Environment Modules

Environment modules are used on the LRZ HPC systems to manage the environment variables in shells and scripting languages. They also permit to provide multiple versions of packages.

- Introduction
- Module command options
  - module list
  - module avail
  - module help
  - module show
  - module load
  - module unload
  - module search
  - module switch
  - module purge
  - module use
  - module path
- Supported shell and scripting languages
  - Shells
  - Using modules from Perl or Python
- Specifics on the setup for LRZ HPC systems
  - Module classes
  - Generating new shells
  - Interactive shells
  - Using modules in batch jobs
  - Additional utilities
  - Versioning policies
- Initialization and writing your own module files
- Naming conventions
- Support and sources of information on modules

Introduction

Modules provide an easy mechanism for managing the user’s environment (PATH, MANPATH, and LD_LIBRARY_PATH, to name a few).

To make the application-specific settings for <package> (the denotation is abstract) available in the current shell, the user types the command

```
module load <package>
```

If, subsequently, the user types

```
module unload <package>
```

the application-specific settings for <package> will be removed again, and the application typically will become inaccessible or unusable.

On the LRZ HPC systems, setting up of the modules package is performed **automatically at login**. This includes

- making the module command (as described below) available
- loading a default set of module files. The exact set depends on the platform used.

Module command options

This section supplies an overview of the most important command options. Note that the module command is defined as an alias.

**module list**

This lists all the modules which are currently loaded into the users’ environment, in the order in which they were loaded. With an additional -t option, only a single column is printed.

**module avail**

This option lists all the modules which are available to be loaded. Notice that most have version numbers associated with them. One of the versions is marked out as default; this is the version that is referred to when the version number is omitted in a module command. Adding the -d switch will only list the default versions of each module. Furthermore, limited pattern matching will also work, so
module avail gs

will print the list of available modules that start with the letters "gs". The -t switch will display the modules as a single column.

module help

This option requires a module name (with or without version number) as an argument. It will display additional information for the specified module; at minimum, a pointer to web documentation for the package should be delivered.

module show

This option requires a module name (with or without version number) as an argument. It will display the changes to the environment that would be done if the module were loaded.

module load

This option requires one or more (space separated) module names (with or without version number) as an argument. For example,

```
module load gaussian
```

will make the default version of the package "gaussian" available for your use. If you need a specific supplied version that is non-default, you need to specify the version number explicitly, for example:

```
module load gaussian/09.D.02
```

Please also note that

- some modules may automatically load others if a package depends on some other package to be available. Unloading dependent modules may give you trouble!
- certain default modules should also not be unloaded, since your environment will not function properly without the settings performed there
- if you specify multiple modules, they will be loaded in the supplied order
- instead of "load", you can also say "add".

module unload

This option requires one or more (space separated) module names (with or without version number) as an argument. It will unload the specified modules, removing all previously supplied environment settings. Please note that

- for multiple modules to be removed, it is recommended to do this in the reverse order of loading, especially if there exist dependencies between two such modules
- it is not necessary to specify a version number - the loaded version will be removed even if not explicitly specified
- instead of "unload", you can also say "rm".

module search

This option requires a text pattern as an argument. Each module contains a module-whatis entry with a keyword list that will be searched for the provided pattern. All modules that contain a matching entry will then be listed. In particular, the module classes (with a colon appended) are always used as a keyword, so this facility can be used to search the module system by application classes:

```
module search libraries:
```

You can also say "apropos" or "keyword" instead of "search".

module switch

This option requires exactly two (space separated) module names (with or without version number) as an argument. The command performs a transition from the module specified as the first argument to that specified by the second one with a single command. Note that this is not exactly the same as an unload followed by a load, since all modules which were loaded after the first specified will also be reloaded, to assure that dependencies are correctly resolved. If this causes difficulties, you may need to revert to explicitly perform an unload/load sequence.
module purge

This command removes all modules from your environment. On the LRZ HPC systems, this causes the environment to become rather unusable. Therefore, after a purge, you should issue the command

```
unset LRZ_LOAD_DEFAULTS  # or unsetenv for csh-like shells
module load lrz
```

before loading or unloading any further modules.

module use

This option requires an existing directory name (or multiple directory names) as an argument. It causes its argument to be added to the modules path, with the result that all module files stored within that directory become visible for the `module` command. The entries are all stored in the `MODULESPATH` environment variable. The `-a` option can be used to append the directory to the path instead of prepending it. It is also possible to remove a path again by specifying it as an argument for the `module unuse` command.

module path

This option requires one module names (with or without version number) as an argument. The complete path name of the module file will be printed as a result.

Supported shell and scripting languages

Shells

The following shells will be supported by the module system:

- bash
- csh
- ksh
- sh
- tcsh

Using modules from Perl or Python

It is possible to use the module command also from non-shell scripting languages. In the case of Perl, a LRZ specific package is available. Here is an example on how to use this:

```perl
#!/usr/bin/perl
use lib "/lrz/sys/share/modules/init";
use Environment_Modules;
module("load","intel mkl");
module("list");
system("make myprog");
system("./dosomething");
```

Note the following details:

- Use standard perl function call syntax with brackets, especially when specifying more than one argument
- In contrast to the shell setup, only the LRZ administrative modules are loaded. Please load all other required modules yourself.
- Within Perl, the environment variables are accessible via the `%ENV` hash.
- All environment variables will be propagated to processes called e.g., via `system`. Hence, you can use these variables in Makefiles and scripts which are executed/called from Perl.

