Lmod is a tool for managing software packages within a shared high-performance computing environment (e.g. the ACCRE cluster). If you started using ACCRE in Winter 2017 or earlier, you can think of Lmod as a replacement for `setpkgs` and `pkginfo`. Lmod is designed to intelligently manage, negotiate, and enforce the complex dependencies between software packages and libraries in a HPC environment. This should lead to better usability of administrator-installed software packages and prevent conflicts between lower-level libraries that many packages depend on. **Starting in Spring 2017, Lmod will be available for testing and general use in parallel with `setpkgs` and `pkginfo`. As the ACCRE operating system is upgraded to CentOS 7 in Summer and Fall 2017, only Lmod will be available in the new CentOS 7 environment.**

1 Using Lmod

For instructions on basic Lmod usage, please start with the excellent and concise User Guide for Lmod.

A detailed overview of using Lmod on the ACCRE cluster is also available here: Lmod Slide Deck

There are a handful of things you must get accustomed to when transitioning from `setpkgs/pkginfo` to Lmod:

1.1 Searching for available packages

When searching for available packages, Lmod will only show you those packages that were built with a lower-level dependency (most often a compiler) that is already loaded. This means that the first time you run `module avail` (the equivalent of pkginfo), you will only see a handful of packages available. However, as you load packages via Lmod, additional packages will be listed when you run `module avail` again. For example:

```
$ module avail

----------------------- Core Modules -----------------------
   Anaconda2/4.3.1    IMPUTE2/2.3.2_x86_64_static
   Anaconda3/4.3.1    Intel/2016.3.210-GCC-5.4.0-2.26
   GCC/5.4.0-2.26     Java/1.8.0_92

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all
possible modules matching any of the "keys".
```

```
$ module load GCC/5.4.0-2.26
$ module avail

---------- Applications built with GCC 5.4.0-2.26 ----------
   GSL/2.1
   OpenBLAS/0.2.18-LAPACK-3.6.1
   OpenMPI/1.10.3
   Perl/5.24.0
   Ruby/2.3.1
   SQLite/3.13.0
   Tcl/8.6.5
   VCFtools/0.1.14-Perl-5.24.0

---------- Applications built with GCCcore 5.4.0 -----------
   CMake/3.5.2

----------------------- Core Modules -----------------------
   Anaconda2/4.3.1        IMPUTE2/2.3.2_x86_64_static
   Anaconda3/4.3.1        Intel/2016.3.210-GCC-5.4.0-2.26
   GCC/5.4.0-2.26  (L)    Java/1.8.0_92

  Where:
   L:  Module is loaded

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all
possible modules matching any of the "keys".
```

Note that additional packages are now listed under the **Applications built with GCC 5.4.0-2.26** section, and can now be loaded with the `module load` command.

1.2 Searching for all installed packages

Because not all packages are shown by default, you should familiarize yourself with the `module spider` and `module keyword` commands in order to check for installed packages. For example, if you want to run R on the cluster:

```
$ module spider R
```

```
---------------------------------------------------------
  R:
---------------------------------------------------------
    Description:
      R is a free software environment for statistical
      computing and graphics. - Homepage:
      http://www.r-project.org/

     Versions:
        R/3.2.3

     Other possible modules matches:
        GCCcore  PROJ  Perl  Ruby  Valgrind  cURL  ...

---------------------------------------------------------
  To find other possible module matches do:
      module -r spider '.*R.*'

---------------------------------------------------------
  For detailed information about a specific "R" module (including
  how to load the modules) use the module's full name.
  For example:

     $ module spider R/3.2.3
---------------------------------------------------------
```

Lmod provides a helpful suggestion for the full name of the package, which must include the version, so let's try that:

```
$ module spider R/3.2.3

---------------------------------------------------------
  R: R/3.2.3
---------------------------------------------------------
    Description:
      R is a free software environment for statistical
      computing and graphics. - Homepage:
      http://www.r-project.org/

     Other possible modules matches:
        GCCcore, PROJ, Perl, Ruby, Valgrind, ...

    You will need to load all module(s) on any one of the
    lines below before the "R/3.2.3" module is available to load.

      GCC/5.4.0-2.26  OpenMPI/1.10.3
      Intel/2016.3.210-GCC-5.4.0-2.26  IntelMPI/5.1.3.181
    Help:
      R is a free software environment for statistical
     computing and graphics. - Homepage: http://www.r-project.org/

--------------------------------------------------------
  To find other possible module matches do:
      module -r spider '.*R/3.2.3.*'
```

