init__.py* file and remembers the absolute path of the directory where it found it. It will thereafter search for all other parts of the package within the same directory. Without the *create_symlinks_before_build* and *remove_symlinks_after_integration_build* settings enabled, all the needed files are not guaranteed to *be* present in one directory at all times, however; they will most likely be spread out over the entire Aegis search path.

The result is that if you were to try and use the approach of setting the *PYTHONPATH* to the Aegis search path, package import will mysteriously fail under (at least) two conditions:

- Whenever you modify a file in a package without modifying the accompanying *__init__.py*, Python will find the *__init__.py* file in the baseline and import the *old* files from there.
- Whenever you modify the *__init__.py* and leave some other file in the package unmodified, Aegis will find the *__init__.py* in the development/integration directory but fail to find the unmodified files there.

8.2. The build step

Python programs do not need to be built, compiled, or linked before they can be run, but Aegis requires a build step as part of the development cycle.

One perfectly valid option is to explicitly declare the build step to be a no-op, by setting

```
build_command = "true";
```

in the project configuration file. *true*(1) is a Unix command which is guaranteed to always succeed.

In practice, however, there often are housekeeping chores that can be done as part of the build process, so you can just as well go ahead and create a Makefile, Cook recipe, or script that performs these tasks and make that your build step.

Here are some examples of tasks that can be performed during the build step:

- Setting the executable flag on your main scripts. Aegis does not store file modes, but it is often convenient to have one or more of the Python source files in your project be executable, so that one does not need to invoke Python explicitly to run them.
- Delete unwanted Python object files (*.pyc* and *.pyo* files). These could arise when you aerm and delete a Python script, but forget to delete the accompanying Python object file(s). Other files will then mysteriously succeed in importing the removed scripts, where you would expect them to fail. Your build process could use *aelcf*(1) and *aelpf*(1) to get a list of 'allowed' scripts, and remove all Python object files which do not correspond to any of these.
- Auto generate your packages *__init__.py* files. Python wants you to declare your intent to have a directory treated as a package by creating the *__init__.py* file (otherwise a stray directory with a common name like 'string', 'test', 'common' or 'foo' could shadow like-named packages later on in the search path). But since Aegis is, by definition, an authoritative source on what is and what is not part of your program it can just as well declare your intent for you.

8.3. Testing

Testing under Aegis using Python is no different from any other language, only much more fun. Python's run-time type checking makes it much easier to develop software from loosely-coupled components. Such components are much more suited to unit testing than strongly-coupled components are.

If the testing script which Aegis invokes is part of your project, there is one important *PYTHONPATH*-related caveat: when Aegis runs the tests, it specifies them with an absolute path. When Python runs any scripts with an absolute path, it prepends that path to its search path, and the danger is that the baseline directory (with the old, unchanged versions of files) is prepended to the search path when doing regression testing.

The solution is to use code like this to remove the test's absolute path from the Python path:

```
selfdir = os.path.abspath(sys.argv[0])
if selfdir in sys.path:
    sys.path.remove(selfdir)
```

Instead of copying these lines into each new test file, you may want to centralize that code in a test harness which imports and runs the tests on Aegis' behalf. This harness can also serve as a central place where you can translate test success or failure into the exit statuses Aegis expects.

The test harness must take care to import the file to be tested without needing to add the absolute path of the file to *sys.path*. Use *imp.find_module* and *imp.find_module*.

I can strongly recommend *PyUnit*, the *Python Unit Testing Framework* by Steve Purcell, available from `http://pyunit.sourceforge.net`. It is based on Kent Beck and Erich Gamma's *JUnit* framework for Java, and is becoming the *de-facto* standard testing framework for Python.

One bit of advice when using *PyUnit*: like Aegis, *PyUnit* also distinguishes between test failures as opposed to test errors, but I find it best to report *PyUnit* test errors as Aegis test failures. This is to ensure that baseline tests fail as Aegis expects them to. *PyUnit* will consider a test which raises anything other than a *AssertionError* to be an 'error', but in practice baseline test failures are often *AttributeError* exceptions which arise when the test invokes methods not present in the old

code. This is a legitimate way to verify, as Aegis wants us to, that the test does indeed invoke and test functionality which is not present the old code.

8.4. Running your programs

Of course you will at some stage want to run the program(s) you are developing.

The simplest approach is to have your program's main script be located at the top of your Python source tree (*src/python*) in our example. Whenever you run that script, Python will automatically add the directory it was found in to the Python path, and will find all your other files from there.

You can safely let your shell's *PATH* environment variable point to that script's location, since the shell *PATH* and the *PYTHONPATH* do not mutually interfere.

Just avoid the temptation to set the absolute path of that script into your *PYTHONPATH*, or otherwise your development code and baseline code will interfere with each other. This is not an Aegis-specific problem, though, since there would be potential confusion on any system, in any language, where two versions of one program are simultaneously visible from the same search path.

.

9. Howto End A Branch

“OK, I give up. I do not understand the ending of branches.”

Usually, you end development of a branch the same way you end development of a simple change. In this example, branch *example.1.42* will be ended:

```
% aede -p example 1 -c 42
aegis: project "example.1": change
42: file "fubar" in the baseline
has changed since the last 'aegis
-DIFFerence' command, you need to
do a merge
%
```

Oops. Something went wrong. Don’t panic!

I’m going to assume, for the purposes of explanation, that there have been changes in one of the ancestor branches, and thus require a merge, just like file *fubar*, above.

You need to bring file *fubar* up-to-date. How? You do it in a change set, like everything else.

At his point you need to do 5 things: (1) create a new change on example.1.42, (2) copy *fubar* into it, (3) merge *fubar* with branch "example.1" (4) end development of the change and integrate it, and (5) now you can end example.1.42

The −GrandParent option is a special case of the −BRanch option. You are actually doing a cross-branch merge, just up-and-down rather than sideways.

```
% aem -gp fubar
%
```

And manually fix any conflicts... naturally.

