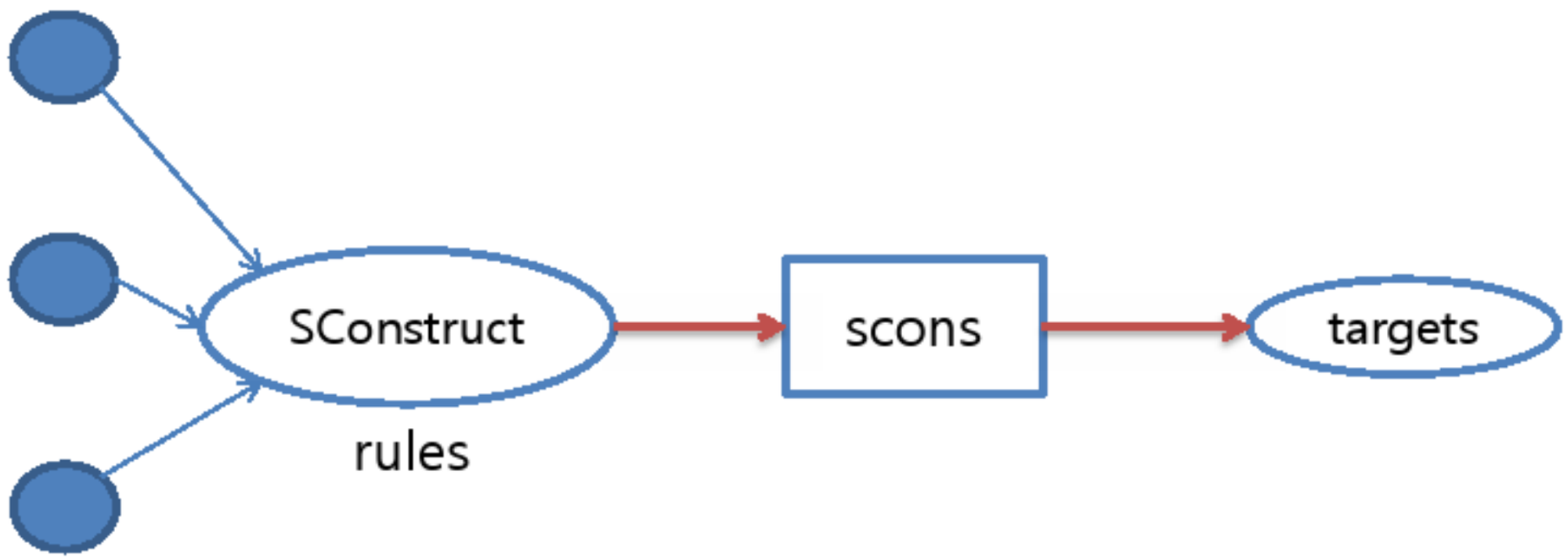
Scons

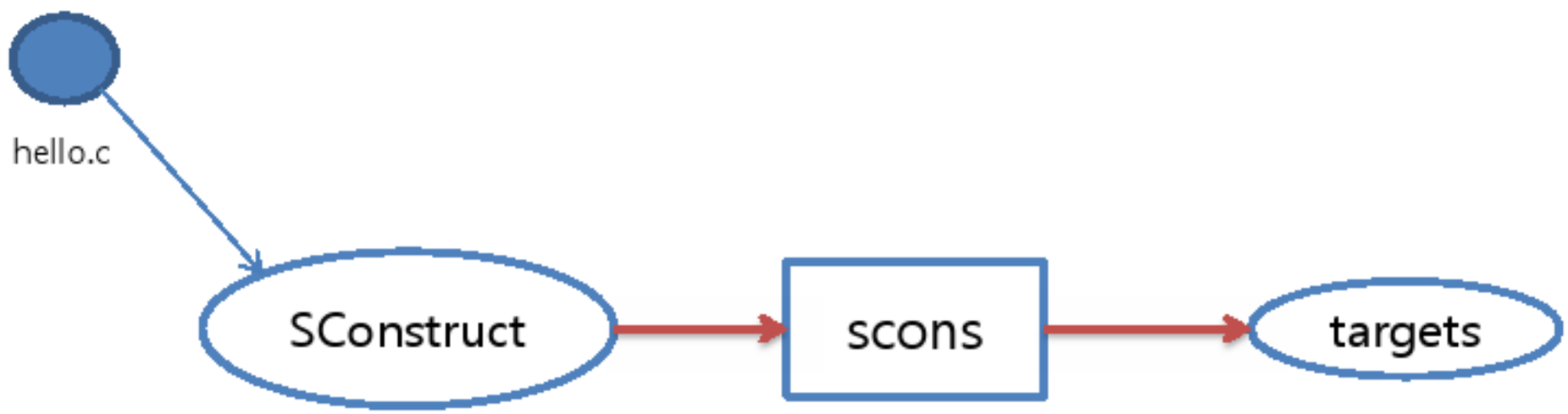
What is SCons

- Software construction tool
  - alternative to “make”
  - in Python
  - not compatible with “make”
- Why powerful?
  - “configuration files” are Python scripts
- Python versions
  - 2.4 <= versions < 3.0

Flow



Simple Builds

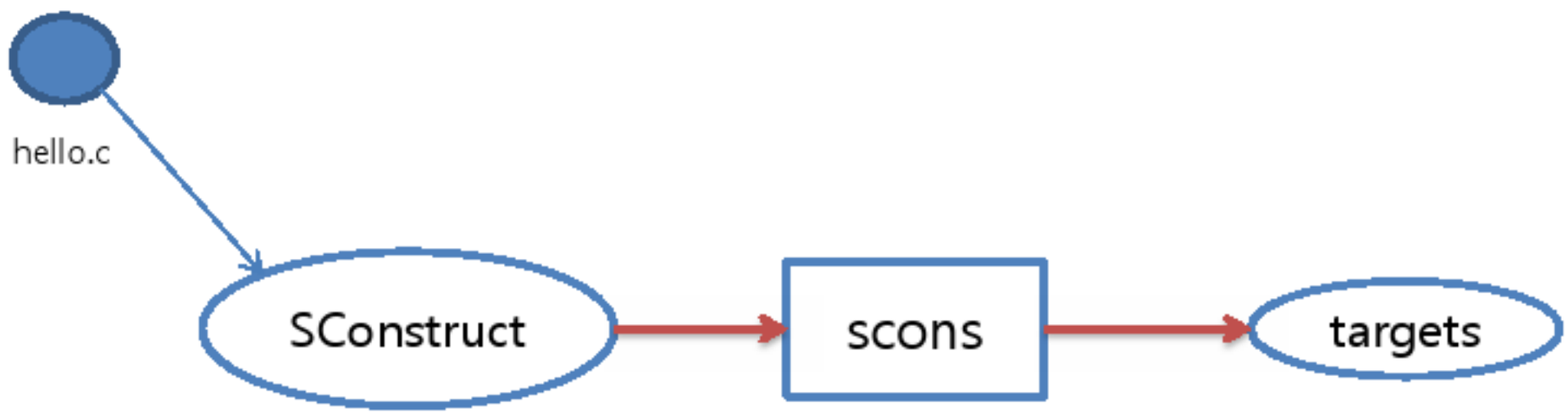


Program('hello.c')

```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
gcc -o hello.o -c hello.c
gcc -o hello hello.o
scons: done building targets.
```

```
hello
```

Building object files



Object('hello.c')

```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
gcc -o hello.o -c hello.c
scons: done building targets.
```

hello.o

Java Builds

Java('classes', 'src')

Destination directory
*.class

Source directory
*.java

Cleaning up

```
scons -c
```

Making Less Verbose

```
scons -Q
```

General form of Program()

```
Program('program', ['prog.c', 'file1.c', 'file2.c'])
```

Target name

sources (list)

General form of Program()

```
Program('program', Split('prog.c file1.c file2.c'))
```

Target name

sources (list)

General form of Program()

```
Program('program', 'prog.c file1.c file2.c'.split())
```

Target name

sources (list)

General form of Program()

```
Program('program', Glob('*.c'))
```

Target name

sources

General form of Program()

```
import glob
Program('program', glob.glob('*.c'))
```

Target name

sources

Multiple programs

```
Program('foo.c')
Program('bar', ['bar1.c', 'bar2.c'])
```

Sharing files

```
common = ['common1.c', 'common2.c']
foo_files = ['foo.c'] + common
bar_files = ['bar1.c', 'bar2.c'] + common

Program('foo', foo_files)
Program('bar', bar_files)
```

Building static libraries

```
Library('foo', ['f1.c', 'f2.o', 'f3.c', 'f4.o'])

StaticLibrary('foo', ['f1.c', 'f2.o', 'f3.c', 'f4.o'])
```

Building shared libraries

```
SharedLibrary('foo', ['f1.c', 'f2.o', 'f3.c', 'f4.o'])
```

Linking with Libraries

Existing library

Library('foo', ['f1.c', 'f2.c', 'f3.c'])

bar.a

Program('prog.c', LIBS=['foo', 'bar'], LIBPATH='.')

