Invocations

Release

Aug 28, 2021

Contents

Invocations is a collection of reusable Invoke tasks, task collections and helper functions. Originally sourced from the Invoke project's own project-management tasks file, they are now highly configurable and used across a number of projects, with the intent to become a clearinghouse for implementing common best practices.

Currently implemented topics include (but are not limited to):

- management of Sphinx documentation trees
- Python project release lifecycles
- dependency vendoring
- running test suites (unit, integration, coverage-oriented, etc)
- console utilities such as confirmation prompts

and more.

CHAPTER 1

Roadmap

While Invocations has been released with a major version number to signal adherence to semantic versioning, it's somewhat 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 Invoke post-1.x. It's also highly likely that Invocations will see a few major releases as its API (and those of its sister library, patchwork) matures.

CHAPTER 2

Contents

2.1 Changelog

- : Added `twine check` (which validates packaging metadata's `long_description`) as a pre-upload step within `packaging.release.publish`.
  - This includes some tweaking of `readme_renderer` behavior (used internally by twine) so it correctly spots more malformed RST, as Sphinx does.
- : Add `packaging.release.push` for pushing Git objects as part of a release.
- : The `packaging.release.all_` task has been expanded to actually do "ALL THE THINGS!!!", given a `dry_run` flag, and renamed on the CLI to `all` (no trailing underscore).
- : `packaging.release.prepare` grew a `dry_run` flag to match the rest of its friends.
- : Add Codecov support to `pytest.coverage`.
- : Add `packaging.release.test_install` task and call it just prior to the final step in `packaging.release.upload` (so one doesn't upload packages which build OK but don't actually install OK).
- : `pytest.coverage` incorrectly concatenated its `opts` argument to internal options; this has been fixed.
- : Correctly test for `html` report type inside of `pytest.coverage` when deciding whether to run `open` at the end.
- : `packaging.release.publish` missed a spot when it grew "kwargs beat configuration" behavior - the `index` kwarg still got overwritten by the config value, if defined. This has been fixed.
- : `packaging.release.prepare` now generates annotated Git tags instead of lightweight ones. This was a perplexing oversight (Git has always intended annotated tags to be used for release purposes) so we're considering it a bugfix instead of a backwards incompatible feature change.
- : `packaging.release.prepare` now runs its internal status check twice, once at the start (as before) and again at the end (to prove that the actions taken did in fact satisfy needs).
- : Rely on Invoke 1.6+ for some of its new features.

- : Add a `warnings` kwarg/flag to `pytest.test`, allowing one to call it with `--no-warnings` as an inline 'alias' for pytest's own `--disable-warnings` flag.
- : Fix minor display bug causing the `pytest` task module to append a trailing space to the invocation of pytest itself.
- : `release.build` and `release.publish` had bad kwargs-vs-config logic preventing flags such as `--wheel` or `--python` from actually working (config defaults always won out, leading to silent ignoring of user input). This has been fixed; config will now only be honored unless the CLI appears to be overriding it.
- : `release.build`'s `--clean` flag has been updated:
  - It now honors configuration like the other flags in this task, specifically `packaging.clean`.
  - It now defaults to `False` (rationale: most build operations in the wild tend to assume no cleaning by default, so defaulting to the opposite was sometimes surprising).

    Warning: This is a backwards incompatible change.

  - When `True`, it applies to both build and dist directories, instead of just build.

    Warning: This is a backwards incompatible change.

- : Modify `release` task tree to look at `main` branches in addition to `master` ones, for "are we on a feature release line or a bugfix one?" calculations, etc.
- : Replace some old Python 2.6-compatible syntax bits.
- : Reverse the default value of `release.build` and `release.publish`)'s `wheel` argument from `False` to `True`. Included in this change is a new required runtime dependency on the `wheel` package.

  Rationale: at this point in time, most users will be expecting wheels to be available, and not building wheels is likely to be the uncommon case.

  Warning: This is a backwards incompatible change.

