VChannel

Library version:	RENAT 0.1.8
Library scope:	test suite
Named arguments:	supported

Introduction

A basic library that provides Terminal connection to routers/hosts

`VChannel` is a core RENAT library that maintains input/output to nodes with an attached virtual terminal. It encapsulates the SSH/Telnet connections behind and provides common usage of access and execute commands to the nodes. Each channel instance has its own log file and a virtual terminal.

Table of Contents

Device, Node and Channel

RENAT has 3 types of connection target. `Device`, `Node` and `Channel`.

Device

Each device stands for a real physical box that has its own IP address and is defined in the master file `device.yaml`. Users do not directly use `device` in keywords.

Node

Node is a logical instance of a `device`. It could stand for a logical instance of a router or just a virtual terminal to the router. Nodes were defined in `local.yaml` of the test case. Several nodes could point to a same device.

Channel

Each channel holds a session to a node. Each channel has its own log file and a virtual terminal. Any command used by *Cmd*,*Write* or *Read* will be logged to the log file. Each channel is identified by a name when it is created with *Connect* keyword and is released with *Close* keyword.

Notes: multi sessions to a same device could be done with predefined multi nodes to same device in the `local.yaml` file or by using multi *Connect* with different *name*.

Connections

The library provides a channel to a target node. Each channel is attached with a virtual terminal. Input and output to the node are made through this virtual terminal. This will help to provide the output looks like the output when operator is using the real terminal.

When keywords *Read*, *Write*, *Cmd* are used, if the connection is not available anymore, the system will try to reconnect to the host with the information provided in the 1st connect. It will try `max_retry_for_connect` times and wait for `interval_between_retry` seconds between retries. The values of `max_retry_for_connect` and `interval_between_retry` are defined in `./config/config.yaml`

Usually when RENAT could not make the connections to the target, the system will raise an exception. But if the `ignore_dead_node` is defined as `yes` in the current active `local.yaml`, the system will ignore the dead node, remove it from the global variable `LOCAL['node']` and `NODE` and keep running the test.

Shortcuts

Change Log · **C**hange Prompt · **C**lose · **C**lose All · **C**md · **C**md And Wait For · **C**md Yesno · **C**onnect · **C**onnect All · **E**xec File · **F**lush All · **G**et Channel · **G**et Channels · **G**et Current Channel · **G**et Ip · **L**og · **R**ead · **R**econnect · **S**et Log Separator · **S**nap · **S**nap Diff · **S**tart Screen Mode · **S**top Screen Mode · **S**witch · **W**rite

Keywords

Keyword	Arguments	Documentation
Change Log	*log_file*, *mode=w*	Stops current log file and create a new log file. Every log from that point will be saved to the new log file Return old log filename
Change Prompt	*str_prompt*	Changes the current prompt of the channel Returns previous prompt. User should change the prompt `before` execute the new command that expects to see new prompt. Example: Router.*Switch* | vmx11 | ${prompt}= | VChannel.*Change Prompt* | % VChannel.*Cmd* | start shell | VChannel.*Cmd* | ls |

		VChannel.*Change Prompt* ${prompt} Vchannel.*Cmd* exit
Close		Closes current connection and returns the active channel name
Close All		Closes all current sessions and flush out all log files. Current node name was reset to `None`
Cmd	*command=, prompt=, match_err=(unknown command.|syntax error, expecting <command>.)*	Executes a `command` and wait until for the prompt. This is a blocking keyword. Execution of the test case will be postponed until the prompt appears. If `prompt` is a null string (default), its value is defined in the `./config/template.yaml` Output will be automatically logged to the channel current log file. See Common for details about the config files.
Cmd And Wait For	*command, keyword, interval=30s, max_num=10, error_with_max_num=True*	Execute a command and expect `keyword` occurs in the output. If not wait for `interval` and repeat the process again After `max_num`, if `error_with_max_num` is `True` then the keyword will fail. Ortherwise the test continues.
Cmd Yesno	*cmd, ans=yes, question=? [yes,no]*	Executes a `cmd`, waits for `question` and answers that by `ans`
Connect	*node, name, log_file, timeout=20m, w=80, h=32, mode=w*	Connects to the node and create a VChannel instance Login information is automatically extracted from yaml configuration. By defaullt a virtual terminal (vty100) with size 80x64 is attachted to this channel. If a login was successful, VChannel will create a log file name `log_file` for the connection in the current result folder of the test case. This log file will contain any command input/output executed on this channel. Multi sessions to the same node could be open with different names. Use *Switch* to change the current active session by its name Examples: *Connect* vmx11 vmx11 vmx11.log *Connect* vmx11 vmx11 vmx11.log 80 64 See `Common` for more detail about the yaml config files.
Connect All	*prefix=*	Connects to **all** nodes that are defined in active `local.yaml`. A prefix `prefix` was appended to the alias name of the connection. A new log file by `<alias>.log` was automatiocally created. See *Common* for more detail about active `local.yaml`

