User-space modules and packages on SciNet

SNUG TechTalk

March 13, 2011

SciNet compute • calcul CANADA

Why should I use local user-space modules?

- There are many potential optional packages for a number of software packages that users could potentially want.
- In fact, there are too many of them, with potential conflicts and dependencies, to be maintainable system-wide.
- Local user-space package and modules are almost certainly the easiest way to deal with the wide range of packages, ensure they're up to date, and ensure that users package choices don't conflict.

SciNet compute • calcul CANADA

Some possible local user-space installations

1. Local builds
2. Local modules
3. Local Python packages
4. Local Perl packages
5. Local R packages

1 Local builds: life without a package manager

General procedure

1 Local builds: life without a package manager

General procedure

1. Download source code package

1 Local builds: life without a package manager

General procedure

1. Download source code package
2. Install locally

```
$ tar -xzvf package-name.tgz  or
$ tar -xjvf package-name.bz2
$ cd package-name
$./configure --prefix=$HOME/somedir [CC=icc CXX=icpc ... ]
$ make
$ make install
```

1 Local builds: life without a package manager

General procedure

1. Download source code package
2. Install locally

```
$ tar -xzvf package-name.tgz   or
$ tar -xjvf package-name.bz2
$ cd package-name   specify directory
$./configure --prefix=$HOME/somedir [CC=icc CXX=icpc ... ]
$ make
$ make install
```

1 Local builds: life without a package manager

General procedure

1. Download source code package
2. Install locally

```
$ tar -xzvf package-name.tgz   or
$ tar -xjvf package-name.bz2
$ cd package-name   specify directory        specify compilers etc.
$./configure --prefix=$HOME/somedir [CC=icc CXX=icpc ... ]
$ make
$ make install
```

1 Local builds: life without a package manager

General procedure

1. Download source code package
2. Install locally

```
$ tar -xzvf package-name.tgz   or
$ tar -xjvf package-name.bz2
$ cd package-name
$./configure --prefix=$HOME/somedir [CC=icc CXX=icpc ... ]
$ make
$ make install
```

specify directory

specify compilers etc.

3. Add directory paths to **PATH** and **LD_LIBRARY_PATH** and whatever else, in your **.bashrc**, e.g.

```
export PATH=$PATH:$HOME/somedir/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/somedir/lib
```

SciNet compute • calcul CANADA

1 Local builds: life without a package manager

General procedure

1. Download source code package
2. Install locally

```
$ tar -xzvf package-name.tgz   or
$ tar -xjvf package-name.bz2
$ cd package-name
$./configure --prefix=$HOME/somedir [CC=icc CXX=icpc ... ]
$ make
$ make install
```

specify directory

specify compilers etc.

3. Add directory paths to **PATH** and **LD_LIBRARY_PATH** and whatever else, in your **.bashrc**, e.g.

```
export PATH=$PATH:$HOME/somedir/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/somedir/lib
```

4. Re-source **.bashrc**

```
$ source .bashrc
```

2 Local modules

Use the software module system!

- Load software modules with **`module load`** *name*, like system-wide ones.
- De-clutter your **`.bashrc`**.
- You can install conflicting versions.

2 Local modules

Use the software module system!

- Load software modules with **`module load`** *name*, like system-wide ones.
- De-clutter your **`.bashrc`**.
- You can install conflicting versions.

How?

1. Type

```
$ module load use.own
```

   This creates a directory **`$HOME/privatemodules`** (if it doesn't exist yet), and adds it to the software module path.

2. Write modulefiles, and put them in that directory.
   `(path/)filename` is the module name!
3. Put **`module load use.own`** in your **`.bashrc`**.
4. Use **`module load`** on your local modules.

Modulefiles

Modulefiles are written in tcl. The typically have no extension.

Modulefiles

Modulefiles are written in tcl. The typically have no extension.