- #21: Only require `enum34` under Python 2 to prevent it clashing with the stdlib `enum` under Python 3. Credit: Alex Gaynor.
- #12: Upgrade our packaging manifest so tests (also docs, requirements files, etc) are included in the distribution archives. Thanks to Tomáš Chvátal for the report.
- : Drop Python 3.4 support. We didn't actually do anything to make the code not work on 3.4, but we've removed some 3.4 related runtime (and development) dependency limitations. Our CI will also no longer test on 3.4.

  Warning: This is technically a backwards incompatible change.

- : Add a `find_opts` argument to `checks.blacken` for improved control over what files get blackened.
- : `checks.blacken` had a typo regarding its folder selection argument; the CLI/function arg was `folder` while the configuration value was `folders` (plural). It's been made consistent: the CLI/function argument is now `folders`.
- : Was missing a 'hide output' flag on a subprocess shell call, the result of which was mystery git branch names appearing in the output of `inv release` and friends. Fixed now.

- : `checks.blacken` had a typo regarding its folder selection argument; the CLI/function arg was `folder` while the configuration value was `folders` (plural). It's been made consistent: the CLI/function argument is now `folders`.
- : Was missing a 'hide output' flag on a subprocess shell call, the result of which was mystery git branch names appearing in the output of `inv release` and friends. Fixed now.
- : `checks.blacken` had a typo regarding its folder selection argument; the CLI/function arg was `folder` while the configuration value was `folders` (plural). It's been made consistent: the CLI/function argument is now `folders`.
- : Was missing a 'hide output' flag on a subprocess shell call, the result of which was mystery git branch names appearing in the output of `inv release` and friends. Fixed now.
- : Remove some apparently non-functional `setup.py` logic around conditionally requiring `enum34`; it was never getting selected and thus breaking a couple modules that relied on it.

  `enum34` is now a hard requirement like the other semi-optional-but-not-really requirements.
- : `checks.blacken` had a typo regarding its folder selection argument; the CLI/function arg was `folder` while the configuration value was `folders` (plural). It's been made consistent: the CLI/function argument is now `folders`.
- : Was missing a 'hide output' flag on a subprocess shell call, the result of which was mystery git branch names appearing in the output of `inv release` and friends. Fixed now.
- : Remove some apparently non-functional `setup.py` logic around conditionally requiring `enum34`; it was never getting selected and thus breaking a couple modules that relied on it.

  `enum34` is now a hard requirement like the other semi-optional-but-not-really requirements.
- : Bump Releases requirement up to 1.6 and leverage its new ability to load Sphinx extensions, in `packaging.release.prepare` (which parses Releases changelogs programmatically). Prior to this, projects which needed extensions to build their doctree would throw errors when using the `packaging.release` module.
- : Remove some apparently non-functional `setup.py` logic around conditionally requiring `enum34`; it was never getting selected and thus breaking a couple modules that relied on it.

  `enum34` is now a hard requirement like the other semi-optional-but-not-really requirements.
- : Remove some apparently non-functional `setup.py` logic around conditionally requiring `enum34`; it was never getting selected and thus breaking a couple modules that relied on it.

  `enum34` is now a hard requirement like the other semi-optional-but-not-really requirements.
- : Add the `checks` module, containing `checks.blacken` which executes the black code formatter. Thanks to Chris Rose.
- : Add 'missing' arguments to `pytest.integration` so its signature now largely matches `pytest.test`, which it wraps.
- : Break out a generic form of the `travis.sudo-coverage` task into `travis.sudo-run` which can be used for arbitrary commands run under the ssh/sudo capable user generated by `travis.make-sudouser`/`travis.make-sshable`.
- : Add `travis.blacken` which wraps the new `checks.blacken` (in diff+check mode, for test output useful for users who cannot themselves simply run black) in addition to performing Travis-oriented Python version checks and pip installation.

  This is necessary to remove boilerplate around the fact that `black` is not even visible to Python versions less than 3.6.
- : Split out the body of the (sadly incomplete) `packaging.release.all` task into the better-named `packaging.release.prepare`. (`all` continues to behave as it did, it just now calls `prepare` explicitly.)

