REPRODUCIBLE COMPUTATIONAL EXPERIMENTS USING SCONS

S. Fomel

Bureau of Economic Geology
University of Texas at Austin
sergey.fomel@beg.utexas.edu

G. Hennenfent

Department of Earth and Ocean Sciences
University of British Columbia
ghennenfent@eos.ubc.ca

ABSTRACT

SCons (from Software Construction) is a well-known open-source program designed primarily for building software. In this paper, we describe our method of extending SCons for managing data processing flows and reproducible computational experiments. We demonstrate our usage of SCons with a simple example.

***Index Terms*—** Software maintenance, Software reusability, Software tools, Data processing, Signal processing

1. INTRODUCTION

This paper introduces an environment for reproducible computational experiments developed as part of the "Madagascar" software package. To reproduce the example experiments in this paper, you can download Madagascar from `http://rsf.sourceforge.net/`. At the moment, the main Madagascar interface is the Unix shell command line so that you will need a Unix/POSIX system or Unix emulation under Windows. Our focus, however, is not only on particular tools we use in our research but also on the general philosophy of reproducible computations.

1.1. Reproducible research philosophy

Peer review is the backbone of scientific progress. From the ancient alchemists, who worked in secret on magic solutions to insolvable problems, the modern science has come a long way to become a social enterprise, where hypotheses, theories, and experimental results are openly published and verified by the community. By reproducing and verifying previously published research, a researcher can take new steps to advance the progress of science.

During the last century, computational studies emerged as a new scientific discipline. Computational experiments are carried out on a computer by applying numerical algorithms to digital data. How reproducible are such experiments? On one hand, reproducing the result of a numerical experiment is a difficult undertaking. The reader needs to have access to precisely the same kind of input data, software and hardware as the author of the publication in order to reproduce the published result. It is often difficult or impossible to provide detailed specifications for these components. On the other hand, basic computational system components such as operating systems and file formats are getting increasingly standardized, and new components can be shared in principle because they simply represent digital information transferable over the Internet.

The practice of software sharing has fueled the miraculously efficient development of Linux, Apache, and many other open-source software projects. Its proponents often refer to this ideology as an analog of the scientific peer review tradition. Eric Raymond writes [1]: "*Abandoning the habit of secrecy in favor of process transparency and peer review was the crucial step by which alchemy became chemistry. In the same way, it is beginning to appear that open-source development may signal the long-awaited maturation of software development as a discipline.*" While software development is trying to imitate science, computational science needs to borrow from the open-source model in order to sustain itself as a fully scientific discipline. In words of Randy LeVeque, [2], "*Within the world of science, computation is now rightly seen as a third vertex of a triangle complementing experiment and theory. However, as it is now often practiced, one can make a good case that computing is the last refuge of the scientific scoundrel [...] Where else in science can one get away with publishing observations that are claimed to prove a theory or illustrate the success of a technique without having to give a careful description of the methods used, in sufficient detail that others can attempt to repeat the experiment?*"

Nearly ten years ago, the technology of reproducible research was pioneered by Jon Claerbout and his students at the Stanford Exploration Project (SEP). SEP's system of reproducible research requires the author of a publication to document creation of numerical results from the input data and software sources to let others test and verify the result reproducibility [3, 4]. The discipline of reproducible research was also adopted and popularized in the statistics and wavelet theory community by Buckheit and Donoho [5, 6, 7]. However, the adoption or reproducible research practice by computational scientists and engineers has been slow. Partially, this is caused by difficult and inadequate tools.

1.2. Tools for reproducible research

The reproducible research system developed at SEP is based on "make [8]", a Unix software construction utility. The "make" program keeps track of dependencies between different components of the system and the software construction targets, which, in the case of a reproducible research system, turn into figures and manuscripts. The targets and commands for their construction are specified by the author in "makefiles", which serve as databases for defining source and target dependencies. A dependency-based system leads to rapid development, because when one of the sources changes, only parts that depend on this source get recomputed.

