repro User's Guide

Gilles Hennenfent & Sean Ross-Ross
Seismic Laboratory for Imaging & Modeling
Earth & Ocean Sciences Department
University of British Columbia

September 12, 2008

Abstract

repro is a Python package for automating reproducible research in scientific computing. repro works in combination with SCons, a next-generation build tool. The package is freely available over the Internet. Downloading and installation instructions are provided in this guide.

The repro package is documented in various ways (many comments in source code, this guide—written using repro itself!—and a reference guide). In this user's guide, we present a few pedagogical examples that uses Matlab, Python, Seismic Unix (SU), and Madagascar. We also include demo papers. These papers are written in LaTeX and compiled using repro. The figures they contain are automatically generated from the source codes provided. In that sense, the demo papers are a model of self-contained documents that are fully reproducible.

The repro package is largely inspired by some parts of Madagascar, a geophysical software package for reproducible research. However, the repro package is intended for a broad audience coming from a wide spectrum of interest areas.

Contents

1 Introduction

To be written…

2 Download and installation

The repro package contains mainly Python scripts and SConstruct files. The whole package requires less than 1.5MB on disk once it is downloaded and installed.

2.1 Requirements

The requirements vary depending on what you plan to do with repro.

2.1.1 Download

Downloading the repro package requires

- a Subversion client (subversion.tigris.org)

Subversion is an open source version control system.

2.1.2 Installation

Installing the repro package requires

- Python 2.4 or later (www.python.org)

Python, already installed on most Linux/Unix, and Mac OS X machines, is an open source dynamic object-oriented programming language.

2.1.3 Execution

Using the repro package requires at least

- Python 2.4 or later (www.python.org)
- SCons 0.98.5 or later (www.scons.org)

SCons is an open source cross-platform software construction tool written in Python.

There are, however, additional packages required to use some functionalities of the repro package.

LaTeX compilation:

- pdflatex
- bibtex

You need to install both programs if you want this functionality.

Website building from LaTeX source code:

- pdflatex
- bibtex
- latex2html

You need to install all these program if you want this functionality.

Image conversion:

- epstopdf (from PS, EPS to PDF)
- pstoimg (from PS, EPS, PDF to PNG)
- supsimage (from SU to EPS)
- vplot2eps (from VPL to EPS)

You need to install the program(s) that perform(s) the conversion(s) you are interested in.

Figure display:

- xpdf (PDF)
- gv (EPS)
- xv (JPEG, PNG)
- suximage (SU)

You need to install the program(s) that display(s) the file format(s) you are interested in.

Developer's guide compilation:

- doxygen

You need to install this program if you want this functionality.

2.2 Download

The source code resides at `https://repro.svn.sourceforge.net/svnroot/repro/TRUNK`. Running the following command

```
svn co https://repro.svn.sourceforge.net/svnroot/repro/TRUNK repro
```

will create a folder "repro" in your current directory. This folder contains all of the source code of the repro package.

2.3 Installation

Assuming your system satisfies the installation requirements, install the repro package simply by running, in the folder "repro", the provided Python-standard setup script as follows:

```
python setup.py install
```

If you do not have system installation privileges, you can use the –prefix option to specify an alternate installation location:

```
python setup.py install --prefix=/Path/To/Install/Directory
```

This will install the repro package in the appropriate locations relative to the prefix /Path/To/Install/Directory–that is, the Python module of the repro package will be in /Path/To/Install/Directory/lib/pythonX.X/site-packages, where X.X is the version number of the Python you used to installed this package. You must add this path to your PYTHONPATH environment variable in order to use the repro package.

3 Getting started

In this chapter, you see several very simple examples using the repro package. These examples are included in the source code of the package (*GettingStarted* in the *examples* folder) and demonstrate how easy it is to use the repro package to generate fully reproducible documents.

3.1 Running simple Matlab scripts

We discuss different options to help you discover the repro package and SCons (see www.scons.org for more details).

Step 1: generate data

Here is a simple Matlab script named *hello.m*

```
rmat = randn(10,20);

save filename
```

This script saves the variable *rmat* to the binary MAT-file *filename.mat*.