- : Pre-history / code primarily for internal consumption

CHAPTER 3

API/task docs

3.1 autodoc

Sphinx autodoc hooks for documenting Invoke-level objects such as tasks.

Unlike most of the rest of Invocations, this module isn't for reuse in the "import and call functions" sense, but instead acts as a Sphinx extension which allows Sphinx's autodoc functionality to see and document Invoke tasks and similar Invoke objects.

Note: This functionality is mostly useful for redistributable/reusable tasks which have been defined as importable members of some Python package or module, as opposed to "local-only" tasks that live in a single project's `tasks.py`.

However, it will work for any tasks that Sphinx autodoc can import, so in a pinch you could for example tweak `sys.path` in your Sphinx `conf.py` to get it loading up a "local" tasks file for import.

To use:

- Add `"sphinx.ext.autodoc"` and `"invocations.autodoc"` to your Sphinx `conf.py`'s `extensions` list.
- Use Sphinx autodoc's `automodule` directive normally, aiming it at your tasks module(s), e.g. `.. automodule:: myproject.tasks` in some `.rst` document of your choosing.
  - As noted above, this only works for modules that are importable, like any other Sphinx autodoc use case.
  - Unless you want to opt-in which module members get documented, use `:members:` or add `"members"` to your `conf.py`'s `autodoc_default_flags`.
  - By default, only tasks with docstrings will be picked up, unless you also give the `:undoc-members:` flag or add `:undoc-members:` / add `"undoc-members"` to `autodoc_default_flags`.
  - Please see the autodoc docs for details on these settings and more!
- Build your docs, and you should see your tasks showing up as documented functions in the result.

3.2 console

Text console UI helpers and patterns, e.g. 'Y/n' prompts and the like.

invocations.console.**confirm**(*question*, *assume_yes=True*)

Ask user a yes/no question and return their response as a boolean.

`question` should be a simple, grammatically complete question such as "Do you wish to continue?", and will have a string similar to `" [Y/n] "` appended automatically. This function will *not* append a question mark for you.

By default, when the user presses Enter without typing anything, "yes" is assumed. This can be changed by specifying `assume_yes=False`.

Note: If the user does not supply input that is (case-insensitively) equal to "y", "yes", "n" or "no", they will be re-prompted until they do.

Parameters

- **question** (*str*) – The question part of the prompt.
- **assume_yes** (*bool*) – Whether to assume the affirmative answer by default. Default value: `True`.

Returns A `bool`.

3.3 docs

Tasks for managing Sphinx documentation trees.

invocations.docs.**build**(*c*, *clean=False*, *browse=False*, *nitpick=False*, *opts=None*, *source=None*, *target=None*)

Build the project's Sphinx docs.

invocations.docs.**doctest**(*c*)

Run Sphinx' doctest builder.

This will act like a test run, displaying test results & exiting nonzero if all tests did not pass.

A temporary directory is used for the build target, as the only output is the text file which is automatically printed.

invocations.docs.**sites**(*c*)

Build both doc sites w/ maxed nitpicking.

invocations.docs.**tree**(*c*)

Display documentation contents with the 'tree' program.

invocations.docs.**watch_docs**(*c*)

Watch both doc trees & rebuild them if files change.

This includes e.g. rebuilding the API docs if the source code changes; rebuilding the WWW docs if the README changes; etc.

Reuses the configuration values `packaging.package` or `tests.package` (the former winning over the latter if both defined) when determining which source directory to scan for API doc updates.

3.4 packaging

3.4.1 packaging.release

Python package release tasks.

This module assumes:

- you're using semantic versioning for your releases
- you maintain a file called `$package/_version.py` containing normal version conventions (`__version_info__` tuple and `__version__` string).

class invocations.packaging.release.**Changelog**
An enumeration.

class invocations.packaging.release.**Release**
An enumeration.

class invocations.packaging.release.**Tag**
An enumeration.

class invocations.packaging.release.**VersionFile**
An enumeration.

invocations.packaging.release.**all_**(*c*, *dry_run=False*)
Catchall version-bump/tag/changelog/PyPI upload task.

