Paste Script

Release 1.7.5

September 24, 2019

Contents

author Ian Bicking <ianb@colorstudy.com>

revision Rev

date $LastChangedDate$

Contents:

CHAPTER 1

News: Paste Script

Contents

1.1 3.1.0 (2019-03-04)

- Remove dependency on `unittest2`.

1.2 3.0.0 (2018-11-26)

- Moved to GitHub.
- Put into maintenance mode, meaning: critical bugs will be fixed, and support for new versions of Python will be handled, but new features are not being considered.

1.3 2.0.2 (2015-05-27)

- Issue #20: Fix "paster points –list" on Python 3.
- Fix cgi_server on Python 3.
- Fix usage of the sort() method on Python 3.
- Fix grep on Python 3.

1.4 2.0.1 (2015-05-26)

- Fix –help command on Python 3. Patch written by Atsushi Odagiri (aodag).
- Fix "paster create –template=basic_package test" command on Python 3. Patch written by Atsushi Odagiri (aodag).
- Fix error when 'paster create –list-template' on Python 3. Patch written by FGtatsuro.
- Create universal wheel package.

1.5 2.0 (2015-05-26)

- Experimental Python 3 support.
- six module is now required.
- Drop support of Python 2.5 and older

1.6 1.7.5

- Import CherryPy directly instead of including its server inline in the `paste.script` package. You must install CherryPy before using `egg:PasteScript#cherrypy`

1.7 1.7.4.2

- 1.7.4 had package release problems, was reverted; 1.7.4.1 also had package problems.
- Include special `here` and `__file__` default vars for logging config files, similar to PasteDeploy config loading.
- Allow Jython to import various bits from `paste.script.command` and `paste.script.copydir` without throwing an import error (subprocess module cannot be imported on Jython). This allows PasteScript to work minimally on Jython, although execution will fail for `command.run_command` and `copydir.copydir`.

1.8 1.7.3

- CherryPy wsgiserver updated to 3.1.1, fixes a regression in Python 2.5 plus a couple other small fixes.

1.9 1.7.2

- Fix a packaging issue that could cause problems when installing PasteScript.

1.10 1.7.1

- filemaker.py's FileOp can now handle a tuple as a source_dir argument that should function the same as the _template_dir option for pkg_resources.
- CherryPy wsgiserver updated to trunk@2063 for Python 2.6 compatibility.

1.11 1.7

- _template_dir now takes a tuple argument, which should be the package name, and the relative location to the package of its template directory. pkg_resources will then be used to load make the templates rather than raw file access making it zip-safe.
- CherryPy wsgiserver updated to the 3.1.0 release's.
- Support Python 2.6.
- Added experimental support for a quicker paster serve –reload for Jython.
- Non-Python files in `paste/script/templates/` causes an error on 2.6; renamed directory to avoid this.

1.12 1.6.3

- Fixes for `paste.script.filemaker`
- A setuptools `egg_info.writers` entry point is now provided that's responsible for writing paster_plugins.txt for projects that define a new `paster_plugins` setup() keyword. paster_plugins.txt will still be created for new projects that need it and lack a `paster_plugins` setup() keyword, but this is deprecated. Projects defining `paster_plugins` in setup() should also define a `setup_requires` setup() keyword including PasteScript.
- An `egg_plugins` variable (a list of strings based off the Templates classes' `egg_plugins` variables) is now available to paster create templates for the new `paster_plugins` setup() keyword.
- PasteScript is no longer included in `egg_plugins`/paster_plugins.txt by default.

1.13 1.6.2

- Fix SkipTemplate (could raise `TypeError: argument 1 must be string or read-only buffer, not None` before)

1.14 1.6.1 (and 1.6.1.1)

- Fix paster serve under Windows.

1.15 1.6

- Added commands `paster request config.ini URL` and `paster post config.ini URL < post-body`, that allow you to do artificial requests to applications.