And here is how to run it using the repro package. Enter the following into a file named *SConstruct*:

```
from repro import *

Matlab( 'filename', None, matfuncs =["hello"])
```

This minimal file contains enough information to run the Matlab script: 1) the output file is *filename.mat* (the suffix is .mat unless otherwise specified); 2) there is no input (None is a Python reserved keyword); and 3) the Matlab function to run is called *hello.m*.

Now issue the scons command to generate the MAT-file.

```
scons
```

The MAT-file is generated!

Notice that, if you issue the scons command again, nothing happens beside a SCons message to notify you that your current directory is up to date. SCons only runs commands if something has changed. Modify the size of the random matrix in your Matlab script to verify that SCons runs again the Matlab command.

To clean up, you simply use the -c or –clean option when you invoke SCons, e.g.,

```
scons -c
```

Step 2: generate data with more control

If you change *filename* to *test* in the *SConstruct* file of the previous example and run the scons command, you will see that SCons does not create *test.mat* but still *filename.mat*. That's because the name of the MAT-file in the *SConstruct* file and in the Matlab script are not linked.

To change that, update your Matlab script as follows:

```
function hello( filename )

rmat = randn(10,20);

save(filename, 'rmat')
```

so that it becomes a Matlab function that takes a filename as input. You also need to update your *SConstruct* file

```
from repro import *

Matlab( 'filename', None, matfuncs =["hello('$TARGET')"])
```

Notice how we pass an argument, $TARGET, to *hello.m*. $TARGET is a SCons variable that refers to the target name. Try different target names and verify that SCons works again as expected.

Step 3: display data

Here is the Matlab file of the previous section

```
function hello( filename )

rmat = randn(10,20);

save(filename, 'rmat')
```

We now want to display, in a separate Matlab script, the random matrix *rmat* and save this figure to an EPS file. Here is this script

```
function MakeFigure( filename, epsname )

load( filename )

figure
imagesc(rmat)

print('-depsc2','-r300',epsname)
```

The *SConstruct* file to run these two Matlab scripts is as follows:

```
from repro import *

Matlab( 'filename', None, matfuncs =["hello('$TARGET')"])

Matlab( 'figure.eps', 'filename', matfuncs =["MakeFigure('$SOURCE','$TARGET')"])
```

You probably already guessed that $SOURCE is a SCons variable that refers to the source name. Now issue the scons command to generate the EPS file. SCons first creates *filename.mat* and then *figure.eps* since this file depends on the first one. Notice, however, that you can change the order of the Matlab commands in the *SConstruct* file without prejudice. That's because an *SConstruct* file is order-independent. The sequence of commands that SCons runs to generate a target is determined by a dependency tree, i.e., a tree that contains dependency relations between the different files that SCons has to create.

For interest, add a title to the figure in *MakeFigure.m* and issue again the scons command. Only *figure.eps* is re-generated from *filename.mat* but SCons does not re-generate *filename.mat* because it knows it is up to date.

Finally, rather than manually opening the EPS file with your favorite viewer, you can use one extra line in the *SConstruct* file

```
View( 'figure.eps' )
```

and simply issue the scons view or scons figure.view command

```
scons view
```

The view option is an alias to display all the figures generated by the *SConstruct* file. Otherwise, just use target base name with the .view suffix.

3.2 Including a Matlab figure in a LaTeX document

We use *hello.m* and *MakeFigure.m* from the previous section as a starting point. The LaTeX source code, named *paper.ltx*, reads:

```
\documentclass{article}

\usepackage{graphicx}

\author{You and only you!}

\title{Your first document using the \textbf{repro} package}

\begin{document}

\maketitle

\begin{figure}
  \begin{center}
    \includegraphics[width=.4\textwidth]{figure.pdf}
  \end{center}
  \caption{Generated using Matlab}
\end{figure}

If you find \textbf{repro} useful and work with it to write publications, feel
free to reference \cite{hennenfent08rap}. Thanks!

\bibliographystyle{plain}
\bibliography{bib}

\end{document}
```

and the bibliography file included, *bib.bib*, is just

```
@manual{hennenfent08rap,
    Author = {Gilles Hennenfent and Sean Ross-Ross},
    Institution = {The University of British Columbia},
    Keywords = {SLIM},
    Month = {August},
    Organization = {Seismic Laboratory for Imaging and Modeling},
    Title = {repro: a Python package for automating reproducible research in scientific computing},
    Type = {software package},
    Year = {2008},
    note = {http://repro.sf.net}
    }
```