Unfortunately, "make" is not well designed from the user experience prospective. It employs an obscure and limited special language, which often appears confusing to unexperienced users. According to Peter van der Linden [9], "*`Sendmail` and `make` are two well known programs that are pretty widely regarded as originally being debugged into existence. That's why their command languages are so poorly thought out and difficult to learn.*" The inconvenience of "make" command language is also in its limited capabilities. The reproducible research system developed by Schwab et al [4] includes not only custom "make" rules but also an obscure and non-portable agglomeration of shell and Perl extensions

Several alternative systems for dependency-checking software construction have been developed in recent years. One of the most promising new tools is SCons, enthusiastically endorsed by Dubois [10]. The SCons initial design won the Software Carpentry competition sponsored by Los Alamos National Laboratory in 2000 in the category of "a dependency management tool to replace make". Some of the main advantages of SCons are:

- SCons configuration files are Python scripts. Python is a modern programming language praised for its readability, elegance, simplicity, and power [11].
- SCons offers reliable, automatic, and extensible dependency analysis and creates a global view of all dependencies.
- While "make" relies on timestamps for detecting file changes (creating numerous problems on platforms with different system clocks), SCons uses by default a more reliable detection mechanism employing MD5 signatures. It can detect changes not only in files but also in commands used to build them.
- SCons is publicly released under a liberal open-source license[1]

In this paper, we propose to adopt SCons as a new platform for reproducible research in scientific computing and signal processing.

[1] As of time of this writing, SCons is in a beta version 0.96 approaching the 1.0 official release. See `http://www.scons.org/`.

2. MADAGASCAR PACKAGE OVERVIEW

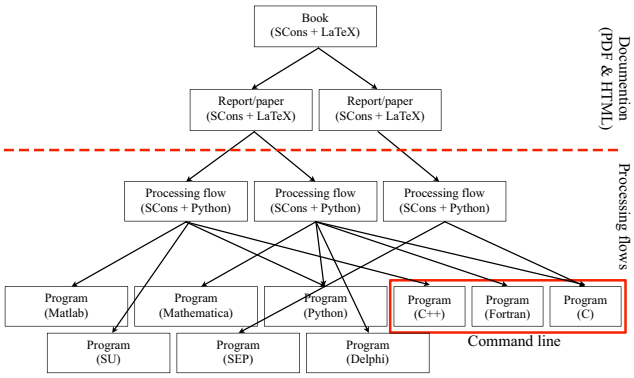

Fig. 1. `Madagascar`'s multi-layer structure.

`Madagascar` is a multi-layered software package (Fig. 1).

2.1. Command line

`Madagascar` is first of all a collection of command line programs. Most programs act as filters on input data and can be chained in a Unix pipeline, e.g.

```
sfspike n1=200 | sfnoise rep=y >noise.rsf
```

Although these programs mainly focus at this point on geophysical applications, users can use the API (application programmer's interface) for writing their own software to manipulate Regularly Sampled Format (RSF) files, `Madagascar` file format. The main software language of `Madagascar` is C. Interfaces to other languages (C++, Fortran-77, Fortran-90, Python) are also provided.

2.2. Data processing flows

`Madagascar` is also an environment for reproducible numerical experiments in a very broad sense. These numerical experiments (or "computational recipes") can be done not only using `Madagascar` command line programs but also Matlab, Mathematica, Python, or other seismic packages (e.g. SEP, Seismic Unix). We adopted SCons for this part as we shall demonstrate later.

2.3. Research documentation

The most upper layer of `Madagascar` and maybe the most critical for reproducible research is documentation. It establishes a direct link between the figures of a paper or a report and the codes that were used to generate them. This layer uses SCons in combination with LaTeX to generate PDF, HTML, and MediaWiki files real easy and undoubtly makes `Madagascar` a convenient environment for technology transfer, report, thesis, and peer-reviewed publication writing.