- Check the writability of the pid and log files earlier. This caused particular problems if you started it in daemon mode, and the files weren't writable. From Chris Atlee.
- Start the monitor (when using `--monitor`) after daemonizing, so that `paster serve --monitor --daemon` works (before it would constantly restart).
- In Paste Script templates, you can give `should_echo=False` in variable definitions, and if the user is queried for the variable then the input will not be echoed (as for a password). From Dirceu Pereira Tiegs.
- Added a method `paste.script.appinstall.Installer.template_renderer`, which can be used to override template substitution with `paster make-config`. The function is similar to the same function used with `paster create` templates.
- Remove `--daemon` option from Windows, as it depends on `os.fork`
- When using `paster create` and inserting text with a `-*-` marker, multi-line text will no longer be reinserted.
- Improved output when skipping templates with `paster create`.
- When starting a server with `paster serve --daemon` and the pid file exists and describes a running process, do not start another process.
- Added `umask` option to `egg:PasteScript#flup_fcgi_thread/fork`.
- Deprecate the flup entry points, as flup now has the necessary entry points in its own package.

1.16 1.3.6

- CherryPy wsgiserver updated to the 3.0.2 release's
- paster no longer hides `pkg_resources.DistributionNotFound` error messages describing the target project's requirements. Aids the somewhat confusing "did you run setup.py develop?" message when it had already been ran, but since then had a requirement added that wasn't installed.
- Logging configuration is now read from the config file during `paster setup-app`.
- Custom Formatters and Handlers (Handlers outside of the logging module) are now supported in logging configuration files.

1.17 1.3.5

- Initialize logging earlier in the serve command for components that want to utilize it. Patch from Christopher Lenz.
- Fixed Python 2.3 incompatibility (no `string.Template`) in `paste.script.appinstall`.

1.18 1.3.4

- Make sure that when using `--monitor` or `--reload`, if the parent monitoring process dies, also kill the subprocess.
- When using `paster serve --log-file`, append to the log file (was truncating any previous contents).
- Read logging information from the server file, using the logging module's standard configuration format

- When adding files don't fail because an svn command fails. Helpful particularly on Windows, when the svn command-line isn't installed (e.g., using TortoiseSVN).

1.19 1.3.3

- Fixed problem with `paster serve` on Windows. Also on Windows, fixed issue with executables with spaces in their names (this case requires the `win32all` module).
- You can use `+dot+` in your project template filenames, specifically so that you can use leading dots in the filename. Usually leading dots cause the file to be ignored. So if you want to have new projects contain a `.cvsignore` file, you can put a `+dot+cvsignore` file in your template.
- Relatedly, `+plus+` has been added so you can include pluses.

1.20 1.3.2

- `paster` was largely broken under Windows; fixed.

1.21 1.3.1

- Fix related to Python 2.5 (when there are errors creating files, you could get infinite recursion under Python 2.5).
- Use `subprocess` module in `paster serve` command. Added `--monitor` option which will restart the server if it exits.
- The `exe` command now does % substitution in keys (e.g., `pid_file=%(here)s/paste.pid`).
- Some import problems with Cheetah should be improved.

1.22 1.3

- Fixed an exception being raised when shutting down flup servers using sockets.
- Fixed the CherryPy 3 WSGI server entry point's handling of SIGHUP and SIGTERM.
- The CherryPy wsgiserver is now available at `paste.script.wsgiserver` (no longer requiring CherryPy to be installed).
- Added entry point for twisted server.
- Made `paste.script.pluginlib:egg_info_dir` work with packages that put the `Package.egg-info/` directory in a subdirectory (typically `src/`).
- Remove Cheetah requirement. Packages using Cheetah templates should require Cheetah themselves. If you are using `paster make-config` and you *don't* want to use Cheetah, you must add `use_cheetah = False` to your `Installer` subclass (it defaults to true for backward compatibility).
- Make scripts work when there is no `setup.py` (if you aren't making a Python/setuptools package).
- When using `paste.script.copydir.copy_dir` (as with most `paster create` templates), you can raise `SkipTemplate` (or call the `skip_template()` function) which will cause the template to be skipped. You can use this to conditionally include files.

