try

Command Reference

Table of contents

1. Description

try catch finally.

The `try` command models the familiar try/catch/finally control structure. The command lets you run an ad hoc or defined command and optionally run a different one if the first one fails and yet another one after the first one has finished.

The `try` command will attempt to execute a command. Commands can be one of these things:

- executable: Run the specified executable
- script: Evaluate the script using the specified executable
- scriptfile: Run the script file using the specified executable
- command: Run the defined command in the specified module

The "catch" and "finally" actions are similarly specified but are prefixed with that corresponding name (eg, catchexecutable, catchscript, finallyexecutable, finallyscript).

If an error occurs during the execution of the command and no catch command is specified then `try` will exit with an error. If a catch command is specified, the `try` command will exit normally (assuming the catch command or finally one does not error too).

static: This command can be run outside of an object context.

2. Usage

```
ctl -m logicutil -c try [-argline <>] [-catchargline <>]
[-catchcommand <>] [-catchexecutable <>] [-catchscript <>]
[-catchscriptfile <>] [-command <>] [-executable <>]
[-finallyargline <>] [-finallycommand <>] [-finallyexecutable
<>] [-finallyscript <>] [-finallyscriptfile <>] [-output <>]
[-script <>] [-scriptfile <>]
```

2.1. Options

Options are grouped into roughly three parts: the action to try, the catch action and the finally action.

```
try [try options] [catch options] [finally options]
```

Ad hoc commands are supported via the `executable` and `script` and `scriptfile` options. You can run a defined command via the `command` options. Defined commands are specified using *type#command* (eg, netutil#listening).

If a script or scriptfile is specified but no executable, the command will default executable to `sh` on unix or `cmd.exe` on windows.

Option	Description	Type	Default
`argline`	*The tryargline to try.*	string	
`catchargline`	*The catch argline.*	string	
`catchcommand`	*The catch CTL command.*	string	
`catchexecutable`	*The catch executable.*	string	
`catchscript`	*The catch script.*	string	
`catchscriptfile`	*The catch scriptfile.*	string	
`command`	*The CTL command to try.*	string	
`executable`	*The executable to try.*	string	
`finallyargline`	*The finally argline.*	string	
`finallycommand`	*The finallycommand CTL command.*	string	
`finallyexecutable`	*The finally executable.*	string	
`finallyscript`	*The finally script.*	string	
`finallyscriptfile`	*The finally scriptfile.*	string	
`output`	*Direct try output to file.*	string	
`script`	*The script to try.*	string	
`scriptfile`	*The scriptfile to try.*	string	

3. Examples

Execute an inline shell script. Echos the string "hello" to the console:

```
ctl -p demo -m logicutil -c try -- -executable /bin/sh -script "echo hello"
```

output: `hello`

Execute a script file. Here's a script called "/tmp/hello.sh" that echos the argument string specified via the "-argline" option:

```
#!/bin/sh
echo "$@"
```

Use the "scriptfile" option. The "argline" argument will be passed as the arguments to the script.

```
ctl -p demo -m logicutil -c try -- -executable /bin/sh -scriptfile
/tmp/hello.sh -argline hello
```

output: `hello`

Catch errors with a *catch* action. A catch action is specified similar to the one being tried. Here's an example that intentionally fails by causing the script to exit non-zero.

```
ctl -p demo -m logicutil -c try -- \
  -executable /bin/sh -script "exit 1" -catchexecutable /bin/sh
-catchscript "echo caught the error"
```

output: `caught the error`

Errors that are caught prevent the `try` command from exiting with an error. The command will exit with a 0 exit code. (eg, `$? = 0`)

A *finally* action will run no matter if an error is occurs or not

```
ctl -p demo -m logicutil -c try -- \
  -executable /bin/sh -script "exit 1" -catchexecutable /bin/sh -script
"echo caught the error" \
  -finallyexecutable /bin/sh -finallyscript "echo caught the error"
```

output:

```

Caught exception: shell-exec returned: 1
caught the error
finally
```

Commands defined in modules can also be called. Commands are referenced using this notation: typename#commandname.

Here's an example that calls the `available` command in the fileutil module.

```
ctl -p demo -m logicutil -c try -- -command fileutil#available -argline
"-file /etc/motd"
```

output: `true`

Here `available` is run again but this time referring to a file that does not exist.

```
ctl -p demo -m logicutil -c try -- \
          -command fileutil#available -argline "-file /tmp/bogus
-failonerror" -catchscript "echo file was bogus"
```

output:

```

Caught exception: The following error occurred while executing this line:
/Users/alexh/ctier/ctl/modules/fileutil/commands/available.xml:26: file not
```

try

```
found: /tmp/bogus
file was bogus
```