At this point, have a look at the file listing, it will show you something you have never seen before − it will show you what it is *going to* set the branch’s edit_number_origin to for each file, at *aeipass*.

```
% ael cf
Type   Action Edit        File Name
------ ------ -------     -----------
source modify 1.3         aerect/rect.c
              {cross 1.2}
```

Now finish the change as usual... *aeb*, *aed*, *aede*, *aerpass*, *aeib*, ..., *aeipass* nothing special here.

One small tip: merge file files one at a time. It makes keeping track of where you are up to much easier.

Now you can end development of the branch, because all of the files are up-to-date.

In some cases, Aegis has detected a logical conflict where you, the human, know there is none. Remember that the *aem* command saves the old version of the file with a `,B` suffix (‘B’ for backup). If you have a file like this, just use

```
% mv fubar,B fubar
%
```

to discard everything from the ancestor branch, and use the current branch.

.

10. How to Become an Aegis Developer

This section describes how to become an Aegis developer, and gives some procedures, some ideas of areas which need work, and some general guidelines.

Please note: if these instructions have a problem, let someone know! If you are having a problem, so is the next guy. *Please* send all problem reports to Peter Miller <pmiller@opensource.org.au>

10.1. Required Software

There are a number of pieces of software you will need to work on Aegis.

- It will probably come as no surprise that Aegis is developed using Aegis (never trust a skinny chef) so the first thing you need is to install Aegis and become familiar with using it. You will need Aegis 4.25 or later.
- You will need a C++ compiler. If your compiler is installed in an uncommon directory or has an uncommon name, you can set the appropriate attribute by editing the *aegis.conf.d/site.conf* file.
- You will need to install Cook, in order to build things. Version 2.8 or later, preferably you should track the latest release.
  http://miller.emu.id.au/pmiller/cook/
- GNU Autoconf 2.53 or later.
  http://ftp.gnu.org/pub/gnu/autoconf/
  If your tools are installed in an uncommon directory or have an uncommon name, you can set the appropriate attribute by editing the *aegis.conf.d/site.conf* file.
- GNU Automake.
  http://ftp.gnu.org/pub/gnu/automake/
- You will need to install FHist, for the history tool.
  http://fhist.sourceforge.net/
- You will need to install *tardy*, for manipulating tarballs.
  http://miller.emu.id.au/pmiller/software/tardy/
- You will need to install *ptx*(1), for the permuted indexes in the documentation. This is now part of GNU coreutils.
  http://ftp.gnu.org/pub/gnu/coreutils/
- You need psutils for the *psselect* utility, to manipulate the documentation files, mostly to put the tables of contents at the start, rather than at the end as GNU Groff generates them.
  http://www.dcs.ed.ac.uk/home/ajcd/psutils/
- You will need the developer libraries for the *rx* library (if you installed from the tarball, you have these, but if you installed from RPM, you need the −devel package as well).
  http://ftp.gnu.org/pub/gnu/rx/
- You will need the developer libraries for the *zlib* library (if you installed from the tarball, you have these, but if you installed from RPM, you need the −devel package as well).
  http://www.gzip.org/zlib/
- You will need the developer libraries for the *libcurl* library (if you installed from the tarball, you have these, but if you installed from RPM, you need the −devel package as well).
  http://curl.haxx.se/
- You need UUID generation capability. This requirement may be satisfied in several different ways depending of your development platform.
  First, on GNU/Linux you could skip this requirement provided that your kernel has support for */proc* filesystem. Please note: in order to work */proc* must be mounted and */proc/sys/kernel/random/uuid* must be present.
  Second, you could install the developer libraries for the *e2fsprogs* package (if you installed from the tarball, you have these, but if you installed from RPM you need the −devel package as well).
  http://e2fsprogs.sourceforge.net/
  Third, you could install the UUID library from OSSP:
  http://www.ossp.org/pkg/lib/uuid/
  Fourth, if your platform has support for an API compliant with DCE 1.1, Aegis also supports the DCE API.
- The GNOME libxml2 library (`http://xmlsoft.org/`) is used to parse XML, you will need version 1.8.17 or later. You do not have to install the rest of GNOME as this library is able to be used by itself. This package is **not** optional, you need it to successfully build Aegis.
- You need to install Bison, the GNU replacement for Yacc.
  http://ftp.gnu.org/pub/gnu/bison/
- You will need to install Flex, the GNU replacement for Lex.
  http://ftp.gnu.org/pub/gnu/non-gnu/flex/
- You need to GNU Gettext (0.16.1 or later) tools installed. Even though Glibc 2.0 and later include Gettext, you need the developer tools

as well. (You need GNU Gettext even on Solaris, because the Solaris Gettext implementation is less than adequate.)
http://ftp.gnu.org/pub/gnu/gettext/

- You need GNU Ghostscript, for the ps2pdf utility, so that you can create PDF documents.
http://ftp.gnu.org/pub/gnu/ghostscript/
- You need a *uudecode* with a `-o` option (to redirect the output). It is part of GNU Sharutils.
http://ftp.gnu.org/pub/gnu/sharutils/
- You need to install GNU awk.
http://ftp.gnu.org/pub/gnu/gawk/
- You need a *ctags*(1) command with a `-L` option (to read file names from standard input).
http://ctags.sourceforge.net/
- You need RCS installed for the automated tests.
http://ftp.gnu.org/pub/gnu/rcs/
- You need to install *sudo*(8). See `etc/set-uid-root` in the distribution for how to configure the */etc/sudoers* file.
ftp://ftp.sudo.ws/pub/sudo/
- The location box icon is generated using *convert* from ImageMagick, but the build can cope if you don't have it.
http://www.imagemagick.org/
- The PNG images are optimized by the *pngcrush* command.
http://pmt.sourceforge.net/pngcrush/
- It is possible to use the dmalloc library for debugging memory abuses. Be warned: the dmalloc library can be instructed to log to a file, circumventing the Aegis I/O layer, thus it's possible to create file owned by root. The dmalloc library should only ever be used as a debugging tool, and *never* be used in a production build of Aegis.
http://dmalloc.com/
On a Debian system, use the `apt-get install libdmalloc-dev` command. You will need to aecp the `etc/Howto.cook` file to alter the build to use the dmalloc library.
- *Probably more things I've forgotten.*
- Some parts of the build need Perl

10.2. Create The Aegis Project

The next thing to do is create an Aegis project to hold the Aegis source. This is done in the usual way. I suggest you create branches which match the current public release, *e.g.* it is 4.25 at present.