- When using `paster serve c:/...`, it should no longer confuse `c:` with a scheme (such as `config:` or `egg:`).
- More careful about catching import errors in `websetup`, so if you have a bug in your `app.websetup` module it won't swallow it.

1.23 1.1

- Added a `warn_returncode` option to `Command.run_command`, and make `ensure_file` use this for its svn command.
- If the svn command-line program isn't working for you, commands that use `ensure_file` will continue to work but just with a warning.
- Removed copyright notice that was accidentally included in new packages created by `paster create`.
- Allow variable assignments at the end of `paster serve`, like `paster serve http_port=80`; then you can use `%(http_port)s` in your config files (requires up-to-date Paste Deploy).
- Allow a `package_dir` variable so that you can put your package into subdirectories of the base directory (e.g., `src/`).
- Changes to the `twisted.web2` wrapper (from pythy).
- Warn when you run `paster setup-app` and no modules are listed in `top_level.txt` (which can happen with a newly checked out project).

1.24 1.0

- Added entry point for CherryPy 3's WSGI server.
- Fixed `paster serve` to hide KeyboardInterrupt (CTRL-C) tracebacks in `--reload` mode.
- Added `template_renderer` argument to `paste.script.copydir.copydir`. This allows you to use arbitrary template languages, instead of just `string.Template` and Cheetah.

1.25 0.9.9

- egg:PasteScript#test (the paste.script.testapp) now accepts `lint` and `text` boolean configuration. `lint` will turn on `paste.lint` validation. `text` will cause it to return a simple text/plain response with the environ, instead of an HTML table.
- Improvements all around to `paster points`, plus documentation for existing entry point groups.

1.26 0.9.8

- New projects will now ignore `Package.egg-info/dependency_links.txt`, just like all the other derivative files in the `egg-info` directory
- `paster serve --reload` was broken on Windows when the Python executable was in a directory with spaces in it. This is probably a bug in the `subprocess` module.

1.27 0.9.7

- Update to filemaker commands to take optional argument so that when new directories are for a Python package, they will have a __init__.py created as well.

1.28 0.9.6

- Do all variable assignment during package creation up-front, before actually creating the files.
- Added the `egg` template variable: provides projects with a safe egg name as generated by setuptools. This should be used for egg-info directories in templates (e.g. `+egg+.egg-info` instead of `+project+.egg-info`), and anywhere else the egg name is expected, to prevent errors with project names containing hyphens.

1.29 0.9

- Installer calls `websetup.setup_app(command, conf, vars)`; `setup_config()` will be deprecated in the future
- Added copyright information
- `paster serve config.ini#section` works now
- `paster make-config/setup-app` will read `$PASTE_SYSCONFIG` to find extra `sysconfig.py` files.
- `paster create` will now query interactively for variables if they aren't explicitly provided.

1.30 0.5

- If the output directory doesn't exist when running `paster create`, do not default to having interactive (confirmation) on.

1.31 0.4.2

- Fixed the Flup FastCGI interface. (There seem to still be problems with forking FastCGI.)
- The `prepare-app` command has been renamed `make-config`
- Changed the way `make-config` and `setup-app` use `sysconfig` – these are now modules that can define various functions
- Allow for default config file names
- Consider config generation that may produce a directory (this case is now generally workable)
- Allow for multiple config files (specifically with –edit)
- Give config file generation the variables `app_install_uuid` and `app_install_secret` that they can use for their config files
- Include Ka-Ping Yee's uuid module in `paste.script.util.uuid`

- `paster help` doesn't bail when commands can't be loaded
- Be a little safer when `--edit` fails and `--setup` is provided (don't automatically set up if the edit seems to have failed)

1.32 0.4.1

- Two small bugfixes, one related to the `basic_package` template (it included a reference to `finddata`, which it should not have), and a fix to how the `.egg-info` directory is determined.

