Patchwork

Release

Jun 20, 2018

Contents

Patchwork is a mid-level library of Unix system administration primitives such as "install package" or "create user account", interrogative functionality for introspecting system state, and other commonly useful functions built on top of the Fabric library.

Specifically:

- Primary API calls strive to be **idempotent**: they may be called multiple times in a row without unwanted changes piling up or causing errors.
- Patchwork **is just an API**: it has no concept of "recipes", "manifests", "classes", "roles" or other high level organizational units. This is left up to the user or wrapping libraries.
  - This is one way Patchwork differs from larger configuration management frameworks like Chef or Puppet. Patchwork is closest in nature to those tools' "resources."
- It is implemented in **shell calls**, typically sent **over SSH** from a local workstation.
  - However, where possible, its functions expect a baseline Invoke `Context` object and can thus run locally *or* remotely, depending on the specific context supplied by the caller.

CHAPTER 1

Roadmap

While Patchwork has been released with a major version number to signal adherence to semantic versioning, it's still early in development and has not fully achieved its design vision yet.

We expect it to gain maturity in tandem with the adoption and development of Fabric 2. It's also highly likely that Patchwork will see a few major releases as its API (and those of its sister library, invocations) matures.

CHAPTER 2

Contents

2.1 Changelog

- #17: Missed some `.succeeded` attributes when porting to Fabric 2 - they should have been `.ok`. Patch via Lucio Delelis.
- : Fix a bug in *`patchwork.transfers.rsync`* where it would fail with `AttributeError` unless your connection had `connect_kwargs.key_filename` defined.
- #20: (via #21) *`patchwork.transfers.rsync`* didn't handle its `exclude` parameter correctly when building the final `rsync` command (it came out looking like a stringified tuple). Fixed by Kegan Gan.
- #23: Fix some outstanding Python 2-isms (use of `iteritems`) in *`info.distro_name`* and *`info.distro_family`*, as well as modules which imported those – such as *`packages`*.

  Includes adding some basic tests for this functionality as well. Thanks to `@ChaoticMind` for the report.
- : Pre-history.

CHAPTER 3

API documentation

3.1 environment

Shell environment introspection, e.g. binaries in effective $PATH, etc.

patchwork.environment.**have_program**(*c*, *name*)
Returns whether connected user has program `name` in their `$PATH`.

3.2 files

Tools for file and directory management.

patchwork.files.**append**(**args*, ***kwargs*)
Append string (or list of strings) `text` to `filename`.

When a list is given, each string inside is handled independently (but in the order given.)

If `text` is already found in `filename`, the append is not run, and None is returned immediately. Otherwise, the given text is appended to the end of the given `filename` via e.g. `echo '$text' >> $filename`.

The test for whether `text` already exists defaults to a full line match, e.g. `^<text>$`, as this seems to be the most sensible approach for the "append lines to a file" use case. You may override this and force partial searching (e.g. `^<text>`) by specifying `partial=True`.

Because `text` is single-quoted, single quotes will be transparently backslash-escaped. This can be disabled with `escape=False`.

patchwork.files.**contains**(**args*, ***kwargs*)
Return True if `filename` contains `text` (which may be a regex.)

By default, this function will consider a partial line match (i.e. where `text` only makes up part of the line it's on). Specify `exact=True` to change this behavior so that only a line containing exactly `text` results in a True return value.

This function leverages `egrep` on the remote end (so it may not follow Python regular expression syntax perfectly), and skips the usual outer `env.shell` wrapper that most commands execute with.

If `escape` is False, no extra regular expression related escaping is performed (this includes overriding `exact` so that no `^/$` is added.)

`patchwork.files.`**`directory`**(**args*, ***kwargs*)
Ensure a directory exists and has given user and/or mode

`patchwork.files.`**`exists`**(**args*, ***kwargs*)
Return True if given path exists on the current remote host.

3.3 info

Functions for interrogating a system to determine general attributes.

Specifically, anything that doesn't fit better in other modules where we also manipulate this information (such as *packages*). For example, detecting operating system family and version.

`patchwork.info.`**`distro_family`**(*c*)
Returns basic "family" ID for the remote system's distribution.

Currently, options include:

- `debian`
- `redhat`

If the system falls outside these categories, its specific family or release name will be returned instead.

`patchwork.info.`**`distro_name`**(*c*)
Return simple Linux distribution name identifier, e.g. `"ubuntu"`.

Uses tools like `/etc/issue`, and `lsb_release` and fits the remote system into one of the following:

- `fedora`
- `rhel`
- `centos`
- `ubuntu`
- `debian`
- `other`

3.4 packages

Management of various (usually binary) package types - OS, language, etc.

`patchwork.packages.`**`package`**(*c*, **packages*)
Installs one or more `packages` using the system package manager.

Specifically, this function calls a package manager like `apt-get` or `yum` once per package given.

`patchwork.packages.`**`rubygem`**(*c*, *gem*)
Install a Ruby gem.

3.5 transfers

File transfer functionality above and beyond basic `put`/`get`.

`patchwork.transfers.`**`rsync`**(*c*, *source*, *target*, *exclude=()*, *delete=False*, *strict_host_keys=True*, *rsync_opts=''*, *ssh_opts=''*)

Convenient wrapper around your friendly local `rsync`.

Specifically, it calls your local `rsync` program via a subprocess, and fills in its arguments with Fabric's current target host/user/port. It provides Python level keyword arguments for some common rsync options, and allows you to specify custom options via a string if required (see below.)

For details on how `rsync` works, please see its manpage. `rsync` must be installed on both the invoking system and the target in order for this function to work correctly.

Note: This function transparently honors the given `Connection`'s connection parameters such as port number and SSH key path.

Note: For reference, the approximate `rsync` command-line call that is constructed by this function is the following:

```
rsync [--delete] [--exclude exclude[0][, --exclude[1][, ...]]] \
    -pthrvz [rsync_opts] <source> <host_string>:<target>
```

Parameters

- **`c`** – `Connection` object upon which to operate.
- **`source`** (*`str`*) – The local path to copy from. Actually a string passed verbatim to `rsync`, and thus may be a single directory (`"my_directory"`) or multiple directories (`"dir1 dir2"`). See the `rsync` documentation for details.
- **`target`** (*`str`*) – The path to sync with on the remote end. Due to how `rsync` is implemented, the exact behavior depends on the value of `source`:
  - If `source` ends with a trailing slash, the files will be dropped inside of `target`. E.g. `rsync(c, "foldername/", "/home/username/project")` will drop the contents of `foldername` inside of `/home/username/project`.
  - If `source` does **not** end with a trailing slash, `target` is effectively the "parent" directory, and a new directory named after `source` will be created inside of it. So `rsync(c, "foldername", "/home/username")` would create a new directory `/home/username/foldername` (if needed) and place the files there.
- **`exclude`** – Optional, may be a single string or an iterable of strings, and is used to pass one or more `--exclude` options to `rsync`.
- **`delete`** (*`bool`*) – A boolean controlling whether `rsync`'s `--delete` option is used. If True, instructs `rsync` to remove remote files that no longer exist locally. Defaults to False.
- **`strict_host_keys`** (*`bool`*) – Boolean determining whether to enable/disable the SSH-level option `StrictHostKeyChecking` (useful for frequently-changing hosts such as virtual machines or cloud instances.) Defaults to True.
- **`rsync_opts`** (*`str`*) – An optional, arbitrary string which you may use to pass custom arguments or options to `rsync`.

- **ssh_opts** (*str*) – Like `rsync_opts` but specifically for the SSH options string (rsync's `--rsh` flag.)

Python Module Index

p

Index