The best-practice technique of having a separate user account to mind the sources is recommended. The following commands should be run as that user, not your usual login. This prevents a variety of accidents which can happen when you are browsing the baseline (master source).

You could use the following command:

```
% aenpr aegis.4.25
aegis: project "aegis": created
aegis: project "aegis.4.25": created
%
```

but experienced Aegis users will know that this means a laborious setting of project attributes. It is easier to create the top-level project, set the attributes, and the create the branches, so that they inherit the attributes on creation.

```
% aenpr aegis -version -
aegis: project "aegis": created
% aepa -e -p aegis
edits the project attributes for single user,
or you may find tkaepa easier
% aena -p aegis you
aegis: user "you" is now a administrator
% aend -p aegis you
aegis: user "you" is now a developer
% aenrv -p aegis you
aegis: user "you" is now a reviewer
% aeni -p aegis you
aegis: user "you" is now an integrator
% aenbr -p aegis 4
aegis: project "aegis.4": created
% aenbr -p aegis.4 25
aegis: project "aegis.4.25": created
%
```

At this point, the rest of the commands in this chapter may (should!) be executed as "*you*," your usual login account. When you added your normal account as an administrator just now, you authorized yourself to perform the necessary actions.

You will need about 120MB of free space to build and integrate Aegis changes, or more, depending on the changes you make and the size of your development directories.

The *.forward* file of the "aegis" user needs to be set to someone appropriate to read mail directed at the project.

You can now set the "aegis" user's password field to "*". This effectively prevents the "aegis" user from logging in. Aegis is designed to make this unnecessary from now on.

10.3. The Download

The Aegis project is distributed in the form of an *aedist*(1) change set. The file to download is

called `http://aegis.sourceforge.net-/aegis-4.25.ae` and can be grabbed using your favorite web browser, or *wget*(1).

The downloaded change set is applied using the following command

```
% aedist -receive \
    -f aegis-4.25.ae \
    -p aegis.4.25
...lots of output...
%
```

It is a good idea to give the project name on the command line, or *aedist* will try to use the project name it finds in the change set, and it probably wont find it if you are using different numbering to the chief maintainer's copy.

The *aedist* command will, in turn, issue a number of other commands. These are all normal Aegis commands you could issue yourself, if you were familiar with Aegis. It will, however, stop with a moderately alarming message:

> Warning: This change contains files which could host a Trojan horse attack. You should review it before building it or testing it or completing development. This change remains in the *being developed* state.

This message comes because in order to build the project, you are going to have to execute a number of commands contained in the project *aegis.conf* file, and in the *etc/Howto.cook* file. For your own protection, *aedist* stops at this point. You may want to inspect these two files before continuing.

I really must get around to signing it with PGP. This would make the −notrojan option safe, because you could tell the file is direct from the chief maintainer, and thus as trustable as you think the chief maintainer happens to be.

In order to complete development of the change set, you must first build it...

```
% ae_p aegis.4.25
% aecd
% aeb
...you will see commands which build the project...
%
```

Things that can go wrong...

- There are checks to make sure that there is no white space on the ends of lines. If you use Emacs, you can add

  ```
  (add-hook 'write-file-hooks
      'delete-trailing-whitespace)
  ```

  to have this done automagically. The same checks also verify that the text files are all printable, and that line lengths are 80 characters or less. Please don't disable the checks, it makes accepting your patches more time consuming.
- Each change set has an architecture list associated with it. Initially you won't care, but Aegis does if you see the following error message:

  > found 1 unlisted architecture, edit the change attributes to remove it or edit the project configuration file to add it

  You need to use the *aeca −e* command (*not* the tkaeca command). You will be placed into an editor (usually *vi* unless you have used Aegis before, and know how to configure it differently). You need to leave just about everything alone, except for the architecture specification. Change it from

  ```
  architecture =
  [
      "unspecified",
  ];
  ```

  to something more meaningful on your machine. For PC users, this is almost always

  ```
  architecture =
  [
      "linux-i386",
  ];
  ```

  The alternatives may be found in the *config* in the current directory (search for `architecture =`). If you can't see a suitable choice, you may need to add one; the *aepconf*(5) man page has more information. Edit the *config* file to contain a suitable entry, and then use the *aeca −e* command to have the change agree with it.
- If you don't have Cook installed, the build command (aeb) will fail.
- If you don't have GNU Bison installed, the build will fail.
- If you don't have GNU Gettext installed, the error message run-time binaries will not be built. This isn't an error, so you can keep going, but you'll get the shorter, cryptic form of the error messages.
- Please note: if these instructions have a problem, let someone know! If you are having a problem, so is the next guy. *Please* send all problem reports to Peter Miller <pmiller@opensource.org.au>

Once the change builds, you need to difference it (this is a little redundant for this first command,

but you'll see how useful it is later).

```
% aed
...you will see commands which "diff" the project...
%
```

Things that can go wrong...

- If you don't have the FHist package installed, the difference (aed) will fail. The *fcomp* command it is looking for is a whole-file context difference command (the GNU **diff -c** is a bit too terse for humans).

Now you will need to test the change. This is the basic test suite for Aegis, it ensures nothing is broken. It takes a while, go grab a cup of coffee.

```
% aet
...lots of output...
%
```

The change is now ready to end development.

```
% aede
aegis: project "aegis.4.25": change 10:
    development complete
%
```

The change set is now ready to be reviewed. In a single-person project like this one, you can review your own work. Obviously this is a conflict of interest, and larger projects are usually configured to have Aegis prevent this.

```
% aerpass -p aegis.4.25 -c 10
aegis: project "aegis.4.25": change 10:
    review pass
%
```

The change is now ready to be integrated. Only when integration is complete are the files actually committed to the repository.

```
% aeib -p aegis.4.25 -c 10
% aeb
...you will see commands which build the project...
-rwsr-xr-x 1 root ... arch/bin/aegis
-rwsr-xr-x 1 root ... arch/bin/aeimport
-rwsr-xr-x 1 root ... arch/bin/aelock
Integrator: please do the following:
  sudo .../baseline/etc/set-uid-root arch aegis aeimport aelock
  if they aren't root already.  See etc/set-uid-root for
  instructions for how to set-up your /etc/sudoers file.
%
```

This message at the end of the build is where Aegis is made set-uid-root in the repository. You want to do this, because you are going to symlink out of */usr/local/bin* (or wherever you installed Aegis) right into the baseline. In this way, you will be executing your bleeding-edge version of Aegis, exercising it before you send it to anyone else. Hang on a bit, the sending part comes later.