Exec File	*file_name, vars=, comment=# , step=False, str_error=syntax,rror*	Executes commands listed in `file_name` Lines started with `comment` character is considered as comments
		`file_name` is a file located inside the `config` folder of the test case.
		This command file could be written in Jinja2 format. Default usable variables are `LOCAL` and `GLOBAL` which are identical to `Common.LOCAL` and `Common.GLOBAL`. More variables could be supplied to the template by `vars`.
		`vars` has the format: `var1=value1,var2=value2`
		If `step` is `True`, after very command the output is check agains an error list. And if a match is found, execution will be stopped. Error list is define by `str_err`, that contains multi regular expression separated by a comma. Default value of `str_err` is *error*
		A sample for command list with Jinja2 template:
		`show interface {{ LOCAL['extra']['line1'] }}`
		`show interface {{ LOCAL['extra']['line2'] }}`
		`{% for i in range(2) %}`
		`show interface et-0/0/{{ i }}`
		`{% endfor %}`
		Examples:
		Router.*Exec File* cmd.lst
		Router.*Exec File* step=${TRUE} str_error=syntax,error
		Note: Comment in the middle of the line is not supported For example if `comment` is "# "
		`# this is comment line <-- this line will be ignored`
		`## this is not an comment line, and will be enterd to the router cli,`
		but the router might ignore this

Flush All		
Get Channel	*name*	Returns a channel by its `name`
Get Channels		Returns all current vchannel instances

Get Current Channel		Returns the current active channel
Get Ip		Returns the IP address of current node Examples: `${router_ip}=` `Router.Get IP`
Log	*msg*	Writes the log message `msg` to current log file of the channel
Read	*silence=False*	Returns the current output of the virtual terminal and automatically logs to file. In `normal mode` this will return the **unread** output only, not all the content of the screen.
Reconnect	*name*	Reconnects to the `name` node using existed information The only difference is that the mode of the log file is set to `a+` by default
Set Log Separator	*sep=*	Set a separator between the log of `read`, `write` or `cmd` keywords
Snap	*name*, **cmd_list*	Remembers the result of a list of command defined by `cmd_list` Use this keyword with *Snap Diff* to get the difference between the command's result. The a new snapshot will overrride the previous result. Each snap is identified by its `name`
Snap Diff	*name*	Executes the comman that have been executed before by `name` snapshot and return the difference. Difference is in `context diff` format
Start Screen Mode		Starts the `screen mode`. In the `screen mode`, the output is just the same with the real terminal. It means that any real-time application likes `top` will be captured as-is. Consecutive *read* from this VChannel instance may produce redundancy ouput.
Stop Screen Mode		Stops the `screen mode` and returns to `normal mode` In `screen mode`, *Write* does not return any thing and no output is logged. In `normal mode`, escape sequences are not processed by the virtual terminal.
Switch	*name*	Switches the current active channel to `name`. There only one active channel at any time Examples: `VChannel.Switch` `vmx12`

Write	*str_cmd*, *str_wait=0s*, *start_screen_mode=False*	Sends `str_cmd` to the target node and return after `str_wait` time.
		If `start_screen_mode` is `True`, the channel will be shifted to `Screen Mode`. Default value of `screen_mode` is False.
		In `normal mode`, a `new line` char will be added automatically to the `str_cmd` and the command return the output it could get at that time from the terminal and also logs that to the log file.
		In `screen Mode`, if it is necessary you need to add the `new line` char by your own and the ouput is not be logged or returned from the keyword.
		Parameters:
		▪ `str_cmd`: the command
		▪ `str_wait`: time to wait after apply the command
		▪ `start_screen_mode`: whether start the `screen mode` right after writes the command
		Special input likes Ctrl-C etc. could be used with global variable ${CTRL-<char>}
		Returns the output after writing the command the the channel.
		When *str_wait* is not *0s*, the keyword `read` and `return the output` after waiting *str_wait*. Otherwise, the keyword return without any output.
		Notes: This is a non-blocking command.
		Examples:
		`VChannel.Write` `monitor interface traffic` `start_screen_mode=${TRUE}`
		`VChannel.Write` `${CTRL_C}` `# simulates Ctrl-C`

Altogether 25 keywords.
Generated by Libdoc on 2018-08-01 21:52:56.