Excellent! Lmod gives us the exact commands to run before we can load the R/3.2.3 package. Notice you have two options: the GCC-compiled version, or the Intel-compiled version. Either is fine. Intel-compiled packages tend to run a bit faster on our Intel processors, however GCC-compiled software tends to be better supported and more widely tested. In this case, let's load the GCC-compiled version of R 3.2.3. In order to do this, simply type:

```
$ module load GCC/5.4.0-2.26 OpenMPI/1.10.3 R/3.2.3
```

Or (see point 4 below for details on default versions of packages):

```
$ module load GCC OpenMPI R
```

To verify that you have the correct version of R loaded type:

```
$ R --version
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
http://www.gnu.org/licenses/.
```

The `module keyword` command is also useful for getting information about available packages. For instance, if you want to see information about MPI versions available on the cluster:

```
$ module keyword MPI

--------------------------------------------------------
The following modules match your search criteria: "MPI"
--------------------------------------------------------

  GCC: GCC/5.4.0-2.26
    The GNU Compiler Collection includes front ends for
    C, C++, Objective-C, Fortran, Java, and Ada, as
    well as libraries for these languages (libstdc++,
    libgcj,...). - Homepage: http://gcc.gnu.org/
  Intel: Intel/2016.3.210-GCC-5.4.0-2.26
    Intel Cluster Toolkit Compiler Edition provides
    Intel C,C++ and fortran compilers, Intel MPI and
    Intel MKL - Homepage:
   http://software.intel.com/en-us/intel-cluster-toolkit-compiler/

  IntelMPI: IntelMPI/5.1.3.181
    The Intel(R) MPI Library for Linux* OS is a
    multi-fabric message passing library based on ANL
    MPICH2 and OSU MVAPICH2. The Intel MPI Library for
    Linux OS implements the Message Passing Interface,
    version 2 (MPI-2) specification. - Homepage:
    http://software.intel.com/en-us/intel-mpi-library/

  OpenMPI: OpenMPI/1.10.3
    The Open MPI Project is an open source MPI-2
    implementation. - Homepage:
    http://www.open-mpi.org/

--------------------------------------------------------
To learn more about a package enter:

   $ module spider Foo

where "Foo" is the name of a module

To find detailed information about a particular package you
must enter the version if there is more than one version:

   $ module spider Foo/11.1
--------------------------------------------------------
```

The `module keyword` command ignores case and also supports regular expressions for filtering your searches (the same is true of the `module spider` command).

1.3 Autocompletion

Lmod supports tab autocompletion, so in order to load a package you can type `module load` followed by the first few letters of a package name and the <tab> key.

1.4 Version precedence

In the event that multiple versions of a software package are available, the default version will be noted with a (D). If you exclude the version information (e.g. `module load GCC`), the default version (or the only version if only a single version exists) of the software will be loaded.

1.5 List loaded packages

Use `module list` to see what packages are currently loaded in your environment:

```
# module list
No modules loaded

$ module load GCC
$ module list

Currently Loaded Modules:
  1) GCCcore/.5.4.0   3) GCC/5.4.0-2.26
  2) binutils/.2.26
```

Notice that more packages than just the GCC module were loaded. This is not unusual, so if you care about the low-level dependencies of your program it's a good idea to run `module list` to check.

1.6 Unloading specific packages

Use `module unload` to remove a package from your environment. For example:

```
$ module load GCC/5.4.0-2.26
$ module list

Currently Loaded Modules:
  1) GCCcore/.5.4.0   2) binutils/.2.26   3) GCC/5.4.0-2.26

$ module unload GCC/5.4.0-2.26
$ module list
No modules loaded
```

Occasionally (although not in the above example), a package's dependencies will not be removed from your environment after unloading the package via `module unload`. However, Lmod will remove those dependencies if they cause conflicts in a subsequent `module load` command.

1.7 Unloading all packages

Use `module purge` to remove all packages from your environment. For example:

```
$ module load GCC/5.4.0-2.26
$ module load OpenMPI/1.10.3
$ module list

Currently Loaded Modules:
  1) GCCcore/.5.4.0   3) GCC/5.4.0-2.26    5) hwloc/.1.11.3
  2) binutils/.2.26   4) numactl/.2.0.11   6) OpenMPI/1.10.3

$ module purge
$ module list
No modules loaded
```