The Cuddletech Guide to Building OpenSolaris

Ben Rockwood

June 14th 2005

1 Concepts

The OpenSolaris build system is based on the internal Sun Solaris development methods and proceedures but modified for those of us who aren't connected to the Sun internal network, otherwise known as SWAN (the Sun Wide Area Network). Because of this, many methods for synchronizing code and updating or veryify code varies from the accepted internal methods. Please understand that kinks are still being found and worked out.

Internally at Sun code is managed using TeamWare, a old componant of Sun Workshop based on SCCS. Code is made avalible via NFS in a golden state, known as a *gate*. The gate contains all the code and is carefully maintained as a fully tested and stable codebase and is maintained by a *gatekeeper*. When you wish to develop, build, or test code you need to setup a local copy for manipulation known as a *workspace*. The workspace is really nothing more than a directory that acts as the root for the directory structure that will be populated with the codebase, binary builds, etc. To populate this local workspace a *bringover* is performed, this essentially copies the contents of the gate into your local workspace. Once code modifications have been reviewed, tested, and approved a reciprical operation known as a *putback* is used to add the changes to the gate. The gate itself is really just a carefully managed workspace, and can be updated from yet another gate, forming a mutli-teered structure of gates if needbe.

Because those of us in the OpenSolaris project don't have access to the Sun gates, we must download code in tarballs or from CVS and populate our workspace manually. Your local workspace will contain the unemcumbered source tree (usr/src/), the encumbered binaries (closed/), as well as provide the build tools with a place to output built binaries (proto/), store installable CPIO archives (archives/), and keep logfiles (log/). Additionally, in the top level directory of your workspace a build configuration file named *opensolaris.sh* should be present which contains enviromental variables that are utilized by the various build tools.

The Solaris Operating Enviroment is a combination of a variety of diffrent componants, such as X11, the Java Desktop Enviroment, etc. These componants are known as *consolidations*. Essentially, OpenSolaris is the *Operating System* *and Networking* consolidation of Solaris, refered to as either *OS/Net* or just *ON*. When all the considations are put together they are collectively refered to as a *WOS*, or "Wod of Stuff", the binary distribution of Solaris that would be put onto CDs and shipped.

Solaris releases are distinguished by build numbers. The final Solaris Express released in Nov of 2004 was build 72 (or just B72). The released version of Solaris10 FCS is B74. OpenSolaris is based, at least initially, on B74L2 codenamed *Nevada* or thought of was Solaris 10.1. Therefore, OpenSolaris is based on the codebase being developed on for the release following Solaris10. At this time it is unknown as to whether this release will be Solaris11 or Solaris 10.1, but for now the codename Nevada suffices.

2 Preparing a development workstation

Install the latest build....... install the tools....... blah.

3 Backing up your system using Flash Archives

OpenSolaris differs significantly from Linux in the sense that when you build OpenSolaris you are building the entire base operating system (the ON consolidation) and not simply the kernel. Because of this fact an upgrade of your system by means of BFU will overwrite large portions of your system, and as such a bad build can result in an unusable system which we jokingly refer to as a *brickified* system.

To protect yourself from turning your system into a brick you can use two Solaris provided methods of protect. The first is Live Upgrade, a mechanism that can duplicate your root disk to a secondary internal drive, so that in the event of an unbootable system you simply boot the *alternate boot enviroment* (ABE). The second method is Live Upgrades sister, Flash Archives. Flash Archives are intellegent cpio archives of your system typically stored on a remote NFS server. When you install Solaris from CD you'll notice that you'll be asked at some point whether you want to do a "New Install", "Upgrade" or "Flash Install". The latter option allows you to, from the boot CD, reinstall your system using a previously saved Flash Archive. This provides a useful and quick backup method whenever you make major changes, such as upgrading from one build to another, or doing heavy modification of the system that you fear may impact the existing system negatively.