Parameters **dry_run** (*bool*) – Handed to all subtasks which themselves have a `dry_run` flag.

Changed in version 2.1: Expanded functionality to run `publish` and `push` as well as `prepare`.

Changed in version 2.1: Added the `dry_run` flag.

invocations.packaging.release.**build**(*c*, *sdist=True*, *wheel=True*, *directory=None*, *python=None*, *clean=False*)
Build sdist and/or wheel archives, optionally in a temp base directory.

All parameters/flags honor config settings of the same name, under the `packaging` tree. E.g. say `.configure({'packaging': {'wheel': False}})` to disable building wheel archives by default.

Parameters

- **sdist** (*bool*) – Whether to build sdists/tgzs. Default: `True`.
- **wheel** (*bool*) – Whether to build wheels (requires the `wheel` package from PyPI). Default: `True`.
- **directory** (*str*) – Allows specifying a specific directory in which to perform builds and dist creation. Useful when running as a subroutine from `publish` which sets up a temporary directory.

  Up to two subdirectories may be created within this directory: one for builds (if building wheels), and one for the dist archives.

  When `None` or another false-y value (which is the default), the current working directory is used (and thus, local `dist/` and `build/` subdirectories).
- **python** (*str*) – Which Python binary to use when invoking `setup.py`.

  Defaults to `"python"`.

  If `wheel=True`, then this Python must have `wheel` installed in its default `site-packages` (or similar) location.

- **clean** – Whether to clean out the build and dist directories before building.

Changed in version 2.0: `clean` now defaults to False instead of True, cleans both dist and build dirs when True, and honors configuration.

Changed in version 2.0: `wheel` now defaults to True instead of False.

`invocations.packaging.release.`**`prepare`**(*c*, *dry_run=False*)

Edit changelog & version, git commit, and git tag, to set up for release.

Parameters **dry_run** (*bool*) – Whether to take any actual actions or just say what might occur. Will also non-fatally exit if not on some form of release branch. Default: `False`.

Returns `True` if short-circuited due to all-ok, `None` otherwise.

Changed in version 2.1: Added the `dry_run` parameter.

Changed in version 2.1: Generate annotated git tags instead of lightweight ones.

`invocations.packaging.release.`**`publish`**(*c*, *sdist=True*, *wheel=True*, *index=None*, *sign=False*, *dry_run=False*, *directory=None*, *dual_wheels=False*, *alt_python=None*, *check_desc=False*)

Publish code to PyPI or index of choice. Wraps `build` and `publish`.

This uses the `twine` command under the hood, both its pre-upload `check` subcommand (which verifies the archives to be uploaded, including checking your PyPI readme) and the `upload` one.

All parameters save `dry_run` and `directory` honor config settings of the same name, under the `packaging` tree. E.g. say `.configure({'packaging': {'wheel': True}})` to force building wheel archives by default.

Parameters

- **sdist** (*bool*) – Whether to upload sdists/tgzs. Default: `True`.
- **wheel** (*bool*) – Whether to upload wheels (requires the `wheel` package from PyPI). Default: `True`.
- **index** (*str*) – Custom upload index/repository name. See `upload` help for details.
- **sign** (*bool*) – Whether to sign the built archive(s) via GPG.
- **dry_run** (*bool*) – Skip upload step if `True`.

  This also prevents cleanup of the temporary build/dist directories, so you can examine the build artifacts.

  Note that this does not skip the `twine check` step, just the final upload.
- **directory** (*str*) – Base directory within which will live the `dist/` and `build/` directories.

  Defaults to a temporary directory which is cleaned up after the run finishes.
- **dual_wheels** (*bool*) – When `True`, builds individual wheels for Python 2 and Python 3.

  Useful for situations where you can't build universal wheels, but still want to distribute for both interpreter versions.

  Requires that you have a useful `python3` (or `python2`, if you're on Python 3 already) binary in your `$PATH`. Also requires that this other python have the `wheel` package installed in its `site-packages`; usually this will mean the global site-packages for that interpreter.

  See also the `alt_python` argument.

