MODULEFILE

Section: Modules configuration (4)
Updated: July 2009
Index Return to Main Contents

NAME

modulefile - files containing Tcl code for the Modules package

DESCRIPTION

*modulefile*s are written in the Tool Command Language, **Tcl**(3) and are interpreted by the **modulecmd** program via the **module**(1) user interface. modulefiles can be loaded, unloaded, or switched on-the-fly while the user is working; and can be used to implement site policies regarding the access and use of applications.

A *modulefile* begins with the magic cookie, '#%Module'. A version number may be placed after this string. The version number is useful as the *modulefile* format may change. If a version number doesn't exist, then **modulecmd** will assume the *modulefile* is compatible with the latest version. The current *modulefile version* is **1.0.** Files without the magic cookie will not be interpreted by **modulecmd.**

Each *modulefile* contains the changes to a user's environment needed to access an application. Tcl is a simple programming language which permits modulefiles to be arbitrarily complex, depending upon the application's and the modulefile writer's needs. If support for extended tcl (tclX) has been configured for your installation of the Modules package, you may use all the extended commands provided by tclX, too.

A typical *modulefile* is a simple bit of code that set or add entries to the PATH, MANPATH, or other environment variables. Tcl has conditional statements that are evaluated when the modulefile is loaded. This is very effective for managing path or environment changes due to different OS releases or architectures. The user environment information is encapsulated into a single *modulefile* kept in a central location. The same modulefile is used by every user on any machine. So, from the user's perspective, starting an application is exactly the same irrespective of the machine or platform they are on.

modulefiles also hide the notion of different types of shells. From the user's perspective, changing the environment for one shell looks exactly the same as changing the environment for another shell. This is useful for new or novice users and eliminates the need for statements such as "if you're using the C Shell do this ..., otherwise if you're using the Bourne shell do this ..." Announcing and accessing new software is uniform and independent of the user's shell. From the modulefile writer's perspective, this means one set of information will take care of every type of shell.

Modules Specific Tcl Commands

The Modules Package uses commands which are extensions to the "standard" *Tool Command Language* **Tcl**(3) package. Unless otherwise specified, the Module commands return the empty string. Some commands behave differently when a *modulefile* is loaded or unloaded. The command descriptions assume the *modulefile* is being loaded.

break

This is not a Modules-specific command, it's actually part of Tcl, which has been overloaded similar to the *continue* and *exit* commands to have the effect of causing the module not to be listed as loaded and not affect other modules being loaded concurrently. All non-environment commands within the module will be performed up to this point and processing will continue on to the next module on the command line. The *break* command will only have this effect if not used within a Tcl loop though.

An example: Suppose that a full selection of modulefiles are needed for various different architectures, but some of the modulefiles are not needed and the user should be alerted. Having the unnecessary modulefile be a link to the following *notavail* modulefile will perform the task as required.

```
#%Module1.0
## notavail modulefile
##
proc ModulesHelp { } {
    puts stderr "       This module does nothing but alert the user"
    puts stderr "       that the [module-info name] module is not available"
}

module-whatis   "Notifies user that module is not available."
set curMod [module-info name]
if { [ module-info mode load ] } {
    puts stderr "Note: '$curMod' is not available for [uname sysname]."
}

break
```

chdir *directory*

Set the current working directory to *directory*.

continue

This is not a modules specific command but another overloaded Tcl command and is similar to the *break* or *exit* commands except the module will be listed as loaded as well as performing any environment or Tcl commands up to this point and then continuing on to the next module on the command line. The *continue* command will only have this effect if not used within a Tcl loop though.

exit [N]

This is not a modules specific command but another overloaded Tcl command and is similar to the *break* or *continue* commands. However, this command will cause the immediate cessation of this module and any additional ones on the command line. This module and the subsequent modules will not be listed as loaded. No environment commands will be performed in the current module.

The integer value *N* after the *exit* command will be used as an command exit value; however, some shells (*bash* being one of them) do not carry this value through an *eval* though. For *bash* and those Bourne shells (*/bin/sh*) being emulated by the *bash* will have return a non-zero value as a result of *test 0 = 1* being appended to the evaluate string.

setenv *variable value*

Set environment *variable* to *value*. The **setenv** command will also change the process' environment. A reference using Tcl's **env** associative array will reference changes made with the **setenv** command. Changes made using Tcl's **env** associative array will **NOT** change the user's environment variable like the **setenv** command. An environment change made this way will only affect the module parsing process. The **setenv** command is also useful for changing the environment prior to the **exec** or **system** command. When a *modulefile* is unloaded, **setenv** becomes **unsetenv**. If the environment variable had been defined it will be overwritten while loading the modulefile. A subsequent unload will unset the environment variable - the previous value cannot be restored! (Unless you handle it explicitly ... see below.)