1.33 0.4

- Added `points` command, for entry-point related queries.
- paste.deploy config files that start with `#!/usr/bin/env paster` can make a script into an executable.
- Improvements to `paster serve` command:
  - Handle bad PID files better
  - Daemonization is more reliable
  - Allow `start`, `stop`, `restart` instead of just options
- Improvements to `paster create` command:
  - Invoked interactively by default (so that you are warned before overwriting files)
- Added new commands:
  - `points` for viewing Egg entry point information
  - `prepare-app` and `setup-app` for installing web applications
- Fixed bug in how Egg distributions are loaded.

1.34 0.3.1

- Fixed small bug with running `paster serve` on Windows. (Small to fix, kept script from running on Windows entirely).

1.35 0.3

Initial release.

CHAPTER 2

Paste Script: Development

author Ian Bicking <ianb@colorstudy.com>

revision Rev

date $LastChangedDate$

Contents

2.1 Introduction

This document is an introduction to how you can extend `paster` and Paste Script for your system – be it a framework, server setup, or whatever else you want to do.

2.2 What Paste Script Can Do

`paster` is a two-level command, where the second level (e.g., `paster help`, `paster create`, etc) is pluggable.

Commands are attached to Python Eggs, i.e., to the package you distribute and someone installs. The commands are identified using entry points.

To make your command available do something like this in your `setup.py` file:

```
from setuptools import setup
setup(...
    entry_points="""
    [paste.paster_command]
    mycommand = mypackage.mycommand:MyCommand

    [paste.global_paster_command]
    myglobal = mypackage.myglobal:MyGlobalCommand
    """)
```

This means that `paster mycommand` will run the `MyCommand` command located in the `mypackage.mycommand` module. Similarly with `paster myglobal`. The distinction between these two entry points is that the first will only be usable when `paster` is run inside a project that is identified as using your project, while the second will be globally available as a command as soon as your package is installed.

2.3 How's the Local Thing Work?

So if you have a local command, how does it get enabled? If the person is running `paster` inside their project directory, `paster` will look in `Project_Name.egg-info/paster_plugins.txt` which is a list of project names (the name of your package) whose commands should be made available.

This is for frameworks, so frameworks can add commands to `paster` that only apply to projects that use that framework.

2.4 What Do Commands Look Like?

The command objects (like `MyCommand`) are subclasses of `paste.script.command.Command`. You can look at that class to get an idea, but a basic outline looks like this:

```
from paste.script import command

class MyCommand(command.Command):

    max_args = 1
    min_args = 1

    usage = "NAME"
    summary = "Say hello!"
    group_name = "My Package Name"

    parser = command.Command.standard_parser(verbose=True)
    parser.add_option('--goodbye',
                      action='store_true',
                      dest='goodbye',
                      help="Say 'Goodbye' instead")

    def command(self):
        name = self.args[0]
        if self.verbose:
            print "Got name: %r" % name
        if self.options.goodbye:
```

(continues on next page)

(continued from previous page)

```
            print "Goodbye", name
        else:
            print "Hello", name
```

`max_args` and `min_args` are used to give error messages. You can also raise `command.BadCommand(msg)` if the arguments are incorrect in some way. (Use `None` here to give no restriction)

The `usage` variable means `paster mycommand -h` will give a usage of `paster mycommand [options] NAME`. `summary` is used with `paster help` (describing your command in a short form). `group_name` is used to group commands together for `paste help` under that title.

The `parser` object is an *optparse <http://python.org/doc/current/lib/module-optparse.html>* `OptionParser` object. `Command.standard_parser` is a class method that creates normal options, and enables options based on these keyword (boolean) arguments: `verbose`, `interactive`, `no_interactive` (if interactive is the default), `simulate`, `quiet` (undoes verbose), and `overwrite`. You can create the parser however you want, but using `standard_parser()` encourages a consistent set of shared options across commands.