['.', '/usr/lib', '/usr/local/lib']
'/usr/lib:/usr/local/lib'

```
env = Environment(LIBPATH='/usr/local/lib')

env.Program('test', 'main.cpp')
```

```
g++ -o main.o -c main.cpp
g++ -o test main.o -L/usr/local/lib
```

Compile options

```
Object('hello.c', CCFLAGS='-DHELLO')
Object('goodbye.c', CCFLAGS='-DGOODBYE')
```

Compile options

```
Object('hello.c', CCFLAGS='-DHELLO -O2')
Object('goodbye.c', CCFLAGS='-DGOODBYE -O2')
```

Compile options

```
Object('hello.c', CCFLAGS=['-DHELLO', '-O2'])
Object('goodbye.c', CCFLAGS=['-DGOODBYE', '-O2'])
```

Node?

- Files
- Directories

```
hello_c = File('hello.c')
Program(hello_c)

classes = Dir('classes')
Java(classes, 'src')
```

List of nodes

- All builder methods return a list of Node objects

```
hello_list = Object('hello.c', CCFLAGS='-DHELLO')
```

list of nodes

List of nodes

```
hello_list = Object('hello.c', CCFLAGS='-DHELLO')
goodbye_list = Object('goodbye.c', CCFLAGS='-DGOODBYE')

Program(hello_list + goodbye_list)
```

Getting the Path From a Node or String

```
env=Environment(VAR="value")
n=File("foo.c")

print env.GetBuildPath([n, "sub/dir/$VAR"])
```

Implicit dependencies

Program('hello.c', CPPPATH='.')

cc -o hello.o -c -I. hello.c

소스에 포함된 header 파일이 수정되었는지
. 디렉토리 검사 후 필요하면 재 컴파일

```
env = Environment(CPPPATH='/usr/local/include')
env.Program('test', 'main.cpp')
```

외부 라이브러리를 사용한다면, 헤더 파일과 라이브러리 파일을 지정해주어야

Caching Implicit Dependencies

스캐너가 의존성을 검사한 내용을
캐쉬해두고 사용하겠다는 뜻

SetOption('implicit_cache', 1)

이 옵션을 자동 처리

```
% scons -Q --implicit-cache hello
        cc -o hello.o -c hello.c
        cc -o hello hello.o
        % scons -Q hello
        scons: `hello' is up to date.
```

Explicit dependencies

```
hello = Program('hello.c')
Depends(hello, 'other_file')
```

Or list

Ignoring dependencies

```
hello_obj=Object('hello.c')
hello = Program(hello_obj)

Ignore(hello_obj, 'hello.h')
```

AlwaysBuild Function

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

Environments

- External environments
  - os.environ (사전) 이용
- Construction environments
  - Environment()
- Execution environments
  - 외부 명령을 실행할 때 Scons가 설정하는 환경

Construction Environment

```
env = Environment()   # default

env = Environment(CC = 'gcc',   CCFLAGS = '-O2')
print "CC is:", env['CC']

env.Program('foo.c')
```

Expanding Values From a Construction Environment

```
env = Environment()   # default
print "CCCOM is:", env['CCCOM']
print "CCCOM is:", env.subs('$CCCOM')
```

```
CCCOM is: $CC $CCFLAGS $CPPFLAGS $_CPPDEFFLAGS $_CPPINCFLAGS -c -o $TARGET $SOURCES
CCCOM is: gcc -DFOO -c -o
```

Setting default environment

```
DefaultEnvironment(CC = '/usr/local/bin/gcc')

env = DefaultEnvironment()
env['CC'] = '/usr/local/bin/gcc'


env = DefaultEnvironment(tools = ['gcc', 'gnulink'],
                         CC = '/usr/local/bin/gcc')
```