Example

```
#%Module1.0
proc ModulesHelp { } {
   puts stderr "This module sets up access to something"
}
module-whatis "sets up access to something"
append-path PATH /home/user/somedir/bin
append-path CPATH /home/user/somedir/include
append-path LIBRARY_PATH /home/user/somedir/lib
append-path LD_LIBRARY_PATH /home/user/somedir/lib
```

Modulefiles

Modulefiles are written in tcl. The typically have no extension.

Example

```
#%Module1.0
proc ModulesHelp { } {
    puts stderr "This module sets up access to something"
}
module-whatis "sets up access to something"
append-path PATH /home/user/somedir/bin
append-path CPATH /home/user/somedir/include
append-path LIBRARY_PATH /home/user/somedir/lib
append-path LD_LIBRARY_PATH /home/user/somedir/lib
```

Explanation by line:

1 identifies this as a module file.

Modulefiles

Modulefiles are written in tcl. The typically have no extension.

Example

```
#%Module1.0
proc ModulesHelp { } {
    puts stderr "This module sets up access to something"
}
module-whatis "sets up access to something"
append-path PATH /home/user/somedir/bin
append-path CPATH /home/user/somedir/include
append-path LIBRARY_PATH /home/user/somedir/lib
append-path LD_LIBRARY_PATH /home/user/somedir/lib
```

Explanation by line:

1 identifies this as a module file.

2-4 is a function definition so that "**module help** *somemodule*" works.

Modulefiles

Modulefiles are written in tcl. The typically have no extension.

Example

```
1 #%Module1.0
2 proc ModulesHelp { } {
3    puts stderr "This module sets up access to something"
4 }
5 module-whatis "sets up access to something"
  append-path PATH /home/user/somedir/bin
  append-path CPATH /home/user/somedir/include
  append-path LIBRARY_PATH /home/user/somedir/lib
  append-path LD_LIBRARY_PATH /home/user/somedir/lib
```

Explanation by line:

1 identifies this as a module file.

2-4 is a function definition so that "**module help** *somemodule*" works.

5 makes "**module whatis** *somemodule*" work.

Modulefiles

Modulefiles are written in tcl. The typically have no extension.

Example

```
1 #%Module1.0
2 proc ModulesHelp { } {
3    puts stderr "This module sets up access to something"
4 }
5 module-whatis "sets up access to something"
6 append-path PATH /home/user/somedir/bin
7 append-path CPATH /home/user/somedir/include
8 append-path LIBRARY_PATH /home/user/somedir/lib
9 append-path LD_LIBRARY_PATH /home/user/somedir/lib
```

Explanation by line:

1 identifies this as a module file.

2-4 is a function definition so that "**module help** *somemodule*" works.

5 makes "**module whatis** *somemodule*" work.

6-9 is what really matters: adds directories to the paths.

Module notes

- Can express conflicts and prerequisites.
- Can load other modules.
- Can use general tcl language.
- Does not load paths more than once: safer re-sourcing of `.bashrc`
- More info by typing:

```
$ man module
$ man modulefile
```

3 Python

. . . a programing language that continues to grow in popularity for scientific computing. Very fast to write code in, but much slower than C or Fortran!

3 Python

. . . a programing language that continues to grow in popularity for scientific computing. Very fast to write code in, but much slower than C or Fortran!

Two versions on the GPC

The default is version 2.6.2

```
module load gcc python/2.6.2
```

The currently newest, version 2.7.1, has to be specified explicitly

```
module load gcc python/2.7.1
```

to be put in your `.bashrc`

3 Python

. . . a programing language that continues to grow in popularity for scientific computing. Very fast to write code in, but much slower than C or Fortran!

Two versions on the GPC

The default is version 2.6.2

```
module load gcc python/2.6.2
```

The currently newest, version 2.7.1, has to be specified explicitly

```
module load gcc python/2.7.1
```

to be put in your `.bashrc`

System-wide Python packages

- NumPy
- SciPy
- MPI4Py
- IPython
- Cython
- setuptools
- matplotlib
- nose
- Mercurial
- PyLab
- PyTables
- PySVN
- NumExpr

These ones work for 2.6 & 2.7. Some require other modules, see wiki.

