Automated source branch selection using SH/CVS

Allan Fields
Afields Research / AFRSL

http://afields.ca
bsd@afields.ca

Outline

- Introduction
- The FreeBSD Source Code
- Automation: Bringing it all together..
  - Why Automate?
  - Source Tag Selection
  - Design Goals
  - Multiple Releases
  - Supported Methods
  - Extracting Tags from CVS
- Why SH?
  - Minimal SH & Standard Unix Tools
  - Lessons on Complexity
- Further Ideas for Updates
- Q&As

Introduction

- Open Source operating systems allow universal access to source code
- Most commercial systems employ binary-only updates, patches or service-packs that replace pre-compiled components with updated versions
- FreeBSD source code can be viewed and downloaded online using multiple methods
- Updating FreeBSD from sources is a simple procedure and offers a great deal of flexibility

The FreeBSD Sources

- The FreeBSD Sources consist of a number of packages which contain the source for various parts of the system
- The FreeBSD CVS repository contains:
  - System Source Code
  - Build Tools and Scripts
  - Libraries
  - System and User Binaries
  - Documentation
- Releases and Snapshots provide (ready to use) source packages

Update Methods

- FreeBSD source code is distributed & updated in numerous ways
- Locally: Off CD-ROM, over NFS, etc.
- Online: Source packages from FTP/HTTP mirrors; rsync; Obscure methods: CTM, etc.
- Repository Access:
  - CVS/Anonymous CVS (anoncvs), CVSup
  - P4 Perforce (for developers)
  - SVN Subversion (soon to be available?)
- Also viewable on web: CVSWeb, FXR

Revisions, Tags & Branches

- Source management: FreeBSD developers use version control systems like CVS which stores files in repository;
- **Revisions** are different versions of file through-out it's lifetime: each set of changes is a diff/changeset;
- Committing to repository is "check-in"; "check-out" is getting revisions from repo;
- **Tags** provide symbolic representations of revisions in repository & specify branches;
- **Branches** represent lines of development in code pointing to current revisions of files.

Working Source Trees

- **Working source tree** is checked-out copy of sources on which work is performed: building, patching, editing source files, etc.
- Basically: Copy of source code for local use
- Not to be confused with **object directory**: destination for objects, intermediary files during build process. Usually: `/usr/obj`
- Most users: working source trees used exclusively for purpose of building system
- Developers keep working trees with modifications to system and commit changes back periodically

Checking Out & Updating

Basic process of getting sources manually:

- AnonCVS
  - Set CVSROOT or pass to each command w/ -d:
    `CVSROOT=:pserver:anoncvs@anoncvs.FreeBSD.org:/home/ncvs; export CVSROOT`
  - Login: `cvs login`
  - Check-out/update source: `cvs co -rTAG <pkg>`
  - Logout, etc. `cvs logout`
- CVSup
  - Install CVSup from port/package if needed
  - Copy example from /usr/share/examples/cvsup
  - Edit to use local CVSup mirror and chhose collections
  - Run cvsup to sync sources: `cvsup -gL2 <supfl>`
- See handbook for further info..

Source Selection

- Releases; Tracking -current & the HEAD tag
- **Selection**: Refers to choosing set of files from source tree to work on
- Selection parameters can include: release or branch tag, revisions and dates, source collections
- Using repository: check-out set of source files by providing revision or tag spec.
- Packages: corresponding files from mirrors
- Currently selecting, updating sources is manual process: but can be automated

Automation: Bringing it all together

- Automate system source management and source update tasks
- Provide a uniform interface to the user:
  - command line interface (CLI) w/ argument parsing;
  - console prompt-based (text menu);
  - curses menus (w/ GNU dialog);
  - further user interfaces possible (web, GUI, etc.) -- but deemed unnecessary for this tool
- Established common routines for repetitive tasks
- Uniform use of multiple methods

Why Automate?