When your command is run, `.command()` is called. As you can see, the options are in `self.options` and the positional arguments are in `self.args`. Some options are turned into instance variables – especially `self.verbose` and `self.simulate` (even if you haven't chosen to use those options, many methods expect to find some value there, which is why they are turned into instance variables).

There are quite a few useful methods you can use in your command. See the Command class for a complete list. Some particulars:

`run_command(cmd, arg1, arg2, ..., cwd=os.getcwd(), capture_stderr=False):`

> Runs the command, respecting verbosity and simulation. Will raise an error if the command doesn't exit with a 0 code.

`insert_into_file(filename, marker_name, text, indent=False):`

> Inserts a line of text into the file, looking for a marker like `-*- marker_name -*-` (and inserting just after it). If `indent=True`, then the line will be indented at the same level as the marker line.

`ensure_dir(dir, svn_add=True):`

> Ensures that the directory exists. If `svn_add` is true and the parent directory has an `.svn` directory, add the new directory to Subversion.

`ensure_file(filename, content, svn_add=True):`

> Ensure the file exists with the given content. Will ask the user before overwriting a file if `--interactive` has been given.

2.5 Templates

The other pluggable part is "templates". These are used to create new projects. Paste Script includes one template itself: `basic_package` which creates a new setuptools package.

To enable, add to `setup.py`:

```
setup(...
    entry_points="""
    [paste.paster_create_template]
    framework = framework.templates:FrameworkTemplate
    """)
```

FrameworkTemplate should be a subclass of paste.script.templates.Template. An easy way to do this is simply with:

```
from paste.script import templates

class FrameworkTemplate(templates.Template):

    egg_plugins = ['Framework']
    summary = 'Template for creating a basic Framework package'
    required_templates = ['basic_package']
    _template_dir = 'template'
    use_cheetah = True
```

egg_plugins will add Framework to paste_plugins.txt in the package. required_template means those template will be run before this one (so in this case you'll have a complete package ready, and you can just write your framework files to it). _template_dir is a module-relative directory to find your source files.

The source files are just a directory of files that will be copied into place, potentially with variable substitutions. Three variables are expected: project is the project name (e.g., Project-Name), package is the Python package in that project (e.g., projectname) and egg is the project's egg name as generated by setuptools (e.g., Project_Name). Users can add other variables by adding foo=bar arguments to paster create.

Filenames are substituted with +var_name+, e.g., +package+ is the package directory.

If a file in the template directory ends in _tmpl then it will be substituted. If use_cheetah is true, then it's treated as a Cheetah template. Otherwise string.Template is used, though full expressions are allowed in ${expr} instead of just variables.

See the templates module for more.

CHAPTER 3

Warning

Paste Script is being maintained on life support. That means that critical bugs will be fixed, and support for new versions of Python will be handled, but other than that new features are not being considered. Development has moved to GitHub.

CHAPTER 4

Introduction

If you are developer, see the Developer Documentation; this will tell you how to add commands and templates to `paster`. For a list of updates see the news file.

Paste Script is released under the MIT license.

CHAPTER 5

Status

Paste Script has passed version 1.0. Paste Script is in maintenance mode. Bugs will be fixed, new features are not being considered.

CHAPTER 6

paster create

This creates the skeleton for new projects. Many different kinds of projects have created skeletons for their projects (Pylons, TurboGears, ZopeSkel, and others).

For a tutorial for making new skeletons, see this tutorial from Lucas Szybalski. It also discusses creating new subcommands for paster.

CHAPTER 7

paster serve

The one useful command you may want to know about for `paster` is `paster serve`. This serves an application described in a Paste Deploy configuration file.