Installing local Python packages

1. create a directory for the packages

```
$ mkdir -p $HOME/lib/python2.X/site-packages
```

Installing local Python packages

1. create a directory for the packages

```
$ mkdir -p $HOME/lib/python2.X/site-packages
```

2. on the command-line and in `.bashrc`:

```
export PYTHONPATH=$PYTHONPATH:$HOME/lib/python2.X/site-packages/
```

(in `.bashrc`, *after* the `module load python`)

SciNet compute • calcul CANADA

Installing local Python packages

1. create a directory for the packages

```
$ mkdir -p $HOME/lib/python2.X/site-packages
```

2. on the command-line and in `.bashrc`:

```
export PYTHONPATH=$PYTHONPATH:$HOME/lib/python2.X/site-packages/
```

   (in `.bashrc`, *after* the `module load python`)

3. a) standard Python package with `easy_install`

```
$ easy_install --prefix=$HOME -O1 package-name
```

SciNet compute • calcul CANADA

Installing local Python packages

1. create a directory for the packages

```
$ mkdir -p $HOME/lib/python2.X/site-packages
```

2. on the command-line and in `.bashrc`:

```
export PYTHONPATH=$PYTHONPATH:$HOME/lib/python2.X/site-packages/
```

   (in `.bashrc`, *after* the `module load python`)

3. a) standard Python package with `easy_install`

```
$ easy_install --prefix=$HOME -O1 package-name
```

   b) packages with a `setup.py` (after downloading them)

```
$ tar -xzvf package-name.tgz   or
$ tar -xjvf package-name.bz2
$ python setup.py install --prefix=$HOME
```

SciNet compute • calcul CANADA

Installing local Python packages

1. create a directory for the packages

```
$ mkdir -p $HOME/lib/python2.X/site-packages
```

2. on the command-line and in `.bashrc`:

```
export PYTHONPATH=$PYTHONPATH:$HOME/lib/python2.X/site-packages/
```

   (in `.bashrc`, *after* the `module load python`)

3. a) standard Python package with `easy_install`

```
$ easy_install --prefix=$HOME -O1 package-name
```

   b) packages with a `setup.py` (after downloading them)

```
$ tar -xzvf package-name.tgz   or
$ tar -xjvf package-name.bz2
$ python setup.py install --prefix=$HOME
```

4. add each `.egg` directory to your Python path in your `.bashrc`.

Example: Brian

```
$ module load gcc python/2.7.1 intel/intel-v12.0.0.084
$ mkdir -p $HOME/lib/python2.7/site-packages
$ export PYTHONPATH=$PYTHONPATH:$HOME/lib/python2.7/site-packages/
$ easy_install --prefix=$HOME -O1 SymPy
$ easy_install --prefix=$HOME -O1 Brian
```

Example: Brian

```
$ module load gcc python/2.7.1 intel/intel-v12.0.0.084
$ mkdir -p $HOME/lib/python2.7/site-packages
$ export PYTHONPATH=$PYTHONPATH:$HOME/lib/python2.7/site-packages/
$ easy_install --prefix=$HOME -O1 SymPy
$ easy_install --prefix=$HOME -O1 Brian
```

```
#%Module1.0
proc ModulesHelp { } {
    puts stderr "\tThis module adds Brian 1.3.0 env variables"
}
module-whatis "adds Brian 1.3.0 environment variables"
# Python 2.7.1 needed, compiled with gcc but needs intel MKL
prereq python/2.7.1
prereq intel/intel-v12.0.0.084
set basedir $::env(HOME)/lib/python2.7
append-path PYTHONPATH $basedir/site-packages
append-path PYTHONPATH $basedir/site-packages/sympy-0.6.7-py2.7.egg
append-path PYTHONPATH $basedir/site-packages/brian-1.3.0-py2.7.egg
```

4 Perl

. . . is a high-level, general-purpose, interpreted, dynamic programming language.

4 Perl

. . . is a high-level, general-purpose, interpreted, dynamic programming language.