The *SConstruct* file to generate the Matlab figure and compile the LaTeX document is as follows:

```
from repro import *

Matlab( 'filename', None, matfuncs =["hello('$TARGET')"])

Matlab( 'figure.eps', 'filename', matfuncs =["MakeFigure('$SOURCE','$TARGET')"])

PDF( 'figure.eps' )

PDF( 'paper.ltx' )
```

Notice that the LaTeX source code includes a PDF file but *MakeFig.m* generates an EPS file. That's why there is the PDF command with *figure.eps* as argument. The target name, *figure.pdf*, is implicit. The LaTeX document is also compiled using the PDF command. The target name, *paper.pdf*, is again implicit.

Now run the scons or scons paper.pdf command.

```
scons paper.pdf
```

Did you notice? Not only did SCons compile your LaTeX document but it also generated first the figure that you included! The good news go beyond that. SCons actually ran pdflatex, bibtex, pdflatex, pdflatex by itself because it detected that your LaTeX document includes a bibliography. And there is more. Make any change to the bibliography, the LaTeX source code, or to the way the figure is generated and SCons will update *paper.pdf*! To achieve this, the PDF command uses a scanner on LaTeX files to find implicit dependencies.

4 repro builders

The repro package adds or extends several SCons builders to facilitate the organization of a reproducible project.

4.1 Fetch file(s) from a remote location

- WGET
- FTP

4.2 Process data

- Command (from SCons, general use)
- Matlab (run Matlab scritps)
- Rush (run RSF programs)
- GetPar (get parameter from RSF file)

4.3 Display data

- View (popup display for .vpl, .pdf, .jpeg, .su. and .eps figures)

4.4 Document data processing

- Preamble (gather LaTeX pieces into full LaTeX document)
- PDF (from SCons, generate .pdf from full LaTeX document)
- HTML (generate website from full LaTeX document)

4.5 Utilities

- PDF (from SCons, extended to generate .pdf from .vpl, .su, and .eps)
- PNG (generate .png from .vpl, .su, .pdf, and .eps)
- EPS (generate .eps from .vpl, and .su)