CHAPTER 8

Configuration

A quickstart (and example), if not complete explanation:

```
[app:main]
use = egg:PasteEnabledPackage
option1 = foo
option2 = bar

[server:main]
use = egg:PasteScript#wsgiutils
host = 127.0.0.1
port = 80
```

`egg:PasteEnabledPackage` refers to some package that has been prepared for use with paste.deploy, and options given to that package. If you are starting out with some framework, you'll have to reference some documentation for that framework to paste-deploy-ify your application (or read the paste.deploy documentation).

In the same file is a server description. `egg:PasteScript#wsgiutils` is a server (named `wsgiutils`) provided by this package, based on WSGIUtils. And we pass various options particular to that server.

Other packages can provide servers, but currently Paste Script includes glue for these:

`wsgiutils`:

> A SimpleHTTPServer based threaded HTTP server, using WSGIUtils.

`flup_(scgi|fcgi|ajp)_(thread|fork)`:

> This set of servers supports SCGI, FastCGI and AJP protocols, for connection an external web server (like Apache) to your application. Both threaded and forking versions are available. This is based on flup.

There is the start of support for twisted.web2 in `paste.script.twisted_web2_server`; patches welcome.

CHAPTER 9

Running the Server

`paster serve --help` gives useful output:

```
usage: /usr/local/bin/paster serve [options] CONFIG_FILE [start|stop|restart|status]
Serve the described application

If start/stop/restart is given, then it will start (normal
operation), stop (--stop-daemon), or do both.  You probably want
``--daemon`` as well for stopping.

Options:
  -h, --help            show this help message and exit
  -v, --verbose
  -q, --quiet
  -nNAME, --app-name=NAME
                        Load the named application (default main)
  -sSERVER_TYPE, --server=SERVER_TYPE
                        Use the named server.
  --server-name=SECTION_NAME
                        Use the named server as defined in the configuration
                        file (default: main)
  --daemon              Run in daemon (background) mode
  --pid-file=FILENAME   Save PID to file (default to paster.pid if running in
                        daemon mode)
  --log-file=LOG_FILE   Save output to the given log file (redirects stdout)
  --reload              Use auto-restart file monitor
  --reload-interval=RELOAD_INTERVAL
                        Seconds between checking files (low number can cause
                        significant CPU usage)
  --status              Show the status of the (presumably daemonized) server
  --user=USERNAME       Set the user (usually only possible when run as root)
  --group=GROUP         Set the group (usually only possible when run as root)
  --stop-daemon         Stop a daemonized server (given a PID file, or default
                        paster.pid file)
```

Basically you give it a configuration file. If you don't do anything else, it'll serve the `[app:main]` application with

the [server:main] server. You can pass in --server-name=foo to serve the [server:foo] section (or even --server-name=config:foo.ini to use a separate configuration file).

Similarly you can use --app-name=foo to serve [app:foo].

--daemon will run the server in the backgroup, --user and --group will set the user, as you might want to do from a start script (run as root). If you don't give a --pid-file it will write the pid to paster.pid (in the current directory).

--stop-daemon will stop the daemon in paster.pid or whatever --pid-file you give. --log-file will redirect stdout and stderr to that file.

--reload will start the reload monitor, and restart the server whenever a file is edited. This can be a little expensive, but is very useful during development.

CHAPTER 10

#! Scripts

On Posix (Linux, Unix, etc) systems you can turn your configuration files into executable scripts.

First make the file executable (`chmod +x config_file.ini`). The you should add a line like this to the top of the file:

```
#!/usr/bin/env paster
```

You can include a command and command-line options in an `[exe]` section, like:

```
[exe]
command = serve
daemon = true
user = nobody
group = nobody
```

(use `true` and `false` for options that don't take an argument).

If you use `daemon = true` then you'll be able to use the script as an rc script, so you can do:

```
$ sudo ./config_file.ini start
$ sudo ./config_file.ini restart
```

and so forth.

Note that this is a little wonky still on some platforms and shells (notably it doesn't work under csh). If you get an error about "Command config_file.ini not known" then this probably won't work for you. In the future an additional script to `paster` will be added just for this purpose.