Specifics on the setup for LRZ HPC systems

Module classes

In order to impose some structure on the large number of provided modules, a number of `module classes` has been defined, one of which each module falls into:

- **applications**: third party application programs will be entered here
- **compilers**: programs for converting computer language source code to (executable) machine code
- **environment**: modules for providing LRZ specific environment variables
- **graphics**: modules for providing graphics and visualization related functionality
io_tools: modules for I/O related libraries and tools
libraries: performance and functionality libraries
parallel: access to parallelization software, especially MPI
tools: development, tuning and debugging aids

There is a further, special extfiles category which will be used e.g., for applications with a rapidly changing profile or for testing. However, this class is not made available by default. You will need to activate it by issuing

```bash
module use -a /lrz/sys/share/modules/extfiles
```

Please note that modules in this area are not guaranteed any support; also they may be modified or removed without any prior warning, on short order.

**Generating new shells**

While the environment generated within a shell should be inherited to newly generated ones (either the same shell, or other shells), the module command itself may not be. We recommend inserting the following lines into your shell init script (e.g., .bashrc for bash, or $ENV for ksh):

```bash
typeset -f module > /dev/null
if [ $? != 0 -a -r /etc/profile.d/modules.sh ] ; then
    source /etc/profile.d/modules.sh
fi
```

Note that this will preserve the environment set up in the calling shell.

**Interactive shells**

In interactive shells, the output from the module system is written to the standard output file descriptor beginning with the release update from May 9, 2017. This release also adds support for tab completion when typing module commands in the bash or tcsh shells.

**Using modules in batch jobs**

The modules package is loaded by default at login. However, the typical batch systems are not able to reliably load the modules package. Hence, if you want to use modules within batch scripts, please do not forget to insert the following command as the first line of your script:

```bash
source /etc/profile.d/modules.sh
```

if your are a sh/bash/ksh user, and

```bash
source /etc/profile.d/modules.csh
```
in case your script is csh derived.

**Additional utilities**

Some additional utilities which are provided specifically with the LRZ installation are described in this section.

1. `ismoduleloaded`: if issued with an argument, will tell whether that module is loaded. If the argument is a versioned module name, the exact version will be checked, otherwise only whether any version of the given module is loaded. Output is a "YES" or "NO" to stdout as well as an error exit of 0 or 1, respectively. If the --command line switch is provided, output to stdout is suppressed. If no module argument is given, "YES" will be written to stdout if any module is loaded, "NO" otherwise.
2. `defaultversion`: if issued with a module name as argument (with or without a version number), this will return the default version of the module on STDOUT. Error 1 is returned if the module does not exist.
3. `loadedversion`: if issued with a module name as argument (with or without a version number), this will return the presently loaded version of the module on STDOUT. Error 1 is returned if a module of the given name is not loaded. The version number on the argument need not be the correct or even an existing one.
4. `removemodules.[c]sh`: this can be used to completely remove all module settings from your environment by sourcing the appropriate version into your running shell; all modules are unloaded, and the module alias as well as all module-related variables are unset. Beware that this will rather cripple your environment since many important settings are performed via LRZ administrative modules.

**Versioning policies**

Please note that software packages as well as the module system undergo a maintenance and update process. As a consequence, the following may happen at LRZ HPC staff's discretion and - unless the user impact is considered very large - without special announcement:
the default version of a software package may change; the method of loading the default module will however normally stay the
same.
* the availability and the module name of non-default versions of a software package may change
* a specific module version may become invisible. This means that it will not be listed any more by the module avail command,
  because the name or the version of the module begins with a dot. To see all versions of a module including the hidden ones, please
  use e.g., the bash command

```
ls -a $(dirname $(module path <package>))
```

We apologize for the potential impact on processing of user jobs, but given the large number of packages we make available we presently
see no alternative. For a given software package, please check out its documentation page on the LRZ web server for updates and/or use
the module avail <package_name> command to see the presently available releases.

**Initialization and writing your own module files**

A user can perform her/his own module initializations at login time using a shell-specific script in the $HOME/.modules subdirectory. This
script must have the same name as your login shell, i.e., $HOME/.modules/$SHELL, and will typically contain

* extension of the MODULEPATH environment variable (e.g., via the module use command) to contain some subdirectory where a
  user's own module files can reside
* loading of additional LRZ or self-written modules at login as needed

**Naming conventions**

When loading a module, package specific environment variables follow the conventions given in this section. In the following table, the
placeholder PAC is used for the package name.

<table>
<thead>
<tr>
<th>Generic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAC_BASE</td>
<td>base directory for package</td>
</tr>
<tr>
<td>PAC_LIBDIR</td>
<td>library directory for package, usually $PAC_BASE/lib</td>
</tr>
<tr>
<td>PAC_LIB</td>
<td>library setting for (static) linking, typically $PAC_BASE/lib /lib&lt;library_name.a&gt;</td>
</tr>
<tr>
<td>PAC_SHLIB</td>
<td>library setting for dynamic linking, typically -L$PAC_BASE/lib -l&lt;library_name&gt;</td>
</tr>
<tr>
<td>PAC_INC</td>
<td>include setting for compilation, usually -I$PAC_BASE/include</td>
</tr>
<tr>
<td>PAC_DOC</td>
<td>directory containing documentation (in other than man or info format)</td>
</tr>
<tr>
<td>PAC_WWW</td>
<td>URL referring to location of package-specific documentation on the LRZ WWW server.</td>
</tr>
</tbody>
</table>

Note that occasionally a module may contain subpackages e.g., SUB, in which case additional variables of the form PAC_SUB_LIB etc.
may be defined.

**Support and sources of information on modules**

If you experience problems using the module package, please notify the LRZ HPC support team.

Further information on modules can be found by reading the man pages on the LRZ HPC systems (module (1), modulefile (4)). The
command "module help" will also provide some information.

The sources can be found on the Sourceforge git repository. For the use of modules, a Tcl (http://www.tcl.tk/) installation is needed as a
prerequisite.