Multiple constructive environment

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

opt.Program('foo', 'foo.c')

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

Making copies of constructive environment

```
env = Environment(CC = 'gcc')
opt = env.Clone(CCFLAGS = '-O2')
dbg = env.Clone(CCFLAGS = '-g')
```

Replacing/Appending values

```
env = Environment(CCFLAGS = '-DDEFINE1')
env.Replace(CCFLAGS = '-DDEFINE2')
env.Program('foo.c')

env = Environment(CCFLAGS = ['-DMY_VALUE'])
env.Append(CCFLAGS = ['-DLAST'])
env.Program('foo.c')
```

• • •

Installing files

```
env = Environment()
hello = env.Program('hello.c')
env.Install('/usr/bin', hello)
```

```
% scons -Q
cc -o hello.o -c hello.c
cc -o hello hello.o

% scons -Q /usr/bin
Install file: "hello" as "/usr/bin/hello"
```

Installing files

```
env = Environment()
hello = env.Program('hello.c')
env.Install('/usr/bin', hello)
env.Alias('install', '/usr/bin')
```

```
% scons -Q
cc -o hello.o -c hello.c
cc -o hello hello.o

% scons -Q install
Install file: "hello" as "/usr/bin/hello"
```

Installing files

```
env = Environment()
hello = env.Program('hello.c')
goodbye = env.Program('goodbye.c')
env.Install('/usr/bin', [hello, goodbye])
env.Alias('install', '/usr/bin')
```

```
% scons -Q install
        cc -c -o goodbye.o goodbye.c
        cc -o goodbye goodbye.o
        Install file: "goodbye" as "/usr/bin/goodbye"
        cc -c -o hello.o hello.c
        cc -o hello hello.o
        Install file: "hello" as "/usr/bin/hello"
```

Installing a file under a different name

```
env = Environment()
hello = env.Program('hello.c')
env.InstallAs('/usr/bin/hello-new', hello)
env.Alias('install', '/usr/bin')
```

```
% scons -Q install
      cc -o hello.o -c hello.c
      cc -o hello hello.o
      Install file: "hello" as "/usr/bin/hello-new"
```

Installing a file under a different name

```
env = Environment()
hello = env.Program('hello.c')
goodbye = env.Program('goodbye.c')
env.InstallAs(['/usr/bin/hello-new', '/usr/bin/goodbye-new'],
              [hello, goodbye])
env.Alias('install', '/usr/bin')
```

```
% scons -Q install
      cc -o goodbye.o -c goodbye.c
      cc -o goodbye goodbye.o
      Install file: "goodbye" as "/usr/bin/goodbye-new"
      cc -o hello.o -c hello.c
      cc -o hello hello.o
      Install file: "hello" as "/usr/bin/hello-new"
```

Alias

```
env = Environment()

p = env.Program('foo.c')
l = env.Library('bar.c')

env.Install('/usr/bin', p)
env.Install('/usr/lib', l)

ib = env.Alias('install-bin', '/usr/bin')
il = env.Alias('install-lib', '/usr/lib')

env.Alias('install', [ib, il])
```

Alias

```
% scons -Q install-bin
cc -o foo.o -c foo.c
cc -o foo foo.o
Install file: "foo" as "/usr/bin/foo"

% scons -Q install-lib
cc -o bar.o -c bar.c
ar rc libbar.a bar.o
ranlib libbar.a
Install file: "libbar.a" as "/usr/lib/libbar.a"
```

Alias

```
% scons -Q -c /
Removed foo.o
Removed foo
Removed /usr/bin/foo
Removed bar.o
Removed libbar.a
Removed /usr/lib/libbar.a

% scons -Q install
cc -o foo.o -c foo.c
cc -o foo foo.o
Install file: "foo" as "/usr/bin/foo"
cc -o bar.o -c bar.c
ar rc libbar.a bar.o
ranlib libbar.a
Install file: "libbar.a" as "/usr/lib/libbar.a"
```

SConscript

- Sconstruct
  - Root of the project
- Sconscript
  - Anywhere else
  - Included by Sconstruct

```
SConscript('SConscript', build_dir='.build_release', duplicate=0, exports={'MODE':'release'})
SConscript('SConscript', build_dir='.build_debug', duplicate=0, exports={'MODE':'debug'})
```