To create a Flash Archive (flar) of your existing system use the following method:

1. Mount a remote NFS server to the local system. If the NFS server is a Linux system ensure that you specify the NFS version during mount:

   ```
   $ mkdir /flash
   $ mount -F nfs -o vers=3 10.10.2.247:/nfs/raid0 /flash
   ```

2. Now use the *flarcreate* command to create the archive itself. In the following example I'm using the -n option to specify a description for the archive, -a to specify the contact address for this archive, -R specifies the root directory, -x excludes a filesystem, and the final argument is the file name and path for the flash archive itself.

   ```
   $ flarcreate -n "Monolyth SX B16" -a "benr@cuddletech.com" \
   > -R / -x /flash /flash/monolyth-‘date ’+%m-%d-%y’‘.flar
   Full Flash
   Checking integrity...
   Integrity OK.
   WARNING:  fdo: Ignoring duplicate filter entry. Choosen entry will be: /flash -
   Running precreation scripts...
   Precreation scripts done.
   Determining the size of the archive...
    ... snip ...
   ```

This proccess can be automated via cron if you wish to use flar as a nightly backup solution, although its not the best method it does provide a nice point-in-time snapshot.

When creating your archives you should ensure that you exclude all network mounted filesystem. Fhe flarcreate command itself should exclude all network mounts, however due to some historical bugs its still common practice to error on the safe side and exclude them explicitly. Multiple excludes are provide seperately, so multiple excludes looks like this: -x /flash -x /opt -x /export.

Please be aware that depending on the size and speed of your disk the archive can take a very long time to create, especially during the size checks. I would highly recommend that flash archives be something you kick off before heading to bed to avoid any performance impacts.

4 Customizing GRUB boot options

5 Building the entire OS/Net codebase

The entire codebase can be easily build using the *nightly* utility. This utility will grab the enviromental variables defined in your opensolaris.sh file and proceed to build the entire codebase. Nightly is a safe way to manage builds because it is complete, maintains good logs of the build, and always cleans up the codebase prior to building (technically, it does a "make clobber" in the top level source directory usr/src).

(Add this stuff)

While the nightly build is running you can verify some of its settings in the */tmp/nightly.tmpdir.*/mail_msg* file. You can also look at dmake output messages in the /tmp/dmake* temp files; they are created and deleted quickly but a 'cat /tmp/dmake*' during the build can give you peaks at the build output to determine if there are problems early on rather than waiting for the nightly to complete. Additional *.out* logs can be found in *usr/src/*.

Once a nightly build is completed you'll recieve a mail containing the status of the nightly build. This message as well as a verbose log can be found in the *log/* directory of your workspace. The following is an example of the mail you would recieve on sun4u after a successful build:

```
==== Nightly distributed build started:   Sat Jan 22 04:11:22 PST 2005 ====
==== Nightly distributed build completed: Sat Jan 22 09:21:07 PST 2005 ====

==== Total build time ====

real    5:09:45

==== Nightly argument issues ====

Warning: the N option (do not run protocmp) is set; it probably shouldn't be

==== Build environment ====

/usr/bin/uname
SunOS betty 5.10 Generic sun4u sparc SUNW,Ultra-2

/opt/onbld/bin/nightly ./opensolaris.sh
nightly.sh version 1.73 2005/01/10

/opt/SUNWspro/bin/dmake
dmake: Sun Distributed Make 7.3 2003/03/13
number of concurrent jobs = 4

/opt/SUNWspro/bin/cc
cc: Sun C 5.5 Patch 112760-07 2004/02/03

64-bit compiler
/opt/SUNWspro/bin/cc
cc: Sun C 5.5 Patch 112760-07 2004/02/03

/usr/java/bin/javac
java full version "1.5.0_01-b08"

/usr/ccs/bin/as
as: Sun Compiler Common 10 s10_73 11/23/2004

/usr/ccs/bin/ld
ld: Software Generation Utilities - Solaris Link Editors: 5.10-1.477
```

```
Build project:  default
Build taskid:    52

==== Build version ====

cuddletech

==== Make clobber ERRORS ====


==== Make tools clobber ERRORS ====


==== Tools build errors ====


==== Build errors (DEBUG) ====


==== Build warnings (DEBUG) ====


==== Elapsed build time (DEBUG) ====

real  4:46:25.6
user  6:39:22.3
sys   1:11:53.5

==== Build noise differences (DEBUG) ====


==== cpio archives build errors (DEBUG) ====

Copying /export/workspace/closed/root_sparc to /tmp/bfu20993...
Copying /export/workspace/proto/root_sparc to /tmp/bfu20993...

==== Check ELF runtime attributes ====

./usr/lib/lddstub: ldd: /export/workspace/proto/root_sparc/usr/lib/lddstub: execution fa
./usr/lib/sparcv9/lddstub: ldd: /export/workspace/proto/root_sparc/usr/lib/sparcv9/lddst

==== Diff ELF runtime attributes (since last build) ====


==== Find core files ====
==== Impact on file permissions ====
```

