Sofware Release Tools Tutorial

Lassi A. Tuura
1997–10–09

This Presentation

Introduction

Motivation

SRT: What It Is

SRT: What It Isn't

Basic Concepts

The `srt` Tool

Setting Yourself Up

Using Software from a Prebuilt Release

Building a Custom Version of a Package

Adding a Package

Release Cycle

Makefiles and Configuration

Example Session

Introduction

Coverage: *Software Release Tools*, *GNU* `make` and *Concurrent Versions System* for the parts most useful for the developers

- `make` and CVS are only covered superficially; please consult their documentation and the IT course materials for further details

  `http://wwwcn.cern.ch/dci/asis/products/GNU.LANG/make/make_toc.html`

  `http://wwwcn.cern.ch/dci/asis/products/GNU.LANG/cvs/cvs_toc.html`

- SRT Manual:

  `http://wwwcn.cern.ch/~lat/exports/srt/manual.ps`

This Presentation:

`http://wwwcn.cern.ch/~lat/slides/97-41/`

Motivation

Our software is made of *packages:*

- Developed quite *independently of each other*
- Thus, package *responsibilities are distributed*
- Packages *use* and *depend on other packages*

We want a common, consistent build environment *for all our own packages, for all operating systems and compilers*

Official releases should be *visible in one central place*

Same tools for everybody—both the developers and the software librarian

Guarantee maximal *internal consistency*

SRT: What It Is

Software Release Tools

- Originally developed in BaBar
- Substantial local modifications

Helping in:

- Defining, building and making available *software releases*
- Common *tasks*
- Simplifying *makefiles:* easier to write and to understand

Works *together with CVS*—does not hide or replace it

Built on top of GNU tools (`autoconf`, `make`)

SRT: What It Isn't

Not a *source code repository*—uses CVS

Little support for random messing around—*the tools assume responsible software development*

Not an all-encompassing *integrated development environment*—just make your favourite tool to use GNU make to compile and CVS for version management

Basic Concepts

Working Model

Packages

Versions and Tags

Releases

Working Model

Software Librarian *releases software*, e.g. once every two months or when predetermined level of functionality is achieved

Package authors inform librarian of the *new versions* their packages have been stamped with

A consistent, successfully built and tested *release is frozen*

A *frozen release can be used as a basis* for further development, both for normal package development, or for other development forking off from some earlier version

- One can check out only the sources for a single package; the rest will be picked up from the frozen base release

Packages

Package ≡ Unit of Software

Package ≡ Subdirectory

Package ≡ CVS module

Packages may contain other packages

Packages are typically developed quite independently from each other, even when they are contained in other packages

For SRT, a package is defined by a subdirectory that has:

- `PACKAGE` file declaring the properties of the package, and
- A makefile, and possibly also
- A `configure` script

Versions and Tags

Packages are versioned using CVS

When package author thinks a package is consistent and ready for use by others, he/she can *stamp* it with a *tag*

SRT uses tags of the form *name-nn-mm-pp*

- *name* is package name (only the last part if the package is a nested one)
- *nn* is major version number
- *mm* is minor version number
- *pp* is patch level

For example, a tag could be `HBookTuple-01-23-04`

Other tags may also be used, but they will not be recognised or accepted by the tools

Releases

Release is just an aggregate package

Typically releases are made available in the repository by the name of their version number, not by their package name:

```
/afs/cern.ch/atlas/software/dist:
    00-00-00
    00-00-01
    current -> 00-00-00
    latest -> 00-00-01
```

The `srt` Tool

Command syntax similar to `cvs`:

`srt` *[general options]* `tool` *[tool options]*

Tools:

- `arch`—Show the canonical architecture of the host system
- `build`—Build a package concurrently on several systems
- `check`—Check the consistency of a package hierarchy
- `create`—Create a new public release from scratch
- `freeze`—Freeze a public release
- `install`—Install Software Release Tools on a new system
- `new`—Create a new release
- `reversion`—Choose a different package version
- `setup`—Print user shell environment setup commands

Setting Yourself Up

With Bourne shell derivatives:

```
% PATH=$PATH:/afs/cern.ch/atlas/software/bin
% . ‘srt setup‘
```

With C-shell derivatives

```
% setenv PATH ${PATH}:/afs/cern.ch/atlas/software/bin
% source ‘srt setup‘
```

The following variables are set:

- `$SRT_HOME`—the location of the release tools
- `$SRT_ARCH`—host system’s canonical architecture name
- `$SRT_CONF`—compiler selection
- `$SRT_TARGET`—`${SRT_ARCH}/${SRT_CONF}`

To change compiler, reset `$SRT_CONF` and rerun `setup`

Using Software from a Prebuilt Release

To run application `arve` from release `00-00-00`

```
% release=/afs/cern.ch/atlas/software/dist/00-00-00
% $release/installed/$SRT_TARGET/bin/arve
```

