Table of Contents

Name

module - command interface to the Modules package

Synopsis

module [*switches*] [*sub-command*] *[* sub-command-args]

Description

module is a user interface to the Modules package. The Modules package provides for the dynamic modification of the user's environment via *modulefiles.*

Each *modulefile* contains the information needed to configure the shell for an application. Once the Modules package is initialized, the environment can be modified on a per-module basis using the **module** command which interprets *modulefiles.* Typically *modulefiles* instruct the **module** command to alter or set shell environment variables such as PATH, MANPATH, etc. *modulefiles* may be shared by many users on a system and users may have their own collection to supplement or replace the shared *modulefiles.*

The *modulefiles* are added to and removed from the current environment by the user. The environment changes contained in a *modulefile* can be summarized through the **module** command as well. If no arguments are given, a summary of the **module** usage and *sub-commands* are shown.

The action for the **module** command to take is described by the *sub-command* and its associated arguments.

Package Initialization

The Modules package and the **module** command are initialized when a shell-specific initialization script is sourced into the shell. The script creates the **module** command as either an alias or function, creates Modules environment variables, and a snapshot of the environment is saved in ${HOME }/*.modulesbeginenv* . The **module** alias or function executes the **modulecmd** program located in ${MODULESHOME }/bin and has the shell evaluate the command's output. The first argument to **modulecmd** specifies the type of shell.

The initialization scripts are kept in ${MODULESHOME }/init/*shellname* where *shellname* is the name of the sourcing shell. For example, a C Shell user sources the ${MODULESHOME }/init/*csh* script. The sh, csh, tcsh, bash, ksh, and zsh shells are supported by **modulecmd.** In addition, python and perl "shells" are supported which writes the environment changes to stdout as python or perl code.

Examples of initialization

In the following examples, replace ${MODULESHOME } with the actual directory name.

C Shell initialization (and derivatives): source ${MODULESHOME }/init/csh
module load modulefile modulefile ...
Bourne Shell (sh) (and derivatives): . ${MODULESHOME }/init/sh
module load modulefile modulefile ...
Perl: require "${MODULESHOME }/init/perl";
&module("load modulefile modulefile ...");

Modulecmd startup

Upon invocation *modulecmd* sources rc files which contain global, user and modulefile specific setups. These files are interpreted as modulefiles. See **modulefile(4)** for detailed information.

Upon invocation of modulecmd module RC files are sourced in the following order:
Global RC file as specified by ${MODULERCFILE } or
${MODULESHOME }/etc/rc
User specific module RC file ${HOME }/.modulerc
All .modulerc and .version files found during modulefile seeking.

Command line switches

The module command accepts command line switches as its first parameter. These may be used to control output format of all information displayed and the *module* behavior in case of locating and interpreting module files.

All switches may be entered either in short or long notation. The following switches are accepted:

--*force*, -f
Force active dependency resolution. This will result in modules found on a *prereq* command inside a module file being load automatically. Unloading module files using this switch will result in all required modules which have been loaded automatically using the *-f* switch being unload. This switch is experimental at the moment.
--*terse*, -t
Display *avail* and *list* output in short format.
--*long*, -l
Display *avail* and *list* output in long format.
--*human*, -h
Display short output of the *avail* and *list* commands in human readable format.
--*verbose*, -v
Enable verbose messages during module command execution.
--*silent*, -s
Disable verbose messages. Redirect *stderr* to */dev/null* if *stderr* is found not to be a tty. This is a useful option for module commands being written into *.cshrc* ,

.login or *.profile* files, because some remote shells (as *rsh* (1)) and remote execution commands (like *rdist*) get confused if there is output on stderr.

--create, -c
: Create caches for *module avail* and *module apropos* . You must be granted write access to the ${MODULEHOME }/modulefiles/ directory if you try to invoke *module* with the *-c* option.

--icase, -i
: Case insensitive module parameter evaluation. Currently only implemented for the *module apropos* command.

--userlvl <lvl>, -u <lvl>
: Set the user level to the specified value. The argument of this option may be one of:

novice, nov Novice

expert, exp Experienced module user

advanced, adv Advanced module user

Module Sub-Commands

help [modulefile...]
: Print the usage of each sub-command. If an argument is given, print the Module specific help information for the *modulefile.*

load modulefile [modulefile...]
add modulefile [modulefile...]
: Load *modulefile* into the shell environment.

unload modulefile [modulefile...]
rm modulefile [modulefile...]
: Remove *modulefile* from the shell environment.

switch modulefile1 modulefile2
swap modulefile1 modulefile2
: Switch loaded *modulefile1* with *modulefile2.*

display modulefile [modulefile...]
show modulefile [modulefile...]
: Display information about a *modulefile.* The display sub-command will list the full path of the *modulefile* and all (or most) of the environment changes the *modulefile* will make if loaded. (It will not display any environment changes found within conditional statements.)

list
: List loaded modules.

avail [path...]
: List all available *modulefiles* in the current MODULEPATH . All directories in the MODULEPATH are recursively searched for files containing the *modulefile* magic cookie. If an argument is given, then each directory in the MODULEPATH is searched for *modulefiles* whose pathname match the argument. Multiple versions of an application can be supported by creating a

subdirectory for the application containing *modulefiles* for each version.

use directory [directory...]

use [-a|--append] directory [directory...]