In the above mail, the ldd errors can be safely ignored.

If a nightly build runs into serious problems it can muck up the ouptput of the next build you run by including diff information in the logs and status mail. To avoid these problems you can clean up the workspace by emptying the *log/*, *archives/*, and *proto* directories. The ”make clobber” preformed before each nightly should take care of the source tree.

6 Installing Kernels & Modules with Install

The *Install* (capital I) utility can be used to install the kernel and its modules from your workspace. The advantage of Install over BFU is that only the kernel and modules are installed and they can installed into an alternate location without having to overwrite the existing kernel, that way if your kernel is unusable you can safely boot the origonal kernel without having to recover the system.

The following steps outline the proccedure for using Install following a successful nightly build:

1. Use *bldenv* to update your build enviroment:

   ```
   $ bldenv /export/workspace/opensolaris.sh
   Build type   is  non-DEBUG
   RELEASE      is
   VERSION      is  cuddletech
   RELEASE_DATE is

   The top-level 'setup' target is available to build headers and tools.

   Using /bin/bash as shell.
   $
   ```

2. Use the *Install* tool to create a tarball including the modules and kernel. The -G argument specifies the subdirectory that will be used for installation ('-G testing' would install to /platform/sun4u/testing/...), -k specifies the machine hardware ('uname -m'), and -K specifies that kmdb should not be copied (this is required at the moment, see the RelaseNotes).

   ```
   $ Install -G testing -k sun4u -K
   ....
   test -f /tmp/Install.benr/sun4u/platform/sun4u/testing/sparcv9/unix
   ln -s sparcv9/unix /tmp/Install.benr/sun4u/platform/sun4u/testing/unix
   Creating tarfile /tmp/Install.benr/Install.sun4u.tar
   Install complete
   ```

3. After *Install* completes it will specify the location of the tarfile it created. You can move this tarball to the system you want to test the kernel on, if it's not the local machine, and untar it from root:

   ```
   $ tar xfv /tmp/Install.benr/Install.sun4u.tar
   x ., 0 bytes, 0 tape blocks
   x ./platform, 0 bytes, 0 tape blocks
   x ./platform/SUNW,Ultra-2, 0 bytes, 0 tape blocks
   ....
   x ./platform/sun4u/testing/pcbe/sparcv9, 0 bytes, 0 tape blocks
   x ./platform/sun4u/testing/pcbe/sparcv9/pcbe.62, 34296 bytes, 67 tape blocks
   ./platform/sun4u/testing/pcbe/sparcv9/pcbe.23 linked to ./platform/sun4u/testing/pcb
   x ./platform/sun4u/testing/unix symbolic link to sparcv9/unix
   ```

4. You can now attempt to boot the new kernel by either specifying it as an argument to the *reboot* command or via the bootloader/OBP:

   ```
   # reboot -- 'testing/unix'
   (or)
   ok  boot testing/unix
   ```

7 Installing Builds with BFU

BFU, the *Blazing Fast Upgrades* utility, can handle the task of installing a build on top of an existing installation. Once a nightly build has completed, BFU can be used to easily and quickly upgrade your system to the newly compiled code. Typically once a nightly build was ready to be tested you would simply "BFU your system" and reboot, however, the BFU upgrade will copy over both your existing utilities and kernel, meaning that a bad BFU could leave you with a system that is unusable or *brickified* (*brickifying* is the official term for the proccess of inadvertantly turning your system into a brick - a system that is either unbootable or unusable). To avoid turning your system into a brick various methods can be utilized to duplicate the root disk and then use the alternate enviroment as either the destination for the BFU or as a recovery rootdisk; LiveUpgrade can serve this purpose well.