To use libraries from a release, add

```
-L$release/installed/$SRT_TARGET/lib
```

to linker options

Building a Custom Version of a Package

To build a custom version of graphics/AVRML package, picking up the rest of packages from the central repository:

```
% cvs checkout -d AVRML offline/graphics/AVRML
% cd AVRML
% srt base 00-00-00/graphics/AVRML
% emacs src/SFColor.cxx
```

If you build on one platform:

```
% ./configure
% gmake
```

If you want to build on multiple platforms, do on each:

```
% mkdir -p .srt/$SRT_TARGET; cd .srt/$SRT_TARGET
% ../../../configure
% gmake
```

Adding a Package

If you want to add a top level package, consult the Software Librarian first to have permissions set for you

To import existing sources:

```
% cd source-directory
% cvs import -m 'Importing package name.' \
        offline/name ATLAS name-00-00-00
% cd ..
% cvs checkout name
% gdiff -rq source-directory name
```

For example:

```
% cd src
% cvs import -m 'Importing package graphics/Aravis.' \
        offline/graphics/Aravis ATLAS Aravis-00-00-00
N offline/graphics/Aravis/ChangeLog
N offline/graphics/Aravis/PACKAGE
```

```
N offline/graphics/Aravis/GNUmakefile.in
N offline/graphics/Aravis/configure
N offline/graphics/Aravis/configure.in

No conflicts created by this import

% cd ..
% cvs checkout -d Aravis offline/graphics/Aravis
cvs checkout: Updating Aravis
U Aravis/ChangeLog
U Aravis/GNUmakefile.in
U Aravis/PACKAGE
U Aravis/configure
U Aravis/configure.in
```

To do the same, but creating the package from scratch:

```
% cvs checkout -l -d graphics offline/graphics
cvs checkout: Updating graphics
U graphics/ChangeLog
U graphics/GNUmakefile.in
U graphics/PACKAGE
U graphics/configure
U graphics/configure.in
% cd graphics; mkdir Aravis; cvs add Aravis
Directory /tmp/lat/cvs/offline/graphics/Aravis added to the repository
```

```
% cd Aravis
% echo '# empty' > PACKAGE; emacs GNUmakefile
% cvs add PACKAGE GNUmakefile
cvs add: scheduling file ‘PACKAGE’ for addition
cvs add: scheduling file ‘GNUmakefile’ for addition
cvs add: use ‘cvs commit’ to add these files permanently
% cvs commit -m 'Skeleton version of graphics/Aravis.'
cvs commit: Examining .
cvs commit: Committing .
RCS file: /tmp/lat/cvs/offline/graphics/Aravis/GNUmakefile,v
done
Checking in GNUmakefile;
/tmp/lat/cvs/offline/graphics/Aravis/GNUmakefile,v  <--  GNUmakefile
initial revision: 1.1
done
RCS file: /tmp/lat/cvs/offline/graphics/Aravis/PACKAGE,v
done
Checking in PACKAGE;
/tmp/lat/cvs/offline/graphics/Aravis/PACKAGE,v  <--  PACKAGE
initial revision: 1.1
done
% cvs tag Aravis-00-00-00
cvs tag: Tagging .
T GNUmakefile
T PACKAGE
```

Release Cycle

This is what happens between two releases:

- You keep on developing the code either in the *head* or in a *branch*

  CVS Commands: `checkout`, `update`, `diff`

- When you consider the sources stable and good for public consumption—and you have checked that it compiles and runs reasonably sanely on at least one platform—you *stamp* the sources *with a tag*

  CVS Commands: `tag`, `rtag`

  Note that you may tag zero, one, or more new versions during any one release cycle.

- Software Librarian contacts you to tell that she is about to start the release build cycle; you tell about new versions and anything interesting related to them
- If the release builds smoothly, you'll hear no more from the librarian

- If there are errors, she'll forward them to you and ask you to fix them (possibly with a fix if it was easy to figure out)
  - You have to go and *fix the errors.* If the stamped version the librarian is using is older and you have already done significant development, you'll have to create a *CVS branch* (using `cvs tag` with the `-b` option) and fix the problems there—otherwise you can just keep developing in the main line
  - Having solved the problems, you *commit your changes* to CVS and tell the librarian to update her version of the sources, and she tries to rebuild and test the release again; note that other packages may need to be upgraded too due to version dependency changes, so there might be a flurry of package changes and lots of communication back and forth
  - If the problems are all solved, the librarian gets back to you one more time, asking you to *stamp the corrected version;* you do so, and if this was a branch, you schedule to *merge all the changes back to your main development line* at some convenient future time (`cvs update` or `checkout` with the `-j` option)

  CVS Commands: `commit`, `update`, `diff`, `tag`, `rtag`

Makefiles and Configuration

Configuration is *adapting the package* to the compiler, host operating system, optional features, installation location etc.