Don't know how to set these files set-uid-root? The above message includes the command to do it:

```
$ cd blahblah/delta*
$ sudo etc/set-uid-root arch aegis aeimport aelock
$
```

You will need to substitute the appropriate architecture name, although it is likely to be "unspicified on a fresh project.

Things that can go wrong...

- If you don't have *ps2pdf* or *psselect* or *ptx* installed, it won't build the documentation (this isn't an error, just keep going).
- If you don't have *tardy*(1) install, it won't build the tarball (this isn't an error, just keep going).
- Please note: if these instructions have a problem, let someone know! If you are having a problem, so is the next guy. *Please* send all problem reports to Peter Miller <pmiller@opensource.org.au>

If all is OK, continue with the integration...

```
% aed
...you will see commands which "diff" the project...
% aet && aet -bl
...lots of output...
% cd
% aeipass
...you will see commands committing the files to fhist...
aegis: project "aegis.1.0": change 10:
    integrate pass
%
```

The "*cd*" command you see is actually important: you need to be out of the development directory and integration directory so that they can be cleaned up (deleted) when the change completes.

10.4. Supporting Several Architectures

Aegis is able to track architectures to make sure that change sets have been built and tested on all necessary architectures. You may hav notices that Aegis is calling your architecture "unspecified", you can give it a more descriptive name, too.

The architecture configuration data is described in the *architecture* field of the *aepconf*(1) man page. It is based on the *uname*(2) data, see the man page for how. You will, of course, need a change set to change it.

Once you have a change set, you need to create the *aegis.conf.d/architecture* file.

```
% aenf aegis.conf.d/architecture
% aefa aegis.conf.d/architecture entire-source-hide
% aefa aegis.conf.d/architecture local-source-hide
%
```

Here are some suggestions for what you may like to set for your architecture or architectures.

```
architecture =
[
  {
    name = "linux-i386";
    pattern = "Linux-*-*-i?86";
  },
  {
    name = "linux-x86_64";
    pattern = "Linux-*-*-x86_64";
  },
  {
    name = "freebsd-i386";
    pattern = "FreeBSD-*-*-i?86";
  },
  {
    name = "sunos-4.1-sparc";
    pattern = "SunOS-4.1*-*-sun4*";
  },
  {
    name = "solaris-2.6-sparc";
    pattern = "SunOS-5.6*-*-sun4*";
  },
  {
    name = "solaris-2.6-i386";
    pattern = "SunOS-5.6*-*-i86pc";
  },
  {
    name = "solaris-7-sparc";
    pattern = "SunOS-5.7*-*-sun4*";
  },
  {
    name = "solaris-7-i386";
    pattern = "SunOS-5.7*-*-i86pc";
  },
  {
    name = "ppc-Darwin-7.x";
    pattern = "Darwin-7.*-Darwin*";
  },
];
```

Remember to only include the architectures from the above list that you actually have. Having architectures in this list that you don't routinely have access to means that you will not be able to *aede*(1) any change sets.

Occasional architectures can be handled, too:

```
architecture =
[
  {
    name = "ppc-Darwin-7.x";
    pattern = "Darwin-7.*-Darwin*";
    mode = optional;
  },
];
```

Again, only do this with architectures you actually have access to.

If you need to have architecture specific options to some commands, you can also have *project_specific* attributes, too. **Note** that you should *first* look into having a `$prefix/share/config.site` or `$prefix/etc/config.site` file for *./configure* to read. This is particularly important if you want to include */usr/local/include*, */usr/local/lib*, *etc*, in the various compiler flags.

10.5. The Bleeding Edge

As I mentioned above, the next thing to do is create symbolic links out of */usr/local/bin* into your Aegis baseline. The reason for doing this is so that you exercise your Aegis changes by using Aegis before you send them to anyone else. (Never trust a skinny chef.)

There is a shell script called *ae-symlinks* in the baseline *$arch/bin* direcdtory. Use it like this:

```
$ aecd -bl
# su
Password:
# linux-i486/bin/ae-symlinks aegis.4.25
# exit
$
```

The *linux-i486* may need to be replaced with the output of the *aesub -bl '$arch'* command if you are using something more interesting than a PC.

10.6. Undiscovered Country

If you got this far, your local Aegis project is ready for use.

It is strongly suggested that you complete the first change "as is" and perform your own customizations in later changes, rather than trying to get the project started and customize it at the same time.

The rest of this file describes how to perform various common changes to the example project.

10.7. Sending Changes

First, read the Distributed Development chapter of the User Guide.

When you have a change set you wish to have the other Aegis deveopers try, use a simple command such as:

```
aedist -send -p aegis.4.25 -c number | \
gpg --clearsign | \
mail aegis-developers@lists.sourceforge.net
```

or similar. (Or maybe *aepatch*(1) instead.) A suitable subject line would be very helpful.

10.8. Guidelines

10.8.1. What You Can Do

Write more documentation. There is a crying need for documentation of all sorts: better manual pages, more and better information in the User Guide, more and better HOWTOs. If you work out how to do something, and it isn't in the documentation, write some documentation and put it in a change set because other folks have probably missed it too.

Add more ease-of-use functionality. Stuff which makes the development process more efficient, or makes the information in the repository more accessible.

Extend the GUI. All of the commands which manipulate the change while in the *being developed* state are candidates. Some kind of wrapper that ties it all together would be good, too. User preferences, project attributes and creating projects are candidates, too.

Most new project configuration things belong in the project *config* file. Only add new project attributes (aepa −e) for things which (a) are catch 22 to change in a change set, or (b) allow a security abuse if in a change set (e.g. the notify commands, particularly aede), or (c) allow the repository to be damaged. (My thanks to Ralf Fassel <ralf@akutech.de> 2 Feb 1999 for pointing this out.)