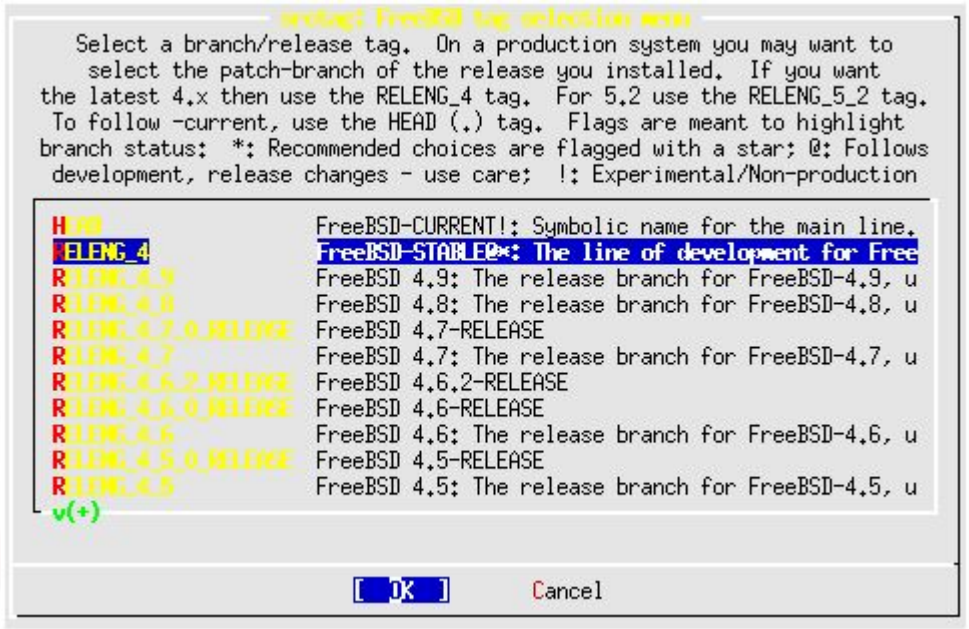
- Consistency: Maintain a uniform interface
- Time Saving: Automate routine procedures
- Simplify update process (focus on other tasks)
- Work with multiple branches: select active branch, update using single command
- Multi-source updates w/o supfile editing, CVS complexities. Simplified logging
- "Any Admin Should Know How to Update": Meant to supplement existing tools, approaches including using CVS directly
- Balance: convenience and power

Source Tag Selection

```
# srctag

[ FreeBSD src tree selection ]

Enter base directory [/usr]:
* Building tag menu from CVS ('skip' to bypass)...
Enter a CVS repository to use ('menu' for a menu) [:pserver:anoncvs:anoncvs@anoncvs.freebsd.org:/home
+/ncvs]:
@ Setting CVSROOT to :pserver:anoncvs:anoncvs@anoncvs.freebsd.org:/home/ncvs
* Parsing tag data
* Extracting tags from CVS
Enter FreeBSD CVS tag to make active ('menu' for a menu) [RELENG_4_7]: menu
```



- Srctag prompts the user for tag, source collections and update method

Automating Source Selection

- Take care of the nitty-gritty details in managing source-directories
- Performs sanity checks
- Uniformly names source directories; makes symbolic links where appropriate:

```
$ ls -CFd /usr/src*
/usr/src/ /usr/src-5_1/ /usr/src-5_2@
/usr/src-4_9/ /usr/src-version
$ ls -ld /usr/src-5_2
lrwxr-xr-x  1 root  wheel  7 Apr 27 06:57  /usr/src-5_2 -> src
```

- Above /usr/src is "active" selected source tree
- Access by both versioned (/usr/src-5_2) name and traditional /usr/src source tree

The cvsupit Port

- Released by Jordan K. Hubbard (jkh) Jan 1998; Light modifications through 2002 and marked broken then deleted 2003
- Simple shell script which is invoked from port Makefile install target (i.e. `make install`)
- What it did:
  - The cvsupit port provided text dialog interface to cvsup
  - Provided menued interface to create supfiles
  - Limitations / Problems
- An extended approach to updates..

