How to Install Software on Abel (and Colossus)

Bjørn-Helge Mevik

Department for Research Infrastructure Services, USIT, UiO

RIS Course Week

General Points

- We encourage you to install software yourself, especially `R`, Perl and Python packages.
- Make sure you have loaded "the modules you need, all the modules you need, and nothing but the modules you need":

```
$ module purge
$ module load foo bar
$ module list
```

- On **Abel**, try to use the Intel compilers when compiling. That is the default when installing `R` and Python packages:

```
$ module load intel/2019.1
$ module list
```

- On **Colossus**, the default is to use the Gnu compilers, so no compiler module needed.

R Packages

- Main repository: https://cran.r-project.org/
- Our documentation: https://www.uio.no/english/services/it/research/hpc/abel/help/software/R.html
- Tip: To avoid having to select repository every time: Add

```
local({
    r <- getOption("repos")
    r["CRAN"] <- "https://cran.uib.no"
    options(repos = r)
})
```

to your ~/.Rprofile file.

CRAN Packages

https://cran.r-project.org/

```
$ module purge
$ module load R/3.5.0
$ module list
Currently Loaded Modulefiles:
  1) intel/2017.2   3) zlib/1.2.8     5) xz/5.2.2        7) R/3.5.0
  2) curl/7.46.0    4) bzip2/1.0.6    6) pcre/8.38
$ R
[...]
> install.packages("pls")
[...]
* DONE (pls)

The downloaded source packages are in
'/tmp/RtmpydUdGZ/downloaded_packages'
>
```

See ?install.packages for details.

Packages with Dependencies

Some packages depend on libraries etc. outside R.

```
> install.packages("rjags")
[...]
checking for pkg-config... /usr/bin/pkg-config
configure: WARNING: pkg-config file for jags 4 unavailable
configure: WARNING: Consider adding the directory containing ‘jags.p
configure: WARNING: to the PKG_CONFIG_PATH environment variable
configure: Attempting legacy configuration of rjags
checking for jags... no
configure: error: "automatic detection of JAGS failed. Please use pk
ERROR: configuration failed for package ‘rjags’
* removing ‘/cluster/home/bhm/nobackup/R/x86_64-pc-linux-gnu-library

The downloaded source packages are in
‘/tmp/Rtmpg9GS7o/downloaded_packages’
Warning message:
In install.packages("rjags") :
  installation of package ‘rjags’ had non-zero exit status
```

Packages with Dependencies (II)

```
$ module avail jags
[...]
jags/3.3.0                    jags/4.2.0_2015_3(default)
jags/4.0.0                    jags/4.3.0
jags/4.2.0
$ module load jags/4.2.0_2015_3
$ module list
$ R
[...]
> install.packages("rjags")
[...]
* DONE (rjags)
[...]
```

Remember to load the modules before using the package later!

Other Packages

- Bioconductor packages (http://bioconductor.org/install/)

```
> source("http://bioconductor.org/biocLite.R")
> biocLite(c("package"))
```

- Local files

```
$ wget \\
  https://cran.r-project.org/src/contrib/abind_1.4-5.tar.gz
$ R CMD INSTALL abind_1.4-5.tar.gz
```

See `R CMD INSTALL --help` for details.

Advanced Options

- Install to a different location:
  `R CMD INSTALL --library=/the/path` or
  `install.packages(..., lib = "/the/path")`.
  `.libPaths("/the/path")` to use location.

- **Abel:** Some packages will not compile with Intel. Then use an `R` built with Gnu compilers:

```
$ module purge
$ module load R/3.5.0.gnu
$ module list
```

  Usually, the package can later be *used* with the standard `R`.

- **Colossus** (on your TSD LinuxVM):

```
> install.packages("package",
                   repos = "file://tsd/shared/R/cran/")
```

Uninstalling Packages

- Inside R:
  ```
  > remove.packages("abind")
  ```
- Command line:
  ```
  $ R CMD REMOVE pls
  ```
- Manually:
  ```
  $ rm -rf ~/R/x86_64-pc-linux-gnu-library/3.5/rjags
  ```

Paths work as for install. See `R CMD REMOVE --help` or `?remove.packages` for details.

Perl Packages

- Main repository: http://www.cpan.org/
- Our documentation: https://www.uio.no/english/services/it/research/hpc/abel/help/software/Perl_modules.html
- Setting up directories and environment variables:

```
$ module load perlmodules  # local::lib and cpanm
$ eval $(perl -Mlocal::lib)
$ export PERL_CPANM_HOME=/tmp/cpanm_$USER
```

- The above can be added to ~/.bashrc or ~/.bash_profile.