10.8.2. What You Can't Do

You can't change Aegis' semantics. Developers around the world, and their managers, rely on Aegis working just the way it does right now. You can't change things that will compromise their ability to get things done.

Particularly, Aegis has a strong security story. Availability, integrity and confidentiality, and all that. If you want it more flexible, that's good, but you can't change the defaults and you can't make it irretrievably weaker. (You can, as a *non*−default make it weaker, within limits.)

Aegis (the executable, not the whole package) is quite big enough. Don't add code to *arch*`/bin/aegis` than can be done with the report generator, or as a separate program like `aesub` or `aefind`. More GUI can be added using Tk/Tcl − unless you have grander plans and even then it *still* shouldn't be added to the set-uid-root executable.

10.9. Coding Style

Please try to emulate the existing coding style. (Indents recently changed from 8 to 4, not all of the code has caugh-up yet.) Lines are to be 80 charcters or less wide, limited to the 95 printable ASCII characters, with no trailing white space.

Probably need a GNU Indent profile for code formatting, too.

10.10. Writing Tests

If you have fixed a bug you should write a test to verify the correct behaviour of Aegis. Because test file names are generated automatically starting from your repository state, it's possible that *aet* will create a test with the same name as one in the P.Miller repository. Because Aegis is not yet able to detect such situation, if you plan to send back your work to P.Miller you may want to modify your `aegis.conf` adding the following lines:

```
new_test_filename =
  "test/${zpad $hundred 2}/"
  "t${zpad $number 4}${left $type 1}−${left ${user log
```

In this way the possibility of a name collision should be reduced. Invoke *aent*:

```
% aent
aegis: appending log to "aegis.log"
aegis: user "walter", group "projadm"
aegis: rm −f etc/cook/change_filesf etc/cook/project_f
aegis: project "aegis.4.16.2": change 11: file "test/0
%
```

Now you can start to implement the test. Remember to invoke the programs under test as `$bin/program`.

- In order to improve error messages you should organize your script as a sequence of activity and use the *activity* variable as sketched below:

  ```
  #
  # create a new change
  #
  activity="new change 163"
  cat > tmp << 'end'
  brief_description = "The first change";
  cause = internal_bug;
  end
  if test $? −ne 0 ; then no_result; fi
  $bin/aegis −nc 1 −f tmp −p foo > log 2>&1
  if test $? −ne 0 ; then cat log; no_result; fi
  ```

  If you are reading this document, you probably don't need help to understand this code fragment, the only thing to note is that the number in the string (163) refer to the current line number and is used when printing a failure message. You don't need to maintain it by hand as

explained in the following step.

- You can use `test/activity.sh` to automatically renumber the activity variables of your tests:

```
$ sh test/activity.sh
test/01/t0159a-walt.sh...
test/01/t0160a-walt.sh...
$
```

  If you have not modified `test/activity.sh` you should find it as `bl/test/activity.sh` or `blbl/test/activity.sh`.

10.11. Debugging

Aegis, as any other software, may contain undiscovered bugs. If you are interested in helping to fix these bugs, and as a developer you should be interested, the first thing to do is compiling Aegis in DEBUG mode. In order to do so you must modify `common/main.h` and uncomment the DEBUG define. (If you use the *aecp −read-only* option, Aegis will remind you to uncopy the file again before develop end, ensuring that you don't accidentally integrate this.)

In DEBUG mode the −Trace command line option is available for most Aegis commands. This option is followed by the names of the source files you want to trace, and may be used more than once.

If you need to add tracing capability to a file, you must first include `trace.h`, modify the code in order to use the trace facility (look at `common/trace.h`) then build the change with *aeb* and run the buggy command with the proper −Trace option.

On Linux >= 2.4 the *aegis* command, wich is setuid-root, is enabled to dump core when needed. If this does not happen, remember to verify the *ulimit*(1) settings; you may need to execute the *ulimit −c unlimited* command.

10.12. The To-Do List

- Add an additonal mode to *aedist* to query an *aeget* server for change set UUIDs and download and apply missing change sets. It needs to be able to be run by *cron*(8). Submitted: PMiller, 1-Jun-2004

10.12.1. aecvsserver

- The aecvsserver needs to be extensively tested by users. Submitted: PMiller, 1-Jun-2004
- Implement more of the CVS client commands which can be sent to the server, usually by saying "yes, bwana" and doing nothing. Submitted: PMiller, 1-Jun-2004
- Implement a cvs commit against a project (at the moment this is not allowed because you have to use a "module" named after a change set) which will create a change set apply the changes and do everything to get to aede. Submitted: PMiller, 1-Jun-2004
- Is it possible to use the same techinique to write an SVN server? Submitted: PMiller, 1-Jun-2004
- Is it possible to use the same techinique to write an arch server? Submitted: PMiller, 1-Jun-2004
- Arch has the concept (if not the implementation) of an SCM-neutral interchange format. Implement it. Submitted: PMiller, 23-Jan-2004

10.12.2. Geographically Distributed Development

- The *aedist −receive* command needs to be enhanced to understand file attributes. Submitted: PMiller, 2-Jun-2004
- The *aepatch −receive* command needs to be enhanced to understand file attributes. Submitted: PMiller, 2-Jun-2004
- Enhance the *aedist −receive* command to understand incoming files with UUIDs. Submitted: PMiller, 1-Jun-2004
- Enhance the *aepatch −receive* command to understand incoming files with UUIDs. Submitted: PMiller, 1-Jun-2004
- Add an additonal mode to *aedist* to query an *aeget* server for change set UUIDs and download and apply missing change sets. It needs to be able to be run by *cron*(8). Submitted: PMiller, 1-Jun-2004
- Enhance *aedist* to preserve change history (both send and receive will need work). Don't forget backwards compatibility. Submitted: Jerry Pendergraft, 2003
- Enhance *aedist −receive* to leave changes in *awaiting development* or *being developed* if that's the state they were in at the origin. Submitted: Jerry Pendergraft, May-2004
- Enhance *aepatch −receive* to run tests on changes which require it. Submitted: PMiller, 1-Jun-2004