Design Goals

- Set the goal of streamlining source management under FreeBSD
- Usage objectives:
  - Ease source updates and selection
  - Ease development, patching
- Manage multiple working branches checked-out at the same time
- Allow specification of "active" working branch (selection) and keep branch info
- Established common routines for updating
- Provide a uniform interface for updating

Multiple Releases

- When there is more than one working source tree on a system it's important to use useful identifiers
- To help track multiple-releases srctag keeps file called src-version holding info about active tree and other working directories:

```
/usr/src-current FreeBSD-CURRENT [HEAD]: Symbolic name for the main line.
(Last Update: 20040427 06:38:00-0400 by root from cvs:///home/ncvs)
(Selected: 20040427 06:38:00-0500 by root)
/usr/src-5_0_0 FreeBSD 5.0-RELEASE [RELENG_5_0_0_RELEASE] (Last Update:
20030327 19:10:00-0500 by root from cvs:///home/ncvs)
/usr/src-4_9 FreeBSD 4.9 [RELENG_4_9]: The release branch for FreeBSD-4.9,
used only for security advisories and other seriously critical fixes.
(Last Update: 20040403 07:06:54-0500 by toor from cvs:///home/ncvs)
/usr/src-5_2 FreeBSD 5.2 [RELENG_5_2]: The release branch for FreeBSD-5.2,
used only for security advisories and other seriously critical fixes.(Last
Update: 2004021 06:11:58-0400 by root from cvsup://cvsup3.freebsd.org)
```

- The active tree is marked "Selected"

Supported Methods

- srctag supports multiple update methods:
  - cvs: CVS repositories
  - cvsup: CVSup protocol
  - rsync: (not yet implemented)
  - none: don't update, just select
- Provide single interface that can use multiple alternatives: simple to add more
- Automatically mangle tag entries and package names:
  - CVS/CVSup modules: src/ vs. src-all
  - CVS/CVSup targets: HEAD vs. .

Extracting Tags from CVS

- Tag entries can exist in script, but might not be kept up to date with new releases
- Wouldn't it be nice if the script automatically got the most recent tags from CVS?
- CVS doesn't yet provide an easy (standard) method for retrieving all available tags from a repository (so used workaround)
- Alternative: keep a script-parseable file with up-to-date tag entries in CVS or releng page
- Extract the tags and use them to form entries for descriptives menu

The dirty trick.. and the sed script that does it:

```
# Find and form tag entries from CVS in sed (a hack):
while read -r line; do eval add_case $line; done <<-  _TAG_DATA_CVS
`$CVS log src/COPYRIGHT | $SED -nE \
 "/^symbolic names/ bah
   #sed...
     :ah
       :ack
         n
         /^  ([A-Za-z0-9_]+): *[0-9\.]*$/ {
             # Step 1: Strip down to tag
             s/^     ([A-Za-z0-9_]+): *[0-9\.]*$/\1/
             # Skip tags aleady seen (previously added)
             /^($TAG_SEEN)$/ back

             # Step 2: Form an appropriate tag entry
             s/^RELENG_([1-9]+)_([0-9]+)$/'RELENG_\1_\2|\1_\2|\1.\2'
                     &               \1_\2          \1.\2           rel
     'The release branch for FreeBSD-\1.\2, used only for security advisories
and other seriously critical fixes.'/
             # [.. snip: more patterns go here ..]

             # Step 3: Print out tag entry
             p
          }
          # If no tag entry could be formed, assume end of symbols in log:
        !t
      back
   " || exit $EX_FAILURE`
_TAG_DATA_CVS
```

Mirroring the FreeBSD sources

- srctag automatically generates cvsupfiles and hooks into `make update` target, but should allow full mirroring
- Having local mirror is handy for quick access to FreeBSD source code
- Currently source mirroring achieved by:
  - using cvsup-mirror port to create cron entry and update script
  - preparing a supfile manually, invoking cvsup periodically -- E.g.: `cvsup -g -L2 stable-supfile`
- srctag should provide equiv. function

Maintaining a Local Repository

- A local repository provides a place to check-in site specific changes to system files and the FreeBSD code
- It can provide for centralized storage of configuration files
- CVS has had security issues in past, but can run chroot jailed. Important to secure local repository
- There are good tutorials on setting up a local repository available online

Patching the Sources

- Normally, patches and bugfixes can be submitted directly to FreeBSD project
- In some cases: it may be desirable to apply custom patches to standard FreeBSD source or build sections of system from a custom branch
- Composite working source tree with any number of additional patches (in diff(1) format); add supplementary code from multiple sources
- Will also aid with check-in and merge of modifications to working sources

Patching (2)

- Auto-create diffs based on changes to working source tree
- UnionFS experiment
- Not all patching operations are possible/desirable to automate..

