AIPS++ Administrator's Guide

Wes Young

2003/01/31

The AIPS++ consortium may be reached by email at aips2-request@nrao.edu. The postal address is: AIPS++ Consortium, c/o NRAO, 520 Edgemont Rd., Charlottesville, Va. 22903-2475 USA.

Contents

Chapter 1

Overview

1.1 Introduction

Greetings. This is the AIPS++Systems Administrator's Guide. It's a nuts and bolts guide to running the system. There are lots of little niggly details that need to be accounted for by the AIPS++Systems Administrator. Duties can be broken down into several categories

- Daily
- Weekly
- Release
- Patching the Release
- Miscellaneous

The AIPS++Systems Administrator needs a wide range of tools in his/her toolbox

1. GNU make
2. C++, C, FORTRAN
3. perl/awk
4. latex/latex2html
5. RCS/CVS
6. shell scripting

1.2 Background

Accounts used by the AIPS++system administrator

aips2mgr for day-to-day building of the system

aips2adm for managing the code repository

aips2msg for email to the various AIPS++system accounts

ddts for managing ClearDDTs

Machines for building AIPS++

tarzan - solaris builds, also home of all things AIPS++.

cluster - linux builds

it - linux 64b builds

argus - IRIX builds

In general cron jobs are run on tarzan, although there is a seperate build/code tree on argus for the SGI.

Chapter 2

Daily Duties

2.1 Check the build health

A build health script runs daily reporting errors of various builds around the world. Up to 30 errors will be displayed. Look at this report and act accordingly. Responsibility for linux, solaris native and gnu, and sgi reside with AIPS++central.

2.2 Check aips2msg

A seperate account exists for recieving email intended for aips2mgr and aips2adm, it's called aips2msg. Aips2mgr and aips2adm have no deciperable password and access is granted via ssh keys. We also manage the aips2 email exploders and mailman requests will come to this account and need to be dealt with accordingly.

Chapter 3

Weekly Duties

3.1 Check for stable

There are two opportunities to declare a stable each week. The first occurs with Thursday daily cumulative build, the second with Friday's weekly build. We check Thursday's build and if it there are problems which might prevent a stable, we try and address them with the hope that the weekly cumulative might be "stable".

These build constitute the stable candidates. The criteria for a stable are:

1. Must be a cumulative
2. All aips unit tests should pass. Tests may fail but the failure must be understood and considered to be minor (i.e. similar to tDirectory failure on linux with NFS mounts).
3. assay().trytests() runs without failure.
4. dishlongtests() runs without failure.
5. no obvious failures in loading the help atoms

So what does this entail?

1. Examine build logs for glaring problems.
2. Look at the unit tests
   (a) linux_gnu
   (b) sun4sol_gnu
   (c) sun4sol_ntv
3. Run assay
   (a) linux_gnu
   (b) sun4sol_gnu
   (c) sun4sol_ntv

If the build is "stable", then as aips2adm on tarzan

1. Send an email to aips2-naug aips2-aoc warning of the impending build

   ```
   Greetings,

   Build xx.yyy has met the stable criteria.  We will begin building the new stable @hhmm MT.
   Please use weekly until the new stable build has completed.

   aips2mgr
   ```

2. as aips2adm;

   ```
   cd /export/aips++/scripts; sh ./do\_blessing xx.yyy
   ```

   This scripts marks xx.yyy as stable; inhales the stable into /aips++/stable; and begins the stable build for linux and solaris. The local sgi stable build is triggered by it's own cron job.

3. Once the stable build completes, as aips2mgr on tarzan;

   ```
   cd; cd scripts; ./updatestables.sh
   ```

   This will update the stable builds area on /home/tarzan5 making binaries available to users via aupdate.

4. Send an email to aips2-sitemgr

   ```
   Greetings,

   Build xx.yyy has been declared stable.  Please update your stable releases.

   aips2mgr
   ```

It's best to declared a stable no later than Monday afternoon, so remote sites can inhale the latest stable Monday evening.