- Enhance *aepatch* to preserve change history (both send and receive will need work). Incoming patches with no meta-data obviously can't do this. Don't forget meta-data backwards compatibility. Submitted: PMiller, 1-Jun-2004
- Enhance *aepatch −receive* to leave changes in *being developed* if that's the state they were in at the origin. Patches with no meta-data stay in *being developed*. Submitted: PMiller, 1-Jun-2004
- Enhance *aepatch* and *aedist* to automagically sign (send) and verify (receive) the contents, using the (revolting) library from the *gpgme* project. This stupid library spawns an *gpg*(1) instance and talks to it; unlike a sensible library *e.g.* the *zlib* project; why on earth couldn't they take the common code from gpg and make a library of *that?* Submitted: PMiller, 1-Jun-2004

10.12.3. Documentation

- Add a section to the branching chapter of the User Guide, describing how a developer may use a branch to temporarily waive the build command. After a series of changes on this branch, the build command is restored, and the branch development ended. This allows regular "non working" commits, without losing any of the strengths of the Aegis process. Submitted: 7-Mar-2000
- The manual pages need to have an example(s) section added to make them clearer. This isn't just for beginners, infrequently used commands need examples even for sophisticated Aegis users. Submitted: Geoff Soutter <geoff@whitewolf.com.au>, 3 Mar 2000
- Get tkdiff 3-way merge working with Aegis (see `http://www.ede.com/free/-tkdiff/` for code). Submitted: 24-jan-2000
- Add information to the History Tool chapter, describing how to use CVSup to access the RCS history tree. Submitted: 28-jan-2000
- the RCS history commands in the aegis user guide all use the '−u' option for 'ci' to check out a copy after registering/updating a file. However 'ci −u' always does keyword expansion. To avoid this, we have omitted the −u, so the working file is gone after the 'ci'. We check it out again using 'co', this time with the '−ko' option to avoid keyword expansion. Note that the −ko option is always given to the 'co' command, never to 'ci' or 'rcs'. Submitted: Ralf Fassel <ralf@akutech.de>, 18 Jan 2000
- * diff ; test $? −le 1 → diff ; test $? −ne 1 means that unchanged files prevent aede!! (Only fly in the ointment is moving files − need to cope with this.) Submitted: Gus <gus@get-systems.com> 28 Jul 1999
- mention in the diff tool part of the User Guide, that you can mess with diff_command to exclude with binary files, or file with CR in them, or lines too long, *etc*. Submitted: PMiller, 28-jun-99
- in the branching chapter, have a section about using sub-branches to turn build_command off (or to ignore exit status), and integrate lots of teensy tiny bug fixes, and then turn it on again. In the front, reference the branching chapter in "how to extend the Aegis process" Can mention extra review steps there, too. Submitted: choffman@dvcorp.com, 22 Jun 1999
- Document the build_time_adjust_notify_command in the DMT chapter of the User Guide. Update the example projects to use it. Update the config example to use it. Submitted: PMiller, 4-Apr-99
- Mention binary files in the diff and merge section (may provide aebinfil command to help choose which behavior?) Submitted: PMiller, 31-mar-99
- mention "rcs −ko" in the User Guide and put it into the examples AND also fhist keywords in the User Guide and put it into the examples. and make sure the examples all have hist_{put,create} the same. Subject: Ralf Fassel <ralf@akutech.de>, 9 Mar 1999
- worked example, "5.2.7 says that the cook file contains all of the above commands but my copy doesn't have them ..." [for config file and howto.cook file] BUT integration diffs not in the worked example. Submitted: Michael McCarty <mmccarty@xinetix.com>, 26-Feb-99
- need discussion (Howto, or maybe the User Guide) of how to use Aegis when you site has a mix of Unix and Wintel. Submitted: Paolo Supino <paolo@schema.co.il>, 4 Feb 1999
- add chapter to User Guide, saying how to config web interface and how to use it. Submitted: Graham Wheeler <gram@cdsec.com>, 27 Jan 1999

- User Guide: big changes bouncing: how to use a branch to get smaller reviews and smaller diffs. Submitted: Ralf Fassel <ralf@akutech.de>, 27 Jan 1999
- note for User Guide: metrics software form ftp://ftp.qucis.queensu.ca/pub/software-eng/-software/Cmetrics/
- correct documentation of file locking in UG: correct the example around the file locking − it gives the wrong text of the aede error − and probably other stuff. also, the wrong person comes back from aerobics

10.12.4. More Reports

- Add a −REVerse option, so that all of the listings (ael) come out in the reverse order to that used at present. Submitted: John Darrington <johnd@ot.com.au>, 20-Jul-2001
- Write an *aereport* file to produce MS-Project views of a project, making sure that the states of each change are linked, use averages to predict any incomplete states. And maybe another to produce HTML pages of the same thing. Submitted: 15-Jan-2000
- On the *aeget*(1) web pages, link the file edit numbers to pages which will retrieve the historical version. Submitted: Anoop Kulkarni <anoop@sasi.com>, 22 Dec 1999
- Add a user_change report (just like “ael user_changes”) which takes a user name, so you can get a list of changes by user. Make *aeget*(1) do this, too. Submitted: Ralf Fassel <ralf@akutech.de>, 9 Dec 1999
- Add a outstanding_changes report (just like “ael outstanding_changes”) which takes a user name, so you can get a list of outstanding changes by user. Make *aeget*(1) do this, too. Submitted: Ralf Fassel <ralf@akutech.de>, 9 Dec 1999
- Write a report which says when you have to do to get a change completed Jerry says he has written most of this. Submitted: jerry.pendergraft@endocardial.com 3-Nov-99
- ael change_history − write as a report and then include project history for sub branches. Don’t forget the web reports, too. Submitted:Jerry Pendergraft <jerry@endocardial.com>, 30 Aug 1999
- ael outstanding_changes − rewrite as a report and then include sub branches. Don’t forget the web reports, too. Submitted: Jerry Pendergraft <jerry@endocardial.com>, 30 Aug 1999
- ael project_history − rewrite as a report and then include parents and sub branches. Don’t forget the web reports, too. Submitted: Jerry Pendergraft <jerry@endocardial.com>, 30 Aug 1999
- aer file_history − include parents and sub branches. Don’t forget the web reports, too. Submitted: Jerry Pendergraft <jerry@endocardial.com> 30 Aug 1999
- Some kind of web report which makes “train track” diagrams of file branching.
- Some kind of web report which makes “train track” diagrams of project branching.
- multivariate linear regression: needed as a report generator function: needed for metrics analysis
- more blurb in the statistics web pages, so they are more self-explaining Submitted: Ralf Fassel <ralf@akutech.de>, 13-Oct-98
- Add anew report like “ael uc” except that it (optionally) takes a user name as well, to list a particular user’s changes.
- File Activity Report (web) does not translate user name and give email link. Should also put user name under change state, as in change lists.

