

Running defined commands via ctl

Table of contents

1 Command Line.....	2
1.1 Files and Environment Variables.....	2
1.2 Default project.....	2
1.3 Options.....	3
1.4 Context.....	4
1.5 Running commands.....	5
1.6 Listings.....	6
2 Logging.....	7

This document assumes you have a controller module you wish to invoke via CTL and would like to know how.

Other documents that are useful to new CTL users:

- [Download CTL](#) - Get and install the CTL software.
- [Project Setup](#) - Setup a CTL project.
- [Running overview](#) - Running ctl-exec versus ctl.

Note:

Ensure the CTL executables are in your PATH and CTL_BASE and CTL_HOME environment variables are also set.

1. Command Line

If you've installed CTL as described in the [Installing CTL](#) section, running the CTL command-line is simple: just type `ctl`.

The `ctl` command is used to execute commands to installed modules. The following sections describe how to list and invoke commands via `ctl`.

For a detailed discussion of the command line consult the [CTL command reference](#).

1.1. Files and Environment Variables

Wrapper scripts exist for Unix and Windows to run the `ctl` launcher. These launcher scripts read an initialization profile. For Unix OSes, the `$CTL_BASE/etc/profile` file is read. On Windows, `%CTL_BASE%\etc\profile.bat` is called. These files are created by the `ctl-setup` command.

The wrapper scripts depend on the following environment variables:

- `CTL_HOME` - full path to the CTL software install base
- `CTL_BASE` - full path to the CTL repository base

If either is not set, the `ctl` command will return an error.

1.2. Default project

CTL commands are always performed within a project. For the sake of convenience you can omit specifying the project parameter if there is just one project. By convention, CTL will create a default project (named "default") where you will initially begin loading modules and running commands.

If there is more than one project, the `ctl` command will give an error saying "no project specified". You will then need to supply the `-p project` parameter.

1.3. Options

CTL has a variety of options that can be displayed via `-help`. Typing `ctl -help` shows a complete listing of all the `ctl` options and some examples.

User defined commands stored in modules may have their own options. Information about these options is displayed via `ctl`'s listing feature. To specify options specific to commands use two dashes after specifying the `ctl` options.

The general option pattern follows this form:

```
ctl [ctl-options] [-- [command-options]]
```

For example, the following command shows several `ctl` options and one option specific to a command.

```
ctl -I 'web.*' -m shellutil -c ps -- -pname bash
```

When no arguments are specified, `ctl` falls into listing mode. See below.

Commands that define their own options can declare them as optional or required. They may also have their own default values. If a required option is not specified, the command will fail printing its usage. Remember to use the `--` (double dash) to separate the `ctl` options from those of the command.

Note:

Bash users: Users that run the bash shell can take advantage of CTL's command completion capability. Begin typing `ctl` command line syntax and hit the tab key. The command completion should suggest the next command option.

1.3.1. Quoting whitespace

Command-options that take arguments with white space must be specially quoted to ensure the strings are not broken up during the option parsing.

The format uses both double and single quotes around your the argument containing whitespace: `" 'string with whitespaces' "`

For example, imagine a command named `hello` that takes an option named `-message`. If the argument passed to `-message` has whitespace use the quoting format shown above:

```
ctl -m helloworld -c hello -- -message "'hi there'"
```

1.3.2. Node Dispatching Options

The `ctl` shell command includes a feature called [node dispatch](#) that allows you to call commands across a set of hosts without having to specify specific host names. Internally, CTL looks up the nodes that have the module (or object if that is what you specified), and then dispatches that command to those matching nodes. Options provide a way to filter the lookup results.

Also, using either of the two following options enables `nodedispatch`.

- `-I, --nodes`: Run the command on all nodes matching the regular expression.
- `-X, --xnodes`: Do not run the command on any node matching the regular expression. If no `-I` flag is specified, all nodes are looked up and then those matching the `-X` expression are filtered out.

See the document [Node Filtering Options](#) for further discussion of node filtering.

Two other options of interest for "nodedispatch" are:

- `-C, --threadcount`: For all the matching `nodedispatch` results, execute the commands in the specified number of threads.
- `-K`: If specified, `ctl` will continue if any errors occur.

Supporting node dispatching, is a configuration file called [deployments.properties](#), that declares to which nodes modules and objects are deployed. See the [server setup](#) section for information about setting up a repository to centrally maintain and distribute the `deployments.properties` file.