unsetenv *variable* [*value*]

Unsets environment *variable*. However, if there is an optional *value*, then when unloading a module, it will set *variable* to *value*. The **unsetenv** command changes the process' environment like **setenv**.

append-path [*-d C* | *--delim C* | *--delim=C*] *variable value*
prepend-path [*-d C* | *--delim C* | *--delim=C*] *variable value*

Append or prepend *value* to environment *variable*. The *variable* is a colon, or delimiter, separated list such as
"PATH=directory:directory:directory". The default delimiter is a colon ':', but an arbitrary one can be given by the *--delim* option. For example a space can be used instead (which will need to be handled in the Tcl specially by enclosing it in " " or { }). A space, however, can not be specified by the *--delim=C* form.

If the *variable* is not set, it is created. When a *modulefile* is unloaded, **append-path** and **prepend-path** become **remove-path**.

remove-path [*-d C* | *--delim C* | *--delim=C*] *variable value*

Remove *value* from the colon, or delimiter, separated list in *variable*. See *prepend-path* or *append-path* for further explanation of using an arbitrary delimiter. Every string between colons, or delimiters, in *variable* is compared to *value*. If the two match, *value* is removed from *variable*.

prereq *modulefile* [*modulefile* ...]
conflict *modulefile* [*modulefile* ...]

prereq and **conflict** control whether or not the *modulefile* will be loaded. The **prereq** command lists modulefiles which must have been previously loaded before the current *modulefile* will be loaded. Similarly, the **conflict** command lists modulefiles which conflict with the current *modulefile*. If a list contains more than one *modulefile*, then each member of the list acts as a Boolean OR operation. Multiple **prereq** and **conflict** commands may be used to create a Boolean AND operation. If one of the requirements have not been satisfied, an error is reported and the current *modulefile* makes no changes to the user's environment.

If an argument for **prereq** is a directory and any *modulefile* from the directory has been loaded, then the prerequisite is met. For example, specifying *X11* as a prereq means that any version of *X11*, *X11/R4* or *X11/R5*, must be loaded before proceeding.

If an argument for **conflict** is a directory and any other *modulefile* from that directory has been loaded, then a conflict will occur. For example, specifying *X11* as a conflict will stop *X11/R4* and *X11/R5* from being loaded at the same time.

is-loaded *modulefile* [*modulefile* ...]

The **is-loaded** command returns a true value if any of the listed modulefiles has been loaded. If a list contains more than one *modulefile*, then each member acts as a boolean OR operation. If an argument for **is-loaded** is a directory and any *modulefile* from the directory has been loaded **is-loaded** would return a true value.

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

Contains the same sub-commands as described in the **module**(1) man page in the Module Sub-Commands section. This command permits a *modulefile* to load or remove other modulefiles. No checks are made to ensure that the *modulefile* does not try to load itself. Often it is useful to have a single *modulefile* that performs a number of **module load** commands. For example, if every user on the system requires a basic set of applications loaded, then a *core modulefile* would contain the necessary **module load** commands.

module-info *option* [*info-args*]

Provide information about the **modulecmd** program's state. Some of the information is specific to the internals of **modulecmd**. *option* is the type of information to be provided, and *info-args* are any arguments needed.

module-info flags

Returns the integer value of **modulecmd's** flags state.

module-info mode [*modetype***]**

Returns the current **modulecmd's** mode as a string if no *modetype* is given.

Returns 1 if **modulecmd's** mode is *modetype*. *modetype* can be: *load, remove, display, help, whatis, switch, switch1, switch2,* or **switch3.**

module-info name

Return the name of the *modulefile*. This is not the full pathname for *modulefile*. See the Modules Variables section for information on the full pathname.

module-info specified

Return the name of the *modulefile* specified on the command line.

module-info shell

Return the current shell under which *modulecmd* was invoked. This is the first parameter of *modulecmd*, which is normally hidden by the *module* alias.

module-info shelltype

Return the family of the shell under which *modulefile* was invoked. As of *module-info shell* this depends on the first parameter of *modulecmd*. The output reflects a shell type determining the shell syntax of the commands produced by *modulecmd*.

module-info alias *name*

Returns the full module file name to which the module file alias *name* is assigned

module-info version *module-file*

Returns a list of all symbolic versions assigned to the passed *module-file*. The paremeter *module-file* might either be a full qualified module file with name and version, another symbolic module file name or a module file alias.

module-version *module-file version-name [version-name ...]*

Assignes the symbolic *version-name* to the module file *module-file* This command should be placed in one of the modulecmd rc files in order to provide shorthand invocations of frequently used module file names.