10.12.5. Core Enhancements

- Use the per-file attributes to record the encoding of the text (e.g. UTF-8) and the line termination. Provide a way (via the iconv(3) function? via recode(1) command?) to change the encoding. Submitted: PMiller, 23-Jun-2004
- "I have determined that one reason [that aedeu is used in preference to aerfail] is the reviewer is afraid they don’t understand the change and once explained they would not fail it. Now the fact that the description, comments etc did not do the job to explain the change is reason enough to fail it notwithstanding... They are saying if they had an aerfu command they would be willing to aerf changes. How difficult would that be?" Not very difficult at all. Provided, of course, that nothing has been changed in the mean time (and Aegis has everything it needs to check that). Submitted: Jerry Pendergraft, 23-Jun-2004
- The project_file_roll_forward function needs to be enhanced to understand file UUIDs.

Submitted: PMiller, 1-Jun-2004

- Now the sources are all being compiled by a C++ cimpiler, convert the various OO portions of the code (inout_ty and its derived classes, output_ty, etc) to true C++. Need a style guide first, so other developers know how I want it done. See SRecord for examples until this is done. Submitted: PMiller, 1-Jun-2004
- More doxygen comments in the header files. Submitted: PMiller, 23-Jan-2004
- Add a "development directory style" configuration option. The current styles are "view path" and two types of "symlink farm", although this is well concealed by the code. Need to add a hard link (arch-ish) / copyfile (cvs-ish) style as well, but only for source files. The code which currently maintains the symlinks can be pressed into doing the extra work fairly easily. Submitted: PMiller, 1-Jun-2004
- It would be nice to have a way to specify a timeout for aegis tests. If a single test does not finish within this time, it should be aborted and considered ‘No Result’. Then aet should continue with the next test (as appropriate if −persevere was given). A ‘−timeout’ argument to ‘aet’ would do the trick, and also a project config field. The implementation could be interesting, since signaling the forked aegis child process might not be enough to stop all processes (process groups?). Submitted: Ralf Fassel <ralf@akutech.de>, 24-Jan-2001
- Problem with aepa that doesn’t specify the default values for all the test features in aeca (there are three types in aeca and only one in aepa). Submitted: Mark Veltzer <mark2776@yahoo.com>, 16 Aug 2001
- The *aedist*(1) program should send changes with no files, or changes in "being developed". Submitted: Mark Veltzer <mark2776@yahoo.com>, 16 Aug 2001
- Have *aem* merge changes properly if another changed moved the file in the baseline. You need to do this across the board, not just in `aegis/aed.c` Submitted: Ralf Fassel <ralf@akutech.de>, 25 Feb 2000
- Add progress (%) indicators (aeib was specific example, but there may be others *e.g.* symlink farms and aecp, even aede for big changes) for use by the GUI interfaces − and maybe the text interface too. Submitted: Ralf Fassel <ralf@akutech.de> 10 Dec 1999
- Extend the create_symlinks_before_build functionality to copy, not just symlink. Because they would edit the files direct, we then need an implicit aecpcorne detector. You need to look for other boundary conditions this is also going to affect. You need a remove_symlinks_after_build analogue, too. Submitted: Darrin Thompson <dthompson@characterlink.net>, 15 Nov 1999
- os.h is a system header on some systems, so os.h has to move Sumbitted: Christophe Broult <broult@info.unicaen.fr> 30 Sep 1999
- aedist −rec needs to preserve (a) copyright years, (b) test exemptions (subject to permissions), and (c) architecture (if possible). AND CHANGE NUMBER? Submitted: Ewolt Wolters <ewolt@pallas-athena.com>, 27 Jul 1999
- Aedist to add project history to end of description when sending change set. Submitted: Jerry Pendergraft <jerry@endocardial.com>, Dec-2000
- can we separate change creation from other administrator permissions? can we make "everyone" able to create changes? Submitted: Ewolt Wolters <ewolt@pallas-athena.com>, 28 Jun 1999
- should explicitly mention CPPFLAGS=−I/usr/local/include; export CPPFLAGS LDFLAGS=−L/usr/local/lib; export LDFLAGS in the configuring section. Submitted: John Huddleston <jhudd@cody.itc.nrcs.usda.gov>, 19 Mar 1999
- Using file attributes, add coupling between files to form file groups; this means when you aecp, you get the whole set of related files. Submitted: PMiller, 18-Feb-99
- The aed command does not promote aenf→aecp unless the ,D file does not exist. This is annoying, should always do it. (So should some other commands.) Subject: Ralf Fassel <ralf@akutech.de>, 1 Feb 1999
- Add a default_regression_test_exemption project attribute. Submitted: Ralf Fassel <ralf@akutech.de>, 31 Jan 1999, Jerry Pendergraft <jerry@endocardial.com>, 2 Feb 2001.
- Need a clean_exceptions file in the project config file (list of strings) so can have local RCS dirs, and do "ci ‘aegis −l cf −ter‘" in the develop_end_command Submitted: 1-Feb-99

