SCM: A look at SCons and GNU Arch

Ben Leslie

benjl@cse.unsw.edu.au

WHAT IS SCM?

CM is the discpline of controlling the evolution of complex systems; SCM is its specialization for computer programs and associated documents

WHAT IS SCM?

CM is the discpline of controlling the evolution of complex systems; SCM is its specialization for computer programs and associated documents

No really, what is it ?

WHAT IS SCM?

CM is the discpline of controlling the evolution of complex systems; SCM is its specialization for computer programs and associated documents

No really, what is it ?

- Versioning

WHAT IS SCM?

CM is the discpline of controlling the evolution of complex systems; SCM is its specialization for computer programs and associated documents

No really, what is it ?

- Versioning
- Building

BUILD SYSTEM – PURPOSE

Automate construction process

- Build rules – how to change .c into .o
- Capture all build rules

Rebuild exactly what is needed

BUILD SYSTEM – PURPOSE

Automate construction process

- Build rules – how to change .c into .o
- Capture all build rules

Rebuild exactly what is needed

- Build everything required – correctness.

BUILD SYSTEM – PURPOSE

Automate construction process

- Build rules – how to change .c into .o
- Capture all build rules

Rebuild exactly what is needed

- Build everything required – correctness.
- Don't build anymore then needed – performance.

BUILD SYSTEM – PURPOSE

Automate construction process

- Build rules – how to change .c into .o
- Capture all build rules

Rebuild exactly what is needed

- Build everything required – correctness.
- Don't build anymore then needed – performance.
- Determine dependencies – robustness.

BUILD SYSTEM – PURPOSE

Automate construction process

- Build rules – how to change .c into .o
- Capture all build rules

Rebuild exactly what is needed

- Build everything required – correctness.
- Don't build anymore then needed – performance.
- Determine dependencies – robustness.
- Easily extensible – flexibility.

BUILD SYSTEM – PURPOSE

Automate construction process

- Build rules – how to change .c into .o
- Capture all build rules

Rebuild exactly what is needed

- Build everything required – correctness.
- Don't build anymore then needed – performance.
- Determine dependencies – robustness.
- Easily extensible – flexibility.
- Easily to use – convenience.

BUILD SYSTEM – TOOLS

The incumbent

make

BUILD SYSTEM – TOOLS

The incumbent

make

> It's sort of a graffiti recorder
> — *Bill Joy*

BUILD SYSTEM – TOOLS

The incumbent

make + automake, autoconf, libtool, gcc -M

> It's sort of a graffiti recorder
> — *Bill Joy*

BUILD SYSTEM – TOOLS

The incumbent

make + automake, autoconf, libtool, gcc -M

> It's sort of a graffiti recorder
> — *Bill Joy*

Others around

Ant, bake, cook, bras, jam, odin, vesta

BUILD SYSTEM – TOOLS

The incumbent

make + automake, autoconf, libtool, gcc -M

> It's sort of a graffiti recorder
> — *Bill Joy*

Others around

Ant, bake, cook, bras, jam, odin, vesta

My fave

SCons

SCons

Features

- Written in python
- Build scripts essentially python EDSL
- Cross-platform
- Gets depencies correct
- Usage scales – easy to write simple and complex systems
- Friendly user group

SCons

Features

- Written in python
- Build scripts essentially python EDSL
- Cross-platform
- Gets depencies correct
- Usage scales – easy to write simple and complex systems
- Friendly user group
- id software uses it for Doom3 ;)

SCons - Hello World

Example SConstruct file:

```
Program('hello.c')
```

Example usage:

```
% scons
```

SCons - Dependency calculation

Implicit: Scons parses source files with a *Scanner* to determine dependencies.

- Scanner knows about generated files.
- Easy to write your own Scanner
- Command line arguments are also implicit dependencies. (E.g: Change cflags)

Explicit: Scons allows you to specify explicit dependencies.

```
Depends('foo.o', 'bar')
```

SCons - Change detection

MD5Sum: Calculate a files MD5Sum to determine if it has changed.

Timestamp: Optional to revert to timestamp method.

Built files: Default is to always rebuild. Optionally treat as other files. So if a .o is unchanged by rebuild nothing else is done.

SCons - Environments

Evironments encapsulate a set of build logic. E.g: which compiler, flags etc. All programs built in an environment inherit the environment's defaults.

Multiple environments can be specified enabling large project to be easily built.

```
opt = Environment(CCFLAGS = '-O2')
dbg = Environment(CCFLAGS = '-g')

opt.Program('foo', 'foo.c')
dbg.Program('bar', 'bar.c')
```

SCons - Misc

Clean is handled automatically by SCons. Just `scons -c`.
PATH is not preserved! Need to manually propagate.
Debugging is nice. It is just python so you can easily use `print`.
Command line args are nicely exposed so you can provide options to your build. (E.g: `-debug`). Hierarchical builds are supported. (Think make's include support, not make's recursive build support.

RANT: CHANGE DETECTION IN LINUX SUCKS!