The special *version-name* **default** specifies the default version to be used for module commands, if no specific version is given. This replaces the definitions made in the *.version* file in former **modulecmd** releases.

The parameter *module-file* may be either

a fully qualified modulefile with name and version

a symbolic module file name

another module file alias

module-alias *name module-file*

Assignes the module file *module-file* to the alias *name*. This command should be placed in one of the modulecmd rc files in order to provide shorthand invocations of frequently used module file names.

The parameter *module-file* may be either

a fully qualified modulefile with name and version

a symbolic module file name

another module file alias

module-trace *{on|off} [command [command ...]] [-module modulefile [modulefile ...]]*

Switches tracing on or off. Without parameters this command will affect globally all tracing setups for all commands and modulefiles. The *command* parameter may be used to affect tracing of specified module commands only and the switch *-module* finally limits the affect of the *module-trace* command to a well defined set of module files.

The *command* may be one of the following

avail - 'module avail' command

clear - 'module clear' command

display - 'module display' command

init - 'module init' command

help - 'module help' command

list - 'module list' command

load - 'module load' command

purge - 'module purge' command

switch - 'module switch' command

unuse - 'module unuse' command

unload - 'module unload' command

update - 'module update' command

use - 'module use' command

The *module* parameter specifies a set of module files using TCL regular expressions. For example

.* will affect all module files

**/2.0* affects all module files at version 2.0

*gnu/.** affects all versions of the gnu modulefile

gnu/2.0 affects only version 2.0 of the gnu modulefile

The *module* parameter is prepended to the current tracing pattern list for the specified module command. It is evaluated from the left to the right. The first matching pattern defines the tracing parameter.

The internal trace pattern list is stored as a colon separated list. In advanced user level only, colons may be specified on the *module* parameter of the *module-trace* command. This will directly take effect in the internal trace pattern list. In novice or expert user level a warning messge will be generated.

module-user *level*

Defines the user level under wich *module-cmd* runs. This takes effect on the error messages being produced and on the behavior of *modulecmd* in case of detecting an outage.

The *level* parameter specifies the user level and may be one of the following values:

advanced, adv - advanced user level

expert, exp - expert user level

novice, nov - novice user level

module-verbosity *{on|off}*

Switches verbose *modulecmd* message display on or off.

module-log *error-weight log-facility*

Defines whether error messages of the specified weight should be logged and conditionally assignes a log-facility.

The *error-weight* parameter specifies the error level to be logged. It may be one of the following values:

verb - verbose messages

info - informal messages

debug - debugging messages

trace - tracing output

warn - warnings

prob - problems (normally the modulecmd may be completed)

error - errors (which normally leads to unsuccessful end of the modulecmd)

fatal - fatal system errors

panic - very fatal system errors, e.g. internal program inconsistencies.

The *log-facility* parameter specifies the log destination. This may either switch off logging for the specified *error-weight*, direct log messages to a special stream or a file or specify a syslog facility for logging. The following values are allowed:

stderr, stdout - predefined output streams for normal and error outputs. Note, that stdout is normally used for passing parameters to the invoking shell. Directing error output to this stream might screw up the *modulecmd* integration to your shell.

a syslog facility - directs logging to the syslog. See **syslog.conf(4)** for detailed description of the valid syslog facilities.

null, none - will suppress logging of the specified *error-weight*.

a filename - is recognized by the first character being either a '.' or a '/'. You must have write permission to the file you specify.

module-whatis *string*

Defines a string which is displayed in case of the invocation of the 'module whatis' command. There may be more than one *module-whatis* line in a modulefile. This command takes no actions in case of load, display, etc. invocations of *modulecmd*.

The *string* parameter has to be enclosed in double-quotes if there's more than one word specified. Words are defined to be separated by whitespace characters (space, tab, cr).

set-alias *alias-name alias-string*

Sets an alias or function with the name *alias-name* in the user's environment to the string *alias-string*. Arguments can be specified using the Bourne Shell style of function arguments. If the string contains "$1", then this will become the first argument when the alias is interpreted by the shell. The string "$*" corresponds to all of the arguments given to the alias. The character '$' may be escaped using the '\' character.