Prepend directory to the MODULEPATH environment variable. The --append flag will append the directory to MODULEPATH .

unuse directory [directory...]

Remove directory from the MODULEPATH environment variable.

update

Attempt to reload all loaded *modulefiles.* The environment will be reconfigured to match the saved *${HOME }/.modulesbeginenv* and the modulefiles will be reloaded. **update** will only change the environment variables that the *modulefiles* set.

clear

Force the Modules Package to believe that no modules are currently loaded.

purge

Unload all loaded *modulefiles.*

whatis [modulefile [modulefile...]]

Display the modulefile information set up by the *module-whatis* commands inside the specified modulefiles. If no modulefiles are specified all whatis information lines will be shown.

apropos string

keyword string

Seeks thru the whatis informations of all modulefiles for the specified string. All module whatis informations matching the string search will be displayed.

initadd modulefile [modulefile...]

Add *modulefile* to the shell's initialization file in the user's home directory. The startup files checked are .cshrc, .login, and .csh_variables for the C Shell; .profile for the Bourne and Korn Shells; .bashrc, .bash_env, and .bash_profile for the GNU Bourne Again Shell; .zshrc, .zshenv, and .zlogin for zsh. The .modules file is checked for all shells. If a 'module load' line is found in any of these files, the *modulefile(s)* is(are) appended to any existing list of *modulefiles.* The 'module load' line must be located in at least one of the files listed above for any of the 'init' sub-commands to work properly. If the 'module load' line is found in multiple shell initialization files, all of the lines are changed.

initprepend modulefile [modulefile...]

Does the same as **initadd** but prepends the given modules to the beginning of the list. **initrm modulefile [modulefile...]** Remove *modulefile* from the shell's initialization files.

initswitch modulefile1 modulefile2

Switch *modulefile1* with *modulefile2* in the shell's initialization files.

initlist

List all of the *modulefiles* loaded from the shell's initialization file.

initclear

Clear all of the *modulefiles* from the shell's initialization files.

Modulefiles

modulefiles are written in the Tool Command Language (tcl) and are interpreted by **modulecmd.** *modulefiles* can use conditional statements. Thus the effect a *modulefile* will have on the environment may change depending upon the current state of the environment.

Environment variables are unset when unloading a *modulefile*. Thus, it is possible to **load** a *modulefile* and then **unload** it without having the environment variables return to their prior state.

Environment

MODULESHOME
: The location of the master Modules package file directory containing **module** command initialization scripts, the executable program **modulecmd,** and a directory containing a collection of master *modulefiles.*

MODULEPATH
: The path that the **module** command searches when looking for *modulefiles.* Typically, it is set to the master modulefiles directory, *${MODULESHOME }/modulefiles*, by the initialization script. MODULEPATH can be set using 'module use' or by the module initialization script to search group or personal *modulefile* directories before or after the master *modulefile* directory.

LOADEDMODULES
: A colon separated list of all loaded *modulefiles.*

_LOADED_MODULEFILES_
: A colon separated list of the full pathname for all loaded *modulefiles.*

MODULESBEGINENV
: The filename of the file containing the initialization environment snapshot.

Files

/usr/local
: The MODULESHOME directory.

${MODULESHOME}/etc/rc
: The system-wide *modules* rc file. The location of this file can be changed using the MODULERCFILE environment variable as described above.

${HOME}/.modulerc
: The user specific *modules* rc file.

${MODULESHOME}/modulefiles
: The directory for system-wide *modulefiles.* The location of the directory can be changed using the MODULEPATH environment variable as described above.

${MODULESHOME}/bin/modulecmd
: The *modulefile* interpreter that gets executed upon each invocation of **module.**

${MODULESHOME}/init/*shellname*
: The Modules package initialization file sourced into the user's environment.

${MODULESHOME}/init/.modulespath
: The initial search path setup for module files. This file is read by all shell init files.

${MODULEPATH}/.moduleavailcache
: File containing the cached list of all *modulefiles* for each directory in the MODULEPATH (only when the avail cache is enabled).

${MODULEPATH}/.moduleavailcachedir
: File containing the names and modification times for all sub-directories with an avail cache.

${HOME}/.modulesbeginenv

A snapshot of the user's environment taken at Module initialization. This information is used by the **module update** sub-command.

See Also

modulefile(4)

Table of Contents