Comprehensive Perl Archive Network (CPAN)

- is an archive of over 20,000 modules of Perl software.
- Installing user-space modules poses a slight hurdle, as CPAN's package system would try to install these in system directories.
- Fortunately, there is a way to setup CPAN to use a local folder.
- In the following, the local Perl directory will be assumed to be `~/perl5`.

Local Perl modules

Modifications to `.bashrc`

```
export FTP_PASSIVE=1
export PATH=$HOME/perl5/bin:$PATH
export PERL_LOCAL_LIB_ROOT=$HOME/perl5
export PERL_MB_OPT="--install_base $HOME/perl5"
export PERL_MM_OPT=$HOME/perl5
export INSTALL_BASE=$HOME/perl5
export PERL5LIB=$HOME/perl5/lib/perl5/x86_64-linux-thread-multi
export PERL5LIB=$PERL5LIB:$HOME/perl5/lib/perl5
```

Re-source `.bashrc`.

Excellent candidate for your own module!

Perl continued...

Setup a user-space CPAN modules directory

1. Since installing Perl modules may involve compilation, be sure to be on a devel node, and make sure that the gcc module is loaded.
2. Not your first time using CPAN? May have to **`rm -r`** your **`~/.cpan`** !
3. Start a CPAN shell with

```
$ cpan
```

   Confirm that you are ready for a manual configuration.

4. It will now ask you a bunch of questions, almost all of which you can answer with the default.
5. When asked for **`Parameters for the 'perl Makefile.PL' command`**, enter:

```
PREFIX=~/perl5/ LIB=~/perl5/lib/perl5
INSTALLMAN1DIR=~/perl5/man/man1 INSTALLMAN3DIR=~/perl5/man/man3
```

   (all on one line)

Perl continued...

`local::lib` module

Perl module **`local::lib`** is required to use those **`.bashrc`** environment vars.
To install the latest **`local::lib`**:

1. Type **`install local::lib`** in the CPAN shell and press enter.
2. As the Perl module is installing, numerous prerequisites will also be installed (including an updated version of CPAN).
3. Confirm these when asked.
4. Answer each with the default option.
5. Everything should install without errors, although there'll be some conplaints.

SciNet compute • calcul CANADA

Perl continued...

Installing CPAN modules

Either from the CPAN shell or from the command line, you can now install packages by

```
perl -MCPAN -e install perlmodule
```

or

```
cpan perlmodule
```

You can now use the local Perl modules just as you would use system-wide Perl modules.

5 R

. . . is a powerful statistical and plotting software.
It is available on the GPC in the module `R`.

5 R

. . . is a powerful statistical and plotting software.
It is available on the GPC in the module **R**.

Comprehensive R Archive Network (CRAN)

- Provides optional packages for R for specific domains.
- R provides a default way for users to install the libraries they need in their home directories.
- You can install those that you need yourself in your home directory:

```
$ R
> install.packages("package-name", dependencies = TRUE)
```

  will download and compile the source for the packages you need in your home directory under

  $HOME/R/x86_64-unknown-linux-gnu-library/2.11

- Some R modules, such as Rmpi, are a bit trickier; see wiki.

Final thoughts

- Before installing, first check if the module is not already available.
- You've only got 10GB quota in your **$HOME**.
- Check quotas with **/scinet/gpc/bin/diskUsage**.
- For local builds, can keep source material in **/scratch** or **/dev/shm**.
- Email support@scinet if you need help.

SciNet compute • calcul CANADA

Final thoughts

- Before installing, first check if the module is not already available.
- You've only got 10GB quota in your **$HOME**.
- Check quotas with **/scinet/gpc/bin/diskUsage**.
- For local builds, can keep source material in **/scratch** or **/dev/shm**.
- Email support@scinet if you need help.

Wiki: https://support.scinet.utoronto.ca/wiki/index.php/X

Topic	X
Using modules	Software_and_Libraries
Writing modules	Installing_your_own_modules
Python	Python
Perl	Perl
R (including Rmpi)	R_Statistical_Package
Brian	Brian

Or just use the search box.