Buildmasters

- Buildmasters are responsible for building FreeBSD which can then be used by multiple machines
- Ideally srctag will provide the ability to automate the build process from selection on-ward and could be used for local site buildmasters
- What could be automated?
  - Set up object trees ready for export, manage multiple object trees uniformly
  - Prepare targets, help install desired version from buildmaster; logging whole process

Automatic Updating from Sources

- Is it a smart idea?
- What about /usr/src/UPDATING ?
- Steps:
  - backups
  - pre-buildworld mergemaster
  - buildworld, buildkernel
  - installkernel, installworld
  - Mergemaster
- srctag wasn't designed for this specific purpose, but would be next logical step; currently meant to manage sources only

Why SH?

- System tools should be made to operate with-out dependency where-ever possible
- The shell is a common ground, albeit functionally minimal
- A Worthy Challenge or Evil Obfuscation Tricks?
  - It's clear large shell scripts have disadvantages to modern scripting languages
  - However: It's surprising how much is possible inside the Bourne shell
  - Unix shell is highly versatile given you know a few idioms & tricks

Minimal SH & Standard Tools

- Standard Unix Shell Scripting Tools include: sh, cat, awk, sed, grep
- Availability: Bourne Shell readily available, even with minimal install (Lowest Common Denominator to reduce dependencies)
- Compatibility: Should run uniformly under FreeBSD installs including older releases
- Authors 1st choice: Perl, while increasingly standard: aim was to support installations where Perl isn't included by default
- Scripting language wars: Ruby vs. Perl vs. Python, etc. And the winner is...

Lessons on Complexity

- Anything is possible in SH...
- But is it practical?
- Line-oriented text processing in SH
- Script has served to demonstrate effectively maintainability issues with large scripts and should be rewritten: debugging hassles, etc.
- Reducing script complexity
- Modularity: Divide and Conquer
- Generality: Factor out common code
- Persistence in the Bourne Shell
- File edit hooks; simulating simple
- regexen and other tricks

SHARnet: Need for a Shell Archive Network?

- The need for a modular, distributed approach to shell programming
- Small is beautiful: specialized vs. modular
- Reusable shell components
- Shell script libraries (and script archives)
- shar(1) shell archives
- Work In Progress – sharnet.org
- Shell packages, the sharnet namespace
- Similarity to other archives such as CPAN
- Development of the sharnet code
- Automated package retrieval
- Potential Ports integration

Further Ideas for Updates

- Automation of source tree updates far less ambitious / controversial than automation of the build process itself
- Always want users/admin aware of any changes to system
- Fully automated updates may not make sense in all situations
- In buildmaster scenario, automation of building from start to finish is possible
- After a new release hits the repository: buildmaster can check-out latest sources and attempt an unattended build, make available if sucessful

Online Resources

- http://afields.ca/bsdcan/
- http://afields.ca/proj/srctag/

- manual pages: sh(1), bash(1), sed(1)
- FreeBSD Handbook Chapter 21: The Cutting Edge (file:/usr/share/doc/handbook/cutting-edge.html)

- http://www.shelldorado.com/links/index.html#archives

Request For Comments

- Similar Projects or Experiences?
- Critical Points: Reasons not to adopt these approaches? What could be done better?
- About Tag Specs:
  - Could releng release tag entries with full meta-data? (space delimited/marked up?)
  - More reliable than constructing on-the-fly from CVS?
  - Should there be high-lighted branches?
- In search of a better name than: `srctag`
- Questions?