If the configuration automatically senses the properties when possible, *porting the package* to other platforms becomes easier

Automatic configuration can be done with `autoconf`

- you write a script (`configure.in`) using the predefined macros mixed with shell scripting language
- autoconf produces from that a fully expanded shell script (`configure`)

Good makefiles are difficult to write

- Supporting compilers on different platforms is tedious
- Several kinds of targets are useful for people with different needs
- Once you have written a good makefile, 90% of it stays the same from one package to another

SRT *factors the common makefile parts out*

- Handles all the *different compiler behaviours* by recording host operating system and compiler quirks into *makefile fragments*
- Handles *file name extensions* for object files, libraries and executables
- Handles all the *common targets* such as `install`, `uninstall`, `check`, `clean`, ...
- Provides convenient *macros to declare various types of targets* so that they will be installed in the right place
- Provides convenient *macros to add more compiler options to specific files*

Typical configure.in

```
dnl $Id: configure.in,v 1.1 1997/10/09 07:14:14 lat Exp $ -*- m4 -*-
dnl $Name:  $
dnl
dnl COPYRIGHT (C) CERN, 1997.
dnl ###########################################################
dnl Introduce revision number to the generated script.
AC_PREREQ(2.10)
AC_REVISION([$Id: configure.in,v 1.1 1997/10/09 07:14:14 lat Exp $])

dnl ###########################################################
dnl Initialisation
dnl
AC_INIT(test/src/LocalHelix.cxx)
SRT_INIT

dnl ###########################################################
dnl Configure locally for SRT compliance, and then configure the
dnl nested packages.
SRT_CONFIG_LOCAL
SRT_CONFIG_PACKAGES

dnl ###########################################################
dnl Generate GNUmakefile, make state files, etc.
SRT_OUTPUT(GNUmakefile test/GNUmakefile)
```

A More Complex configure.in

```
dnl $Id$ -*- m4 -*-
dnl $Name$
dnl
dnl COPYRIGHT (C) CERN, 1997.
dnl
AC_PREREQ(2.10)
AC_REVISION([$Id$])

dnl ############################################################
dnl Initialisation
dnl
AC_INIT(ghits/gcderr.F)
SRT_INIT

dnl ############################################################
dnl Determine canonical system name and select configuration data
dnl for it
dnl
AC_CANONICAL_SYSTEM
```

```
AC_MSG_CHECKING(which host fragment to use)
host=${host_cpu}-${host_os}
old=
until test "$host" = "$old" || test -f $srcdir/config/${host}.mk; do
  old="$host"
changequote(,)
  host=‘echo $host | sed 's%\.[0-9]*$%%'‘
changequote([,])
done

if test -f $srcdir/config/${host}.mk; then
  AC_MSG_RESULT(${host}.mk)
else
  AC_MSG_RESULT(none)
  AC_MSG_ERROR(unsupported target architecture)
fi

HOST_OPTIONS=$srcdir/config/${host}.mk
AC_SUBST_FILE(HOST_OPTIONS)

dnl ########################################################
dnl Configure locally for SRT compliance.
dnl
SRT_CONFIG_LOCAL
```

```
dnl ###########################################################
dnl Generate GNUmakefile, make state files, etc.
dnl
makefiles="GNUmakefile
           block/GNUmakefile:libdummy.mk.in
           cgpack/GNUmakefile:libdummy.mk.in
           ...
           guser/GNUmakefile:libdummy.mk.in
           peanut/GNUmakefile:libdummy.mk.in"
SRT_OUTPUT($makefiles)
```

Typical GNUmakefile.in

```
# $Id: GNUmakefile.in,v 1.2 1997/10/08 12:07:17 lat Exp $
# $Name:  $
#
# COPYRIGHT (C) CERN 1997.
#############################################################
TARGET_LIBS              = libCLHEP.a

libCLHEP.a_USES          = c++
libCLHEP.a_CXXCPPFLAGS   = -I$(top_srcdir)
libCLHEP.a_SRC           = AIteratorBase.cxx \
                           LorentzRotation.cxx \
                           RandomEngine.cxx \
                           ThreeVector.cxx \
                           AListBase.cxx \
                           LorentzVector.cxx \
                           Rotation.cxx \
                           JamesRandom.cxx \
                           Random.cxx
all:: ;
distclean-local:; -rm -f GNUmakefile
#############################################################
include $(SRT_HOME)/standard.mk
```