Below are the steps involved in setting up an *alternate boot enviroment* (ABE) using LiveUpgrade and then BFU'ing the ABE. Please note that Live Upgrade requires that you have an additional disk installed that is at least as large as the current root disk. If you do not wish to use LiveUpgrade, simply skip the LiveUpgrade related steps and just utilize *bldenv* and *bfu*, then reboot and cross your fingers.

1. Copy the partition table of the root disk to the disk that will be our ABE:

   ```
   $ prtvtoc /dev/rdsk/c0t0d0s2 | fmthard -s - /dev/rdsk/c0t1d0s2
   fmthard:  New volume table of contents now in place.
   ```

2. Create the LiveUpgrade ABE using *lucreate*. -c specifies the name to identify the current boot enviroment (BE), -m specifies the vfstab information for the ABE disk, and -n specifies the name of the new BE:

   ```
   $ lucreate -c "S10-B74L2" -m /:/dev/dsk/c0t1d0s0:ufs -n "S10-Testing"
   Discovering physical storage devices
   Discovering logical storage devices
   ....
   Making boot environment <S10-Testing> bootable.
   Population of boot environment <S10-Testing> successful.
   Creation of boot environment <S10-Testing> successful.
   $
   $ lustatus
   Boot Environment           Is       Active Active    Can    Copy
   Name                       Complete Now    On Reboot Delete Status
   -------------------------- -------- ------ --------- ------ ----------
   S10-B74L2                  yes      yes    yes       no     -
   S10-Testing                yes      no     no        yes    -
   $
   ```

3. Now create a mountpoint for the ABE and mount it:

   ```
   $ mkdir /a
   $ mount /dev/dsk/c0t1d0s0 /a
   ```

4. Update your build enviroment using the *bldenv* utility:

   ```
   $ bldenv /export/workspace/opensolaris.sh
   Build type   is  non-DEBUG
   RELEASE      is
   VERSION      is  cuddletech
   RELEASE_DATE is

   The top-level 'setup' target is available to build headers and tools.

   Using /bin/bash as shell.
   $
   ```

5. Now BFU the ABE by using the *bfu* utility. The first argument specifies the location of the nightly archive to be used and the second argument optionally specifies root directory for the BFU destination; if you are not using LiveUpgrade you do not require the second argument:

   ```
   $ bfu /export/workspace/archives/sparc/nightly /a
   Copying /opt/onbld/bin/bfu to /tmp/bfu.546
   ...
   ```

```
Entering post-bfu protected environment (shell: ksh).
Edit configuration files as necessary, then reboot.

bfu#
```

6. When BFU finishes updating, you'll be presented with a list of files that conflict and a korne shell (a bfu# prompt) in which you can safely resolve each of the conflicts. You should resolve any conflicts that look like a problem before continueing. Type *exit* to exit the shell when done.

7. You can now unmount the ABE and activate it:

```
$ umount /a
$ lustatus
Boot Environment           Is       Active Active    Can    Copy
Name                       Complete Now    On Reboot Delete Status
-------------------------- -------- ------ --------- ------ ----------
S10-B74L2                  yes      yes    yes       no     -
S10-Testing                yes      no     no        yes    -
$ luactivate S10-Testing
....
```

8. Finally, reboot the system using shutdown. *DO NOT* use the *reboot* command; if you do the BE will not change and you'll have to reactivate it and reboot properly:

```
$ sync;sync;shutdown -y -g0 -i6
```

If everything works properly the new boot enviroment that was BFU'ed should boot up safely. If it doesn't boot properly or the kernel crashes you can simply boot to the origonal boot disk and resolve the issues.