SConscript

```
project_root/  (new project that builds bar app using the libfoo built from source)

    libfoo_subrepo/  (standalone project repo from bitbucket)
        src/
            SConscript
            libfoo.c
            libfoo.h
        test/
            SConscript
            test_foo.c
        SConstruct
        SConscript

    barapp_subrepo/  (standalone project repo from bitbucket that uses libfoo)
        src/
            SConscript
            bar.c
            bar.h
        test/
            SConscript
            test_bar.c
        SConstruct
        SConscript

    test/
        SConscript
        test_bar_with_foo.c
    SConstruct
```

- project_root/SConstruct

```
# This SConstruct orchestrates building 3 subdirs

import os

subdirs = ['libfoo_subrepo', 'barapp_subrepo', 'test']
env = Environment()

for dir in subdirs:
    SConscript(os.path.join(dir, 'SConscript'), exports = ['env'])
```

- libfoo_subrepo/SConstruct

```
# This SConstruct does nothing more than load the SConscript in this dir
# The Environment() is created in the SConstruct script
# This dir can be built standalone by executing scons here, or together
# by executing scons in the parent directory

env = Environment()
SConscript('SConscript', exports = ['env'])
```

- libfoo_subrepo/SConscript

```
# This SConstruct orchestrates building 2 subdirs
import os

Import('env')
subdirs = ['src', 'test']

for dir in subdirs:
    SConscript(os.path.join(dir, 'SConscript'), exports = ['env'])
```

- barapp_subrepo/SConstruct

```
# This SConstruct does nothing more than load the SConscript in this dir
# The Environment() is created in the SConstruct script
# This dir can be build standalone by executing scons here, or together
# by executing scons in the parent directory
env = Environment()
SConscript('SConscript', exports = ['env'])
```

- barapp_subrepo/SConscript

```
# This SConstruct orchestrates building 2 subdirs
import os

Import('env')
subdirs = ['src', 'test']

for dir in subdirs:
    SConscript(os.path.join(dir, 'SConscript'), exports = ['env'])
```

Sharing Environments

- **Exporting Variables**

```
env = Environment()
Export('env')
```

```
env = Environment()
debug = ARGUMENTS['debug']
Export('env', 'debug')              # Export('env debug')
```

Sharing Environments

- **Exporting Variables**

```
SConscript('src/SConscript', 'env')

SConscript('src/SConscript', exports='env')  # same
```

```
SConscript(['src1/SConscript',
            'src2/SConscript'], exports='env')
```

Sharing Environments

- **Importing Variables**

```
Import('env')

env.Program('prog', ['prog.c'])
```

```
Import('env', 'debug')              # Import('env debug')

env = env.Clone(DEBUG = debug)
env.Program('prog', ['prog.c'])
```

```
Import('*')

env = env.Clone(DEBUG = debug)
env.Program('prog', ['prog.c'])
```

Returning Values from an Sconscript File

```
# SConscript
env = Environment()
Export('env')
objs = []
for subdir in ['foo', 'bar']:
    o = SConscript('%s/SConscript' % subdir)
    objs.append(o)
env.Library('prog', objs)
```

```
# foo/Sconscript

Import('env')
obj = env.Object('foo.c')
Return('obj')
```

VariantDir

```
VariantDir(variant_dir, src_dir, [duplicate])
env.VariantDir(variant_dir, src_dir, [duplicate])
```

src_dir 디렉토리 트리가 variant_dir로 복사된다.
variant_dir에서 빌드할 목적으로 사용된다.

```
VariantDir('build-variant1', 'src')
SConscript('build-variant1/SConscript')

VariantDir('build-variant2', 'src')
SConscript('build-variant2/SConscript')
```

VariantDir

```
VariantDir('build', 'src')
env = Environment()
env.Program('build/hello.c')
```

```
% scons -Q
cc -o build/hello.o -c build/hello.c
cc -o build/hello build/hello.o

% ls build
hello  hello.c  hello.o
```

VariantDir

```
VariantDir('build', 'src', duplicate=0)
env = Environment()
env.Program('build/hello.c')
```

```
% scons -Q
cc -o build/hello.o -c src/hello.c
cc -o build/hello build/hello.o

% ls build
hello  hello.o
```

From make to Scons example

- http://www.bravegnu.org/blog/make-to-scons.html