Example Session

```
% cd /tmp/lat
% cvs checkout -d arve offline/arve
cvs checkout: Updating arve
U arve/GNUmakefile.in
U arve/MAKEINFO
U arve/MAKEINFO.base
U arve/Makefile
U arve/PACKAGE
U arve/arve.dsw
U arve/configure
U arve/configure.in
cvs checkout: Updating arve/CLHEP
U arve/CLHEP/AIterator.h
U arve/CLHEP/AIterator.icc
U arve/CLHEP/AIteratorBase.cxx
U arve/CLHEP/AIteratorBase.h
U arve/CLHEP/AIteratorBase.icc
U arve/CLHEP/AList.h
U arve/CLHEP/AList.icc
U arve/CLHEP/AListBase.cxx
U arve/CLHEP/AListBase.h
```

```
U arve/CLHEP/AListBase.icc
U arve/CLHEP/CLHEP.DSP
U arve/CLHEP/CLHEP.h
U arve/CLHEP/GNUmakefile.in
U arve/CLHEP/JamesRandom.cxx
U arve/CLHEP/JamesRandom.h
U arve/CLHEP/LorentzRotation.cxx
U arve/CLHEP/LorentzRotation.h
U arve/CLHEP/LorentzRotation.icc
U arve/CLHEP/LorentzVector.cxx
U arve/CLHEP/LorentzVector.h
U arve/CLHEP/LorentzVector.icc
...
U arve/wingui/wingui.dsp
U arve/wingui/wingui.rc
% cd arve
% srt base 00-00-00/arve
```

```
% ./configure
creating cache ./config.cache
configure: warning: no SRT config.site, creating one here
checking for gmake... gmake
checking which base release to use... /afs/cern.ch/atlas/software/
dist/00-00-00
checking where the release root it... .
checking which base package to use... /afs/cern.ch/atlas/software/
dist/00-00-00/arve
checking for PACKAGE... exists
checking transitive closure of used packages... (none)
updating cache ./config.cache
creating ./config.status
creating GNUmakefile
creating .makestate
creating CLHEP/GNUmakefile
creating CLHEP/.makestate
creating arve/GNUmakefile
creating arve/.makestate
creating batch/GNUmakefile
creating batch/.makestate
...
creating test_display/GNUmakefile
creating test_display/.makestate
creating wingui/GNUmakefile
creating wingui/.makestate
```

```
% make |& sed "s%$SRT_TARGET%\$SRT_TARGET%g"
Creating rules for subdirectory wingui
Creating rules for subdirectory test
Creating rules for subdirectory ifile
Creating rules for subdirectory gismo
Creating rules for subdirectory gheisha
Creating rules for subdirectory egs
Creating rules for subdirectory batch
Creating rules for subdirectory test_display
Creating rules for subdirectory motifgui
Creating rules for subdirectory graphics
Creating rules for subdirectory geometry
Creating rules for subdirectory control
Creating rules for subdirectory arve
Creating rules for subdirectory CLHEP
make[1]: Entering directory ‘/tmp/lat/arve/CLHEP’
Creating rules for library libCLHEP.a
make[1]: Leaving directory ‘/tmp/lat/arve/CLHEP’
Memorizing source file list for libCLHEP.a
make[1]: Entering directory ‘/tmp/lat/arve/CLHEP’
make[1]: Leaving directory ‘/tmp/lat/arve/CLHEP’
make[1]: Entering directory ‘/tmp/lat/arve/CLHEP’
g++ -I../. -I.././installed/$SRT_TARGET/include -I.././installed/
include -I/afs/cern.ch/atlas/software/dist/00-00-00/installed/
$SRT_TARGET/include -I/afs/cern.ch/atlas/software/dist/00-00-00/
installed/include -ansi -pedantic -Wall -Wwrite-strings -Wpointer-
```

```
arith -Wnested-externs -W -Woverloaded-virtual -Wbad-function-
cast -g -MD -c AIteratorBase.cxx -o AIteratorBase.o

g++ -I../. -I.././installed/$SRT_TARGET/include -I.././installed/
include -I/afs/cern.ch/atlas/software/dist/00-00-00/installed/
$SRT_TARGET/include -I/afs/cern.ch/atlas/software/dist/00-00-00/
installed/include -ansi -pedantic -Wall -Wwrite-strings -Wpointer-
arith -Wnested-externs -W -Woverloaded-virtual -Wbad-function-
cast -g -MD -c LorentzRotation.cxx -o LorentzRotation.o

g++ -I../. -I.././installed/$SRT_TARGET/include -I.././installed/
include -I/afs/cern.ch/atlas/software/dist/00-00-00/installed/
$SRT_TARGET/include -I/afs/cern.ch/atlas/software/dist/00-00-00/
installed/include -ansi -pedantic -Wall -Wwrite-strings -Wpointer-
arith -Wnested-externs -W -Woverloaded-virtual -Wbad-function-
cast -g -MD -c RandomEngine.cxx -o RandomEngine.o

...
```