3.2 Check the Documentation

A rudimentary documentation check consists of

1. Point your web browser at the daily/weekly docs tree,
2. Pick the reference manual, and
3. For each package, select various modules and tools.

More thorough testing of the docs can be done with linkscan. Linkscan will produce a series of reports most useful of which are missing and broken links in the docs.

Chapter 4

Preparing the Release

4.1 General timeline

There are several steps involved going from active development to generating a release ready ISO image. In general it takes a about of 5 weeks, although if things go well it may be possible to squeeze the whole process into three weeks.

-5 weeks Slushy Freeze

-3 weeks Hard Freeze

-2 weeks 2 days Uprev

-2 weeks Generate Release Candidate

- Make release build
- Test release build
- Patch
- Make ISO image
- Test ISO image

0 Release to Duplicators

4.2 Slushy Freeze

During the slushy freeze, active development is suppose to cease and only changes that address defects are suppose to be checked in to the system. During the slushy freeze state, we actively check for stable builds everyday.

4.3 Hard Freeze

Ideally the "Hard Freeze" occurs with a stable. At this point the current development master will become the release master once the "upreve" is completed.

4.4 Doing the uprev

The uprev is where we actually sync the current release master with the current development master. There are several steps involved in doing the uprev and several days should be set aside in case something goes wrong. If everything runs smoothly the uprev typically takes about two days.

- Check the disk space. The worst thing that can happen is to run out of disk space during the uprev. Make sure a minimum of 2GB (peferrably 3GB) is available.
- Send a early morning/late night [to give Oz a chance to respond] notice to aips2-workers telling them that the uprev and check-in freeze will happen later today/tomorrow.
- Notify aips2-workers so that check-ins will be disabled in 30 minutes to allow any in progress check-ins to be completed.
- (adm) Freeze check-ins on the development master repository by running:

```
/export/aips++/scripts/do_disable develop
```

- (adm) Stop the regular exhales in the (aips2adm) crontab; if the misery drags on, the regular exhale can gum up the works
- (adm) Do an initial exhale to ensure that everything is up to date:

```
ksh
. $HOME/.profile
exhale 2>&1 | mailx -s "AIPS++ (initial) uprev exhale" aips2-inhale
```

- (mgr) Update the data repository in release installation, e.g.

```
 cvsup /usr/local/aips++/data/supfile
```

- (mgr) Run cumulative inhale (after exhale completes) on local installation
- (user) run "stable" tests
- (adm) Back up release master (/export/aips++/repository/release), if desired.
- (adm) Delete the release master and copy development to release:

```
cd /export/aips++/repository
rm -rf release
mkdir release
cd release
(cd ../develop; tar cf - .) | tar xf -
```

- (adm) In /export/aips++/repository/release, edit:

```
.cshrc  .login  .profile  aipsinit.csh
aipsinit.es  aipsinit.rc  aipsinit.sh
replacing each occurrence of /develop with /release remove editor backups
```

- (adm) Update ftp link, VERSION, and base release tar file (where ¡REL¿ is the (last) base release upon which the release master is based):

```
cd /export/aips++/repository/release
rm pub
ln -s /export/aips++/pub/versions/release pub
rm -f pub/master/VERSION
cp master/VERSION pub/master
cp ../develop/pub/master/master-<REL>.000.tar.gz pub/master
/opt/local/gnu/bin/touch --file ../develop/pub/master/master-<REL>.000.tar.gz pub/master/master-<REL>.000.tar.gz
```

- (adm) Run exhale on the release master:

```
ksh
. /export/aips++/repository/release/.profile
exhale 2>&1 | tee $HOME/master/etc/rexhale.log | mailx -s "AIPS++ release exhale" aips2-inhale &
```

- (adm) Do uprev (note, backup of VERSION is kept in VERSION.¡REL¿):

```
/export/aips++/scripts/do_uprev
```

- (mgr) Run test inhales for both the release and develop masters to ensure that they work properly.
- (adm) Enable regular exhales in the (aips2adm) crontab
- (adm) Unfreeze check-ins on the development master repository by running:

```
/export/aips++/scripts/do_enable develop
```

- (adm) Unlock any locked files in the release RCS tree

```
/export/aips++/scripts/do_unlock.sh
```

4.4.1 Update the Docs

1. Update the README and readme.html to current versions
2. Update the release notes
3. Run a cross-referencing build of latex documents.