Here are some `ctl` examples using node dispatch.

1. Check if the web servers are listening on port 80:

```
ctl --nodes "web.*" -m netutil -c listening -- -port 80
```

2. Run an arbitrary shell script to see who is logged in on all the dev machines except don't run it on localhost:

```
ctl --nodes ".*.dev.acme.com" --xnode localhost -m shellutil -c exec -- -executable who
```

3. Run the `pgrep` command across all the machines in dev using 10 threads:

```
ctl -C 10 -I ".*.dev.acme.com" -X localhost -m shellutil -c exec -- \
  -executable bash -script "pgrep httpd"
```

1.4. Context

CTL commands execute within a data context. There are two kinds:

- **Static:** In a static context, the command can access control dispatch metadata, framework and project configuration, local system and environment and any command line option data, as well as any data defined within the command's module.
- **Object:** In an object context, the control dispatcher will also load instance data for the specified object. All objects are derived from subtypes of the `Managed-Entity` module.

Note:

Why have object context? Object context provides a way to manage additional unique information. Say you have a module that has different sets of data depending on what machine or environment it needs to run. You can define an object for each case and invoke your same commands but targeted to the context of each object. See the [CTL Objects](#) page for an in depth explanation.

1.5. Running commands

The `ctl` command has two command invocation styles to support the static and object contexts described above.

The usage pattern for a static command is:

```
ctl -m module -c command
```

The `-m` option specifies the name of the module. The `-c` option specifies the name of the command in that module.

By convention, simple utility commands are designed to run in a static context. For example, the commands in the [coreutils](#) library work in a static context.

One of the `coreutil` modules, called "shellutils" has a command called `ps`, that shows a normalized set of process information.

```
ctl -m shellutil -c ps
```

The other mode in which a command can be run is within an object context. Instead of using the `-m module` argument, you will use the `-t type` and `-o object` options to specify an object context.

The usage pattern for an object context command is:

```
ctl -t type -o object -c command
```

The `type` argument is the object's type name but also corresponds to the module of the same name. Object instance data is stored in the project depot and is loaded at execution time.

One module that is designed to run in both a static and object context is "ProjectBuilder". This module is used to develop new CTL modules. The example below shows the

create-type command running in the context of an object named "default".

```
ctl -t ProjectBuilder -o default -c create-type
```

1.6. Listings

The `ctl` command supports a listing feature allowing you to list all the projects and their contents: modules their commands, and possibly their objects.

As mentioned earlier the install will have created a project called "default" along with a set of modules.

The example below shows modules in the default project:

```
$ ctl
shellutil: various shell utilities
commands: [env, exec, kill, pkill, ps, whoami]
...snip...
```

The example above gave an overview of the modules and commands in that project but if you want to know more about the commands for a given module add the "`-m modulename`" arguments:

```
$ ctl -m shellutil
shellutil: various shell utilities
[commands]
  env*: get environment info
        [options]
        -key <>: env key
  exec*: executes a command
        [options]
        -scriptfile <>: scriptfile to execute
        -failonerror <>: fail if there is an error
        [-executable <>]: executable path
        -timeout <>: Stop the command if it doesn't finish within the specified
time
        -os <>: list of Operating Systems on which the command may be executed.
        -output <>: Name of a file to which to write the output.
        -script <>: script to execute
        -argline <>: arguments for execututable
  kill*: kill process
        [options]
        -sig <>: signal to send
        [-pid <>]: process id
  pkill*: kill named process
        [options]
        -pname <>: process name
        -sig <>: signal to send
```

```
ps*: get process info
  [options]
  -pname <>: process name
  -output <>: output file
  -pid <>: process id
  -format <>: output format. (plain, yaml)
whoami*: print the user name
```

Note:

What does the asterisk next to the command mean? Commands that are marked as `static` get an asterisk denoting that you can run them outside of an object context.

2. Logging

All commands run via the `ctl` command are logged in the framework's log, `command.log`. The default configuration locates the log as shown below.

```
$CTL_BASE/var/logs/command.log
```

CTL uses [Apache log4j](#) for logging and its configuration can be controlled via the log4j configuration file: `$CTL_BASE/etc/log4j.properties`.

Users can control the logging level using the `--level` option for `ctl`.