- aenpr −dir −keep: allow directory to already exist if has right owner and is empty? Submitted: Jerry Pendergraft <jerry@endocardial.com>, 22 Jan 1999
- Add a new post_merge_command so can generate summary of files needing editing. Subject: Ralf Fassel <ralf@akutech.de>, 21 Dec 1998
- Create a new aepatch command: “aepatch −send” to create "ordinary" OpenSource source patches, and “aepatch −receive” to turn patches into an Aegis change − and not necessarily only patches generated with aepatch. Yes, intentionally similar to aedist.
- integrate difference should look for missing ,D files (usually impossible) and re-instate them. Submitted: PMiller, 22-Sep-98
- tests 7, 20, 70 warn symlink.c: In function ‘main’: symlink.c:5: warning: return type of ‘main’ is not ‘int’ Submitted: Bruce Stephen Adams <brucea@cybernetics.demon.co.uk>, 10 Sep 1998
- change_set_env needs to set LINES and COLS
- commands which accept −branch and/or −trunk should also accept −grandparent but not all do. check.
- Add a −no-baseline-lock option to the aeb and aecp commands. Warn them not to in the manual pages.
- list locks − need to spot the case where *all* of a set are taken (all 64k) and report sensibly (not 64K lines)
- aemv does not correctly check the *to* filename. (specific example = file name length)
- aefind needs a sort option
- aefind needs the rest of the find functionality added
- * Add a −output option to the aent command (*others?*) for scripting support.
- aed − when auto upgrade create to modify, clear move if set.
- aede needs to make sure that the files (and directories) are readable (and searchable) by reviewers.
- make aemv rename files within a change
- aecp −anticipate
- Make the listing more specific for aecp aecpu aenfu aentu aerm aermu, etc
- add a file copy notification command to the project config file
- Add pseudo change do can do many integrations at once (this pseudo change would be created by aeib and destroyed by aeipass, aeifail or aeibu).
- Version punctuation: at the moment you gets dots between the branch numbers. Need more flexible punctuation: especially, want a hyphen first, then dots (sometimes).
- * aecp −delta bug
  “I’ve been making good use of the "−delta" option of aecp lately.
  But there has been a complication in its use. Let’s say a file
  was aerm’ed in delta 100. Let’s further say that we are at delta
  175 and are trying to restore the source code as of delta 75.
  If I do a "aecp − delta 75 file.c" I’m told that file.c is no
  longer part of the project.” Should aecp −del fake aenf for deleted files in earlier deltas? Submitted: markm@endo.com
- internationalize −interactive
- Enhance aet to allow reviewers to run tests.
- check library state files on project creation
  “I was creating a new release from a large project.
  After copying the
  baseline and creating hundreds of history files the aenrls failed
  because the library dir I specified wasn’t writable by aegis and no
  state file was created. Couldn’t this be checked first?” Submitted: Lloyd Fischer <lloyd@dvcorp.com>
- Add precedence constraints: a list of prerequisite changes, which must all be in the “completed” state before the change may end development. Submitted: Christian F. Goetze <c-goetze@u-aizu.ac.jp>
- If there is a read error when reading the template source file during aent, get a stupid error within error message, and never tells you about the file
- How about "include" support for the config file? That way one could also cover architecture specific things by “include ${libdir}/${project}.defs” in the config file. Submitted: Jerry Pendergraft <jerry.pendergraft@endocardial.com>, 7 Sep 2001

- Add an *aetouch* command, to touch all of the (non-build) source files in the change. Submitted: 2001
- Have the “aeclean −list” option say what aeclean would do, rather than list the change source files. Submitted: 2001
- Have aed (aem) *remember* the previous state when it finds a problem (much like aet does, now). Submitted: Ralf Fassel <ralf@akutech.de> 3-Mar-2002

10.12.5.1. More O(1) Scalability

- Need to supplement the {project,change}_file_find and {project,change}_file_nth interfaces with {project,change}_file_name_nth interfaces. Then, use them as often as possible.
- Need the fstate file to have a manifest field; access this for file names. Then, store each file into in a separate file; only access this file is file state is required.
- The presence or absence of the manifest field in the top-level fstate file tells you if the old or new file state usage is present.

10.12.6. GUI

- tkaeca barfs when there are no changes on the branch. should be more graceful. Submitted: Ewolt Wolters <ewolt@pallas-athena.com>, 11 Aug 1999
- using tkaegis: project > branch > role > integrate, a window pops up "Error in tcl script, Error: invalid command name ".mbar.review.menu"". Submitted: Ewolt Wolters <ewolt@pallas-athena.com> 9 Aug 1999
- user pasted in text (including back slash) into aeifail edit window. which was accepted, but broke change state (illegal escape sequence). Submitted: Michael McCarty <mmccarty@xinetix.com>, 10 May 1999
- A new ${architecture_list} substitution to give all architectures in a command. Submitted: jerry.pendergraft@endocardial.com, 31 Mar 1999
- have aedist −rec accept a −delta option, so you can tell it where to apply from. Anticipated use is “−delta 0” meaning start of branch. (also a −reg option). Submitted: PMiller, 22-mar-99

10.12.7. Release and Build and Install

- add debian `.deb` file, add notification to <cd@debian.org> for new releases. Submitted: PMiller, 22-Jun-99
- building documentation needs to talk about libz some more. particularly, you either need it on ROOT’s LD_LIBRARY_PATH or you need to static link it. Submitted: Ralf Fassel <ralf@akutech.de> 5-Apr-99
- have configure script whine about missing libz Submitted: PMiller, 7 Apr 99
- have configure script whine about missing regcomp Submitted: PMiller, 7 Apr 99
- Sample documentation needs to make the *group* thing obvious. And also the umask at aenpr time! Submitted: Alan Zimmerman <alanz@electrosolv.co.za>, 5 Apr 1999
- generated makefile CC=cc needs to quote cc in case has spaces Submitted: Aaron Johnson <adj@ccltd.com>, 31 Mar 1999
- The BUILDING file needs to mention that you should install zlib with −−prefix=/usr because many systems think /usr/local/lib "insecure directory". Submitted: Fabien Campagne <campagne@Inka.MSSM.EDU>, 26 Mar 1999
- add piece to BUILDING file saying to get Apache first. Submitted: Graham Wheeler <gram@cdsec.com> 27 Jan 1999

10.12.8. Database

- Write an ODBC interface to the database? Submitted: P. Miller, 16 Aug 2001
- Does it make sense to have an NNTP interface? Would it be any use? Submitted: P. Miller, 16 Aug 2001

.

.