CPAN Packages

```
$ cpanm JSON
--> Working on JSON
Fetching http://www.cpan.org/authors/id/I/IS/ISHIGAKI/JSON-4.02
Configuring JSON-4.02 ... OK
Building and testing JSON-4.02 ... OK
Successfully installed JSON-4.02
1 distribution installed
$ perl -MJSON -e 'print "ok\n"'
ok
```

Advanced Options

- Install from local file:
  ```
  $ wget \
  http://www.cpan.org/authors/id/I/IS/ISHIGAKI/JSON-4.02.tar.gz
  $ cpanm JSON-4.02.tar.gz
  ```
- Install to a separate directory:
  ```
  eval $(perl -Mlocal::lib=/the/path)
  cpanm Some::Module
  ```
  (Good for keeping projects separate.)
- It is possible to avoid `module load perlmodules` every time. See https://www.uio.no/english/services/it/research/hpc/abel/help/software/Perl_modules.html
- **Colossus:** Currently no CPAN mirror inside TSD, so packes must be imported and installed from local file. *Hopefully fixed soon!*

Uninstalling Packages

- `cpanm --uninstall package`. Note: experimental!
- If too old, try `cpanm --self-upgrade; hash -r` first.
- Paths work the same as when installing.
- Example: `cpanm --uninstall JSON`.

Python Packages

- Main repository: https://pypi.python.org/
- Our documentation:
  - Python 2: https://www.uio.no/english/services/it/research/hpc/abel/help/software/Python%202.html
  - Python 3: https://www.uio.no/english/services/it/research/hpc/abel/help/software/Python%203.html

Python 2

```
$ module purge
$ module load python2/2.7.10
$ module list
Currently Loaded Modulefiles:
  1) intel/2015.3     2) libffi/3.0.13     3) python2/2.7.10
$ pip install --user dexml
Collecting dexml
  Downloading dexml-0.5.1.tar.gz
Installing collected packages: dexml
  Running setup.py install for dexml ... done
Successfully installed dexml-0.5.1
$
```

See `pip --help` and `pip install --help` for details.

Python 3

```
$ module purge
$ module load python3/3.5.0
$ module list
Currently Loaded Modulefiles:
  1) intel/2016.0    2) libffi/3.0.13    3) python3/3.5.0
$ pip3 install --user dexml
Collecting dexml
    Using cached https://[...]/dexml-0.5.1.tar.gz
Installing collected packages: dexml
  Running setup.py install for dexml
Successfully installed dexml
$
```

See `pip3 --help` and `pip3 install --help` for details.

Packages with Dependencies

```
$ module purge
$ module load python2/2.7.10
$ module list
$ pip install --user mpi4py
[...]
error: Cannot compile MPI programs. Check your configuration!!!
[...]
$ module load openmpi.intel/3.1.2
$ pip install --user mpi4py
[...]
Successfully installed mpi4py-3.0.0
```

Remember to load the modules before using the package later!

Advanced Options

- Install from local file:
```
wget \
https://pypi.python.org/packages/source/d/dexml/dexml-0.5.1.tar
pip install --user dexml-0.5.1.tar.gz
```
- Install in separate directory (good for separating projects):
```
pip install --target=/the/path thepackage
PYTHONPATH=/the/path:$PYTHONPATH
```
- **Colossus** (on your TSD Linux VM):
```
pip install --user \
  --index-url=file:///shared/pypi/mirror/web/simple package
```

(Or `pip3`.)

Uninstalling Packages

- `pip uninstall package` (or `pip3`).
- Paths work the same as for installing.
- Example: `pip uninstall dexml`

General Software Packages

- You can install software in your home directory, or in a project area you have access to.
- We recommend installing from source if possible.
- If a package depends on some software or library that is installed on Abel, use `module load` before installing it, and before using it later.

Different Types of Installs

- How to install depends on what the developers have decided. *Read the documentation!* There is no alternative.
- Some common types of installs:
  - GNU Autoconf based
  - Cmake based
  - Manually written Makefile
  - Binary install in Tar og Zip file
- Note: Ubuntu/Debian binary packages cannot be installed, since Abel and Colossus run RedHat (CentOS). (So *no* `apt-get` or `aptitude`.)
- It *might* be possible to install some RPMs, but it is tricky.

GNU Autoconf Based Install

In general:

- `./configure --prefix=/some/dir`
- `make`
- `make install`

See `./configure --help` for more options

Autoconf Example: cfitsio

```
$ module purge
$ module load intel/2018.1
$ module list
$ wget \
ftp://heasarc.gsfc.nasa.gov/software/fitsio/c/\
cfitsio3370.tar.gz
$ tar xf cfitsio3370.tar.gz
$ cd cfitsio
$ less README
$ ./configure --prefix=$HOME/lib/cfitsio
$ make
$ make install
```

Cmake

Typical usage:

- `mkdir build`
- `cd build`
- `cmake -DCMAKE_INSTALL_PREFIX=/some/dir ..`
- `make`
- `make install`

Cmake Example: bamm

```
$ wget https://github.com/macroevolution/bamm/archive/v2.5.0.tar.gz
$ tar xf v2.5.0.tar.gz
$ cd bamm-2.5.0
$ less README.md
$ module purge
$ module load intel/2018.1 cmake/3.7.1
$ module list
$ mkdir build
$ cd build
$ cmake -DCMAKE_INSTALL_PREFIX=$HOME/bamm ..
$ make -j
[...]
/usr/include/c++/4.4.7/c++0x_warning.h(31): error: #error directive:
This file requires compiler and library support for the upcoming
ISO C++ standard, C++0x. [...]
```