3. EXAMPLE REPRODUCIBLE EXPERIMENT

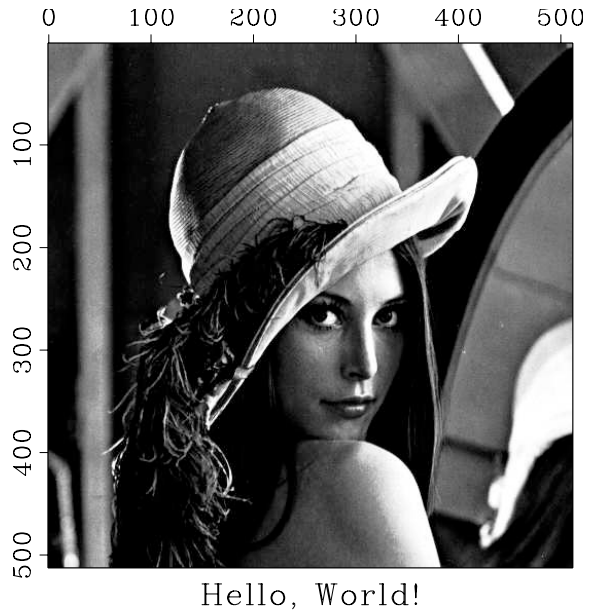

Fig. 2. The output of the first numerical experiment.

The main `SConstruct` commands defined in our reproducible research environment are collected in Table 3.

To follow the first example, select a working project directory and copy the following code to a file named `SConstruct`[2].

```
from rsfproj import *

# Download the input data file
Fetch('lena.img','imgs')

# Create RSF header
Flow('lena.hdr','lena.img',
     '''echo n1=512 n2=513 in=$SOURCE
     data_format=native_uchar''',stdin=0)

# Convert to float and window out first trace
Flow('lena','lena.hdr',
     'dd type=float | window f2=1')

# Display
Result('lena',
       '''
       grey title="Hello, World!" transp=n
       color=b bias=128 clip=100 screenratio=1
       ''')

# Wrap up
End()
```

[2]The source of this file is also accessible at book/rsf/scons/easystart/SConstruct.

This is our "hello world" example that illustrates the basic use of some of the commands presented in Table 3. The plan for this experiment is simply to download data from a public data server, to convert it to an appropriate file format and to generate a figure for publication. But let us have a closer look at the `SConstruct` script and try to decorticate it.

```
from rsfproj import *
```

is a standard Python command that loads the Madagascar project management module `rsfproj.py` which provides our extension to SCons.

```
Fetch('lena.img','imgs')
```

instructs SCons to connect to a public data server (the default server if no FTP server information is provided) and to fetch the data file `lena.img` from the `data/imgs` directory. Try running "`scons -Q lena.img`" on the command line. The successful output should look like

```
bash$ scons -Q lena.img
retrieve(["lena.img"], [])
```

with the target file `lena.img` appearing in your directory. In the following examples, we will use `-Q` (quiet) option of `scons` to suppress the verbose output.

```
Flow('lena.hdr','lena.img',
     '''echo n1=512 n2=513 in=$SOURCE
     data_format=native_uchar''',stdin=0)
```

prepares the Madagascar header file `lena.hdr` using the standard Unix command `echo`.

```
bash$ scons -Q lena.hdr
echo n1=512 n2=513 in=lena.img \
data_format=native_uchar > lena.hdr
```

Since `echo` does not take a standard input, stdin is set to 0 in the Flow command otherwise the first source is the standard input. Likewise, the first target is the standard output unless otherwise specified. Note that `lena.img` is referred as `$SOURCE` in the command. This allows us to change the name of the source file without changing the command.