4.5 Testing the build

Testing the build is broken down into two phases,

1. Test the release build
    - As aips2mgr; run testsuite

   - As a user; run assay().trytests()
   - As a user; exercise the tool manager
   - As aips2mgr; run linkscan

2. Test an installed ISO image

   - Excersize the install procedures for linux and solaris
   - run assay().trytests
   - excersize the tool manager
   - take a random walk through the documentation

4.6 Patching the release candidate

Once the release master has been updated, patches for defect resolution and documentation will be sent. These patches are suppose to be sent a binary shar files (shar -B dir/thefile ¿ myfile.shar). In general only defects fixes are allowed for .cc, .h and .g files. Documentation is always welcomed. Steps for patching always done as a user:

- Make a seperate code tree
- Save the shar files into a directory, i.e. /home/tarzan5/wyoung/patches/jun21.
- Edit the files to remove non-shar file text
- Unpack the shar files into a clean code tree,

  ```
  cd /home/tarzan5/wyoung/patches
  ./unpack.sh jun21 2&>1 | tee patch.log
  ```

  Check the log to make sure everything unpacked cleanly. If there are problems notify the sender.

- Patch the release master
  - (adm) Enable checkins in the release master; /export/aips++/scripts/do_enable release
  - (user) Apply the patches

    ```
    cd /home/tarzan5/wyoung/aips++
    ../patches/do_patch.sh
    ```

  - (adm) Disable checkins in the release master; /export/aips++/scripts/do_disable release
  - (adm) Exhale the release
  - (mgr) inhale -R release -c
  - (mgr) check the build log for problems
  - (user) Notify aips2-workers of accepted patches

```
cd /home/tarzan5/wyoung/patches
awk -f okpatch.awk jun21/* > patch.mailing
mailx -s "Accepted patches for release" aips2-workers@aoc.nrao.edu < patch.mailing
```

Note: the okpatch.awk script will need to be modified with each new version.

4.7 Preparing the binary

4.7.1 The penultimate build

Once the release build is in good shape we need to produce the files necessary for the binary distribution. Binrel must be run on each architecture. One should avoid binrel on nfs mounted disks as it takes considerably longer.

linux

1. . /usr/local/aips++/aipsinit.sh
2. binrel -tgt /home/cluster2/aips++/binary_release
3. cp linux.tar.gz /home/tarzan5/aips++/binary_release

solaris

1. . /usr/local/aips++/aipsinit.sh
2. binrel -tgt /home/tarzan5/aips++/binary_release

4.7.2 Creating the ISO and CD images

Typically we burn on the aips2 administrator's computer. This involves copying the source contents from the binrel/YY.XXX directory to the CD distribution directory. The current content for v1.7 are listed below. Note the INSTALL, README, and VERSION files need to be adjusted for each release.

```
[wyoung@crestone cds]$ ls -1 cd1/*
cd1/data.tar.gz
cd1/docs.tar.gz
cd1/INSTALL
cd1/lib.tar.gz
cd1/linux.tar.gz
cd1/README
cd1/startup.tar.gz
cd1/sun4sol.tar.gz
cd1/VERSION

cd1/source:
admin.tar.gz
aips.tar.gz
atnf.tar.gz
bima.tar.gz
contrib.tar.gz
dish.tar.gz
```

```
display.tar.gz
doc.tar.gz
hia.tar.gz
install.tar.gz
jive.tar.gz
nfra.tar.gz
npoi.tar.gz
nral.tar.gz
nrao.tar.gz
synthesis.tar.gz
tifr.tar.gz
trialdisplay.tar.gz
trial.tar.gz
vlbi.tar.gz
```

Once the release has been assembled in the proper directory, xcdroast is used to create both the ISO image and burn the CD. Note: the ISO/CD image should always be labeled aips2install. Xcdroast usually creates a file track-01.img as the name for the ISO image, rename it to something more sensible, i.e. aips2vXY.iso.