5 More examples

Check the *examples* folder of the repro package to discover all examples. Here are a few instances.

```
# load repro to gain access to WGET builder
from repro import *

# fetch RSF data file, shotdepp.p.rsf, from SLIM
# public FTP server
WGET( 'shotdeep.p.rsf', None,
      user='anonymous',
      url='ftp://slim.eos.ubc.ca',
      path='data/synth/delphi' )

# fetch README file from SLIM private FTP server
# password asked at run time
# WARNING: command fails without valid password
WGET( 'README', None,
      user='private',
      password='$ASK',
      url='ftp://slim.eos.ubc.ca',
      path='data/synth/CVX/Tuiko2')

# load os to gain access to environ
from os import environ

# fetch README file from SLIM private FTP server
# password assumed available from SLIM_PASSWORD
# environment variable
# WARNING: command fails without valid password
WGET( 'FILE', None,
      user='private',
      password=environ.get('SLIM_PASSWORD'),
      url='ftp://slim.eos.ubc.ca',
      path='data/users/hegilles/misc')

# define an environment for WGET with common parameters
wgetenv  = Environment( user='anonymous',
                        url='ftp://slim.eos.ubc.ca',
                        path='data/users/hegilles/Fig' )

# define list of figures to fetch
figs = ['bincentering.pdf',
        'regularization.pdf']

# fetch figures
for fig in figs:
    wgetenv.WGET( fig, None )
```

```
# load repro to gain access to FTP builder
from repro import *

# fetch RSF data file, shotdepp.p.rsf, from SLIM
# public FTP server
FTP( 'shotdeep.p.rsf', None,
     user='anonymous',
     host='slim.eos.ubc.ca',
     path='data/synth/delphi' )

# fetch README file from SLIM private FTP server
# WARNING: command fails without valid password
FTP( 'README', None,
     user='private',
     host='slim.eos.ubc.ca',
     path='data/synth/CVX/Tuiko2')

# define an environment for FTP with common parameters
ftpenv  = Environment( user='anonymous',
                       host='slim.eos.ubc.ca',
                       path='data/users/hegilles/Fig' )

# define list of figures to fetch
figs = ['bincentering.pdf',
        'regularization.pdf']

# fetch figures
for fig in figs:
    ftpenv.FTP( fig, None )
```

```
# DESCRIPTION: load repro to gain access to the extended
# PDF builder.
from repro import *
# HELP: SCons already have a PDF builder but repro extends
# its functionality. * actually means all the builders
# that repro contains.

# DESCRIPTION: make a PDF, named hello.pdf, out of LaTeX document
PDF( 'hello.ltx' )
# HELP: the source is 'hello.ltx', the target is implicit
# (SCons makes an educated guess based on the definition of
# PDF). The target name is constructed using the base of source
# and suffix .pdf, i.e., hello.pdf.
```

```
# load repro to gain access to PDF, Preamble, and HTML builders
from repro import *

# import join from os.path module to build platform-independent path
from os.path import join

# instruct SCons to store file signatures in a separate .sconsign
# file in each directory
SConsignFile( None )

# create a hierarchical build, i.e., connect this SConstruct to
# the one in the "proj1" folder
SConscript( join('proj1','SConstruct') )

# generate LaTeX document with preamble from different pieces
```

```
Preamble('paper.ltx',['opening.tex','section1.tex',
                      'section2.tex','bib.tex'],
        packages = ['graphicx'] )

# make a PDF, named paper.pdf, out of LaTeX document
PDF( 'paper.ltx' )

# make a website, stored in PaperWebsite folder, out of LaTeX document
HTML( 'PaperWebsite','paper.ltx' )
```

```
# load repro to gain access to PDF, and Preamble builders
from repro import *

# define custom LaTeX commands
custom_include = """
\\renewcommand{\\vector}[1]{\\ensuremath{\\mathbf{\\MakeLowercase{#1}}}}
\\hyphenation{hy-phen-ate}
"""

# gather all .tex files in report.ltx.
# documentclass is report (article is default), classoptions
# are 10pt, and a4paper (no options by default), two packages
# (hyperref and amsmath) are included (none included by default),
# and custom LaTeX commands defined above included (nothing
# included by default).
Preamble('report.ltx',['AuthorsInfo.tex','Title.tex',
                       'Abstract.tex','Section1.tex',
                       'Section2.tex'],
        documentclass= 'report',
        classoptions = ['10pt','a4paper'],
        packages     = ['hyperref','amsmath'],
        include      = custom_include )

# make pdf, named report.pdf, out of LaTeX document
PDF( 'report.ltx' )

# gather some .tex files in article.ltx.
Preamble('article.ltx',['AuthorsInfo.tex','Title.tex',
                        'Abstract.tex','Section2.tex'],
        classoptions = ['12pt'],
        packages     = ['hyperref','amsmath'])

# make pdf, named article.pdf, out of LaTeX document
PDF( 'article.ltx' )