The data format of the `lena.img` image file is `uchar` (unsigned character), the image consists of 513 traces with 512 samples per trace. Our next step is to convert the image representation to floating point numbers and to window out the first trace so that the final image is a 512 by 512 square. The two transformations are conveniently combined into one with the help of a Unix pipe.

```
scons -Q lena.rsf
< lena.hdr /RSF/bin/sfdd type=float | \
/RSF/bin/sfwindow f2=1 > lena.rsf
```

Table 1. Basic methods of an `rsfproj` object.

`Fetch(data_file,dir[,ftp_server_info])` A rule to download <data_file> from a specific directory <dir> of an FTP server <ftp_server_info>.
`Flow(target[s],source[s],command[s][,stdin][,stdout])` A rule to generate <target[s]> from <source[s]> using command[s]
`Plot(intermediate_plot[,source],plot_command)` or `Plot(intermediate_plot,intermediate_plots,combination)` A rule to generate <intermediate_plot> in the working directory.
`Result(plot[,source],plot_command)` or `Result(plot,intermediate_plots,combination)` A rule to generate a final <plot> in the special `Fig` folder of the working directory.
`End()` A rule to collect default targets.

Notice that Madagascar modules `sfdd` and `sfwindow` get substituted for the corresponding short names in the `SConstruct` file. The file `lena.rsf` is in a regularly sampled format. In the last step, we will create a plot file for displaying the image on the screen and for including it in the publication.

```
Result('lena',
       '''
       grey title="Hello, World!" transp=n
       color=b bias=128 clip=100 screenratio=1
       ''')
```

Notice that we broke long command strings into multiple lines by using Python's triple quote syntax. The `Result` command has special targets associated with it. Try, for example, "`scons lena.view`" to observe the figure `Fig/lena.vpl` generated in a specially created `Fig` directory and displayed on the screen. The output should look like Figure 2.

1. Run `scons -c`. The `-c` (clean) option tells SCons to remove all default targets (the `Fig/lena.vpl` image file in our case) and also all intermediate targets that it generated. Run `scons` again, and the default target will be regenerated.

2. Edit your `SConstruct` file and change some of the plotting parameters. For example, change the value of `clip` from `clip=100` to `clip=50`. Run `scons` again and observe that only the last part of the processing flow (precisely, the part affected by the parameter change) is being run. `SCons` is smart enough to recognize that your editing did not affect any of the previous results in the data flow chain! Keeping track of dependencies is the main feature that separates data processing and computational experimenting with SCons from using linear shell scripts. For computationally demanding data processing, this feature can save you a lot of time and can make your signal processing experiments more efficient and enjoyable.

4. REFERENCES

[1] E. S. Raymond, *The Art of UNIX programming*, Addison-Wesley, 2004.

[2] R. J. LeVeque, "Wave propagation software, computational science, and reproducible research," in *Proc. International Congress of Mathematicians*, to appear, 2006.

[3] J. Claerbout, "Electronic documents give reproducible research a new meaning," in *62nd Ann. Internat. Mtg*. 1992, pp. 601–604, Soc. of Expl. Geophys.

[4] M. Schwab, M. Karrenbach, and J. Claerbout, "Making scientific computations reproducible," *Computing in Science & Engineering*, vol. 2, pp. 61–67, 2000.

[5] J. Buckheit and D. L. Donoho, "Wavelab and reproducible research," in *Wavelets and Statistics*, vol. 103, pp. 55–81. Springer-Verlag, 1995.

[6] B. B. Hubbard, *The World According to Wavelets: The Story of a Mathematical Technique in the Making*, AK Peters, 1998.

[7] S. Mallat, *A wavelet tour of signal processing*, Academic Press, 1999.

[8] R. M. Stallman, R. McGrath, and P. D. Smith, *GNU make: A Program for Directing Recompilation*, GNU Press, 2004.

[9] P. van der Linden, *Expert C Programming*, Prentice Hall, 1994.

[10] P. F. Dubois, "Why Johnny can't build," *Computing in Science & Engineering*, vol. 5, no. 5, pp. 83–88, 2003.

[11] G. Van Rossum, *Python Tutorial*, Iuniverse Inc, 2000.