The ISO image maybe tested using the loopback file system

```
mount aips2vXY.iso /mnt/iso -o loop=/dev/loop3,blocksize=1024
```

Now test install and verify both the ISO and CD images.

Chapter 5

Patching the Release

5.1 Assemble the patches

5.2 Build

5.3 Test the patch build

5.4 Prepare the binary patch

5.5 Test the binary patch

Chapter 6

Miscellaneous Duties

6.1 ClearDDTS

We use ClearDDTS as our defect tracking system. There is fairly extensive written and web base documentation for the administrator. ClearDDTs defects reside in /users/ddts and is licensed to run on tarzan. A cron job runs four times a day looking for lost defects.

From time-to-time, and email address will need to be corrected. The procedure follows:

1. logon on to tarzan as ddts
2. run patchbug .i.e.

   ```
   patchbug -a Submitter-mail right@address AOCsoXXXXX
   ```

3. run adminbug; and inside adminbug run dbms.

Note: this can be done for any field in the ClearDDTs bug system. I suggest consulting the ClearDDTs Administrator's Guide.

6.2 Linkscan

We use linkscan software to scan the AIPS++ web pages for broken and missing links. As aips2mgr on tarzan; cd /home/tarzan/httpd/linkscan; ./linkscan.pl. Once the script has run it produces a series of reports that can be accessed at http://aips2.nrao.edu/linkscan.

Configuration scripts for linkscan are found in /home/tarzan/httpd/linkscan/default and userdoc. Default is setup to scan the entire aips++ web site, where userdoc will scan only the release doc tree.

6.3 Htdig

Htdig provides the search engine for the AIPS++ site. Several cron jobs (run as wyoung) on tarzan that scan the AIPS++ code and docs tree. Configuration files are found in /aips++/local/htdig-3.1/conf. Details on the care and feeding of htdig maybe found at it's web site http://www.htdig.org.

6.4 Email

Several email aliases are maintained on tarzan in /etc/mail/tarzan.aliases-tail. The aliases are update once a day around 0300 and require root privileges to make any changes.

We use mailman (maintained in Charlottesville) to handle the mailing lists although messages are sent to local aips++ logs in /export/aips++/Mail on tarzan.

6.5 Compiler updates

One of they joys is porting to new compilers or updating the compiler. Use the following change proposal template to initiate the process.

```
Title:                  Adopt gcc X.X as project compiler
 Person responsible:     The Boss (bossdude@nrao.edu)
 Originator of proposal: The Worker (workerdude@nrao.edu)
 Exploders targeted:     aips2-lib
 Time table:
 Date of issue:          200X Mth XX
 Comments due:           200X Mth XX
 Revised proposal:       200X Mth XX
 Final comments due:     200X Mth XX
 Decision date:          200X Mth XX

 Statement of goals:

 Adopt gcc X.X as project compiler.

 Background:

 Gcc-X.X is the latest version of the gcc compiler series.  It has been adopted as the default
 compiler by most of the major Linux releases.

 [--- additional back ground text goes here ---]

 For those looking for further details please visit http://www.gnu.org/software/gcc/gcc.html and
 http://www.gnu.org/software/gcc/gcc-3.1/changes.html

 Summary:
 --------

 All that should be required is a switch to the new compiler in the local makedefs.


 Expected Impact:
 ----------------
```

```
[ Address the following points for both solaris and linux ]

Changes to exsisting code:
    [ Detail any changes to existing code here, in particular what needed changing and possibly why
      include filename and line numbers ]

Unit test conformance:
    [ Run runtests and report differences, and how to resolve unexpected failures (with modified code)]


Assay results:
    [ Run the assay, should give the same results as the old compiler ]

Benchmark tests
    [ Run a set of benchmarks, current test is imagertester().runtests('mftest'), possibly get a more
      complete set from Sanjay. ] Summarize results.

Support Tools
   [ Try a few of our standard tools and see if there are any problems and what work arounds may be
     necessary ]

   Rational
   Debugger

---------------------------------------
```