```

```
# DESCRIPTION: load repro to gain access to Matlab builder
from repro import *
# HELP: * actually means all the builders that repro
# contains.

# DESCRIPTION: create MAT-file
Matlab( 'filename', None, matfuncs =["hello('$TARGET')"])
# HELP: 'filename.mat' is the target, the suffix is .mat if
# no suffix is specified. There is no source required (None
# is a  reserverd keyworkd in Python). $TARGET refers to the
# first target, i.e., 'filename.mat'.
```

```
# load repro to gain access to Matlab builder
from repro import *

# define an environment for Matlab with common parameters
# MATLABOPTS refers to the options passed on the command
# line to Matlab, in this case, no Java mode and no splash.
# matpath is a Python list of relative path you want to add
# to the Matlab path.
matenv = Environment( MATLABOPTS = '-nojvm -nosplash -memmgr debug',
                      matpath  =['matfuncs'])

# create a MAT-file in the matenv, otherwise mfile will not
# be found (file not in the current directory).
matenv.Matlab( 'file', None, matfuncs =["mfile('$TARGET')"])
# 'file.mat' is the target, the suffix is .mat if
# no suffix is specified. There is no source required (None
# is a  reserverd keyworkd in Python). $TARGET refers to the
# first target, i.e., 'file.mat'.

# define python variable
matvar = 10

# run mfile1.m and mfile2.m in the same Matlab session
# $SOURCE is a SCons variable that refers to 'file.mat',
# $TARGET refers to 'file1.mat' and ${TARGETS[1]} to
# 'file2.mat' (Python indices start at 0). %vars() is a Python
# function that returns a dictionary of name, value of variables
# in the current workspace.
matenv.Matlab( ['file1.mat','file2.mat'],'file.mat',
               matfuncs =["mfile1('$SOURCE','$TARGET')",
                          "mfile2('${TARGETS[1]}',%(matvar)d)"%vars()])
```

```
# load repro to gain access to Matlab builder
```

```
from repro import *

# instruct SCons to store file signatures in a separate .sconsign
# file in each directory
SConsignFile( None )

# generate a LaTeX table from Matlab with dynamic values
Matlab( 'dummytable.tex',None,
        matfuncs =["GenerateTable('$TARGET')"])
```

```
# load repro to gain access to Matlab, View, and PNG builders
from repro import *

# instruct SCons to store file signatures in a separate .sconsign
# file in each directory
SConsignFile( None )

# loop over the different Matlab scripts to produce EPS figures
figs = []
for i in range(1,6):
    figs += Matlab( 'fig%d.eps'%i,None,
                    matfuncs =["SSSRRRFig%d('$TARGET')"%i])

# view EPS figures
View( figs )

# generate PNG figures, used for the paper website, from EPS figures
PNG( figs, density=100 )
```

```
# DESCRIPTION: load repro to gain access to View, and
# PNG builders. Command is a standard SCons function
from repro import *
# HELP: * actually means all the builders that repro
# contains.

# DESCRIPTION: create common offset data with three planes
Command('plane.su',None,'suplane dip1=1 >$TARGET')
# HELP: 'plane.su' is the target, there is no
# source required (None is a reserved keyword in Python).
# The command to issue is 'suplane dip1=1 >$TARGET', where
# $TARGET refers to the first target, i.e., plane.su.

# DESCRIPTION: view SU data
View( 'plane.su', XIMAGEOPTS="cmap=hsv'2' " )
# HELP: 'plane.su' is the file to view, XIMAGEOPTS are the
# options passed to the SU viewer.
```

```
# example adapted from SU demos

# load repro to gain access to View, and PNG builders
# Command is a standard SCons function
from repro import *

# import join from string module to construct long commands
from string import join

# define size of the synthetic data
nt  =101
nxm =101

# define reflectors
ref = ((0.0,0.5),
       (1.0,0.5),
       (2.0,0.8),
       (2.5,1.0),
       (3.0,0.8),
       (4.0,0.5),
       (5.0,0.5))

# define labels for plots
def PlotOpts(title=None):
    labels = 'label1="Time (s)" label2="Midpoint (km)" '
    if title is not None:
        return labels+'title=%s '%title
    else:
        return labels

# make synthetic data
syn = Command( 'syn.su', None,
               join( ['susynlv nt=%d dt=0.04 ft=0.0 nxo=1 '%nt,
                      'nxm=%d dxm=.05 fxm=0.0 er=0 ob=1 '%nxm,
                      'v00=1.0 dvdz=0 dvdx=0 smooth=1 ',
                      'ref="%s" | '%(';'.join(map(lambda x: ','.join(map(str,x)),ref)) ),
                      'sushw key=d2 a=.05 >$TARGET']) )

# view data
View( syn, XIMAGEOPTS=PlotOpts("Synthetic data") )

# migrate data
mig = Command('mig.su',syn ,'<$SOURCE sugazmig tmig=0 vmig=1 >$TARGET')
```

```
# view migrated data
View( mig, XIMAGEOPTS=PlotOpts("Phase shift migration") )

# generate PNG figure, used for the paper website, from SU data
PNG( [syn, mig],  PSIMAGEOPTS=PlotOpts() )
```