Cmake Example: bamm, take 2

Let's try with gcc instead:

```
$ cd ..
$ rm -rf build
$ module purge
$ module load gcc/7.2.0 cmake/3.7.1
$ module list
$ mkdir build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX=$HOME/bamm ..
$ make -j
[...]
[100%] Built target bamm
$ make install
[...]
-- Installing: /usit/abel/u1/bhm/bamm/bin/bamm

$ ~/bamm/bin/bamm
Usage: bamm -c <control-file> [--<parameter-name> <parameter-value>
```

Manual Makefile

- The documentation (`README`, `INSTALL`, etc.) should give you the information needed.
- Well written Makefiles use a variable to control the installation directory. Then you can use `make` *`VARIABLE`*`=/the/path install` to build and install.
- If the Makefile uses an environment variable, you can set it with `export` *`VARIABLE`*`=/the/path` before running `make`.
- If the installation directory is hard coded in the Makefile, you must edit the file.

Manual Makefile Example: subread

```
$ wget http://downloads.sourceforge.net/project/subread/\
subread-1.5.0-p1/subread-1.5.0-p1-source.tar.gz
$ tar xf subread-1.5.0-p1-source.tar.gz
$ cd subread-1.5.0-p1-source/; ls; less README.txt
$ cd src; less Makefile.Linux
$ module purge
$ module load gcc/7.2.0; module list
$ make -f Makefile.Linux
[...]
#  Installation finished.                                    #
#  Generated executables were copied to directory ../bin/ #
[...]
$ # optional: mv ../bin ~/<somewhere>

$ ../bin/exactSNP
Version 1.5.0-p1
Usage:
[...]
```

Advanced Options for Makefile Based Packages

- Some packages have a `make check` to test the build
- Some packages can be run from the build area without using `make install`
- Some packages can be uninstalled with `make uninstall`. See the generated `Makefile`. Otherwise: `rm -rf`
- Compiler options can be passed as environment variables (`export CFLAGS=-O3` before `./configure` or `cmake`), or directly to `make`: `make CFLAGS=-O3`. *But be careful: they will* ***replace*** *settings in* `Makefile`*, not add to them.*

Optimising Compiled Code on Abel

These options are good on **Abel**, but the `-march` depends on the CPU type, so **don't** use this blindly on other machines.

- Intel
  - `-O3 -march=sandybridge`
  - Vectorized math library (`libm`) where applicable: `-fp-model fast=1`. *Note: Less strict IEEE FP accuracy!*
  - Information about AVX vectorization of the code: `-qopt-report-phase=vec -qopt-report=5`
  - See `icc -help` for details and explanations.
- Gnu
  - `-O3 -march=sandybridge`
  - Vectorized math library (`libm`) where applicable: `-ffast-math`. *Note: Less strict IEEE FP accuracy!*
  - Information about AVX vectorization of the code: `-fopt-info-vec`
  - See `man gcc` for details and explanations.

Binary Installs

- Usually Tar or Zip files.
- Often, you only need to unpack the files. Optionally, you can rename the directory, or copy the extracted files to the desired directory.
- Tip: First check whether the file unpacks in a subdirectory or into the current directory (`tar tvf file.tar.gz` or `unzip -t file.zip`).
- There is a risk that the programs need libraries that are missing, too old or too new on the cluster. Then you need to install these as well, or switch to installing from source, if possible.
- To uninstall, simply remove the directory.

Creating Module Files

It is possible to create your own collection of module files. For instance:

```
$ mkdir -p $HOME/etc/modulefiles
# Tell module to use your files:
$ module use --append $HOME/etc/modulefiles
$ mkdir -p $HOME/etc/modulefiles/bamm    # A dir per package
$ emacs $HOME/etc/modulefiles/bamm/2.5.0 # A file per version
# Check and use:
$ module avail bamm
$ module load bamm/2.5.0
$ module list
$ bamm
```

The `module use` command can be put in `.bashrc` or similar.

The Bamm Modulefile

```
#%Module1.0#####################################################
#
set apphome    /usit/abel/u1/bhm/bamm

## Load any needed modules:
module load gcc/7.2.0

## Modify as needed, removing any variables not needed.  Non-path
## variables can be set with "setenv VARIABLE value".
prepend-path    PATH            $apphome/bin
#prepend-path    LD_LIBRARY_PATH $apphome/lib64/R/lib
#prepend-path    LIBRARY_PATH    $apphome/lib64/R/lib
#prepend-path    MANPATH         $apphome/share/man
#prepend-path    CPATH           $apphome/lib64/R/include
#prepend-path    PKG_CONFIG_PATH $apphome/lib64/pkgconfig
```

That's all, folks!

The End