For some shells, aliases are not possible and the command has no effect. For Bourne shell derivatives, a shell function will be written (if supported) to give the impression of an alias. When a *modulefile* is unloaded, **set-alias** becomes **unset-alias**.

unset-alias *alias-name*

Unsets an alias with the name *alias-name* in the user's environment. If the shell supports functions then the shell is instructed to unset function *alias-name*.

system *string*

Pass *string* to the C library routine **system**(3). For the **system**(3) call **modulecmd** redirects stdout to stderr since stdout would be parsed by the evaluating shell. The exit status of the executed command is returned.

uname *field*

Provide fast lookup of system information on systems that support **uname**(3). **uname** is significantly faster than using **system** to execute a program to return host information. If **uname**(3) is not available, **gethostname**(3) or some program will make the nodename available. **uname** will return the string "unknown" if information is unavailable for the *field*.

uname will invoke **getdomainname** in order to figure out the name of the domain.

field values are:

sysname - the operating system name

nodename - the hostname

domain - the name of the domain

release - the operating system release

version - the operating system version

machine - a standard name that identifies the system's hardware

x-resource *resource-string*
x-resource *filename*

Merge resources into the *X11* resource database. The resources are used to control look and behavior of X11 applications. The command will attempt to read resources from *filename*. If the argument isn't a valid file name, then string will be interpreted as a resource. If a file is found, it will be filtered through the **cpp**(1) preprocessor, just as **xrdb**(1) would do.

modulefiles that use this command, should in most cases contain one or more *x-resource* lines, each defining one X11 resource. Reading resources from *filename* is much slower, due to the preprocessing. The DISPLAY environment variable should be properly set and the X11 server should be accessible. If **x-resource** can't manipulate the X11 resource database, the *modulefile* will exit with an error message.

Examples:

x-resource /u2/staff/leif/.xres/Ileaf

The file *Ileaf* is preprocessed by **cpp**(1) and the result is merged into the X11 resource database.

x-resource [glob ~/.xres/ileaf]

The Tcl *glob* function is used to have the *modulefile* read different resource files for different users.

x-resource {Ileaf.popup.saveUnder: True}

Merge the *Ileaf* resource into the X11 resource database.

Modules Variables

The **ModulesCurrentModulefile** variable contains the full pathname of the *modulefile* being interpreted.

Locating Modulefiles

Every directory in MODULEPATH is searched to find the *modulefile*. A directory in MODULEPATH can have an arbitrary number of sub-directories. If the user names a *modulefile* to be loaded which is actually a directory, the directory is opened and a search begins for an actual *modulefile*. First, **modulecmd** looks for a file with the name *.modulerc* in the directory. If this file exists, its contents will be evaluated as if it was a module file to be loaded. You may place *module-version* and *module-alias* commands inside this file. Additionally, before seeking for *.modulerc* files in the module directory, the global *.modulerc* file is sourced, too. If a named version *default* now exists for the module file to be loaded, the assigned modulefile now will be sourced. Otherwise the file *.version* is looked up in the directory. If the *.version* file exists, it is opened and interpreted as Tcl code. If the Tcl variable **ModulesVersion** is set by the *.version* file, **modulecmd** will use the name as if it specifies a *modulefile* in the directory. This will become the *default* module file in this case. If **ModulesVersion** is a directory, the search begins anew down that directory. If the name does not match any files located in the current directory, the search continues through the remaining directories in MODULEPATH.

Every *.version* and *.modulerc* file found is Tcl interpreted. So, changes made in these file will affect the subsequently interpreted *modulefile*.

If no *default* version may be figured out, then the highest lexicographically sorted *modulefile* under the directory will be used.

For example, it is possible for a user to have a directory named *X11* which simply contains a *.version* file specifying which version of X11 is to be loaded. Such a file would look like:

```
#%Module1.0
##
##  The desired version of X11
##
set ModulesVersion "R4"
```

Modulefile Specific Help

Users can request help about a specific *modulefile* through the **module**(1) command. The *modulefile* can print helpful information or start help oriented programs by defining a **ModulesHelp** subroutine. The subroutine will be called when the 'module help *modulefile*' command is used.

Modulefile Display

The 'module display *modulefile*' command will detail all changes that will be made to the environment. After displaying all of the environment changes **modulecmd** will call the **ModulesDisplay** subroutine. The **ModulesDisplay** subroutine is a good place to put additional descriptive information about the *modulefile*.

ENVIRONMENT

${MODULEPATH}
Path of directories containing *modulefiles*.

VERSION

3.2.7

SEE ALSO

module(1), **Tcl**(3), **TclX**(3), **xrdb**(1), **cpp**(1), **system**(3), **uname**(3), **gethostname**(3) **getdomainname**(3)

NOTES

Tcl was developed by John Ousterhout at the University of California at Berkeley.

TclX was developed by Karl Lehenbauer and Mark Diekhans.

Index

This document was created by man2html, using the manual pages.
Time: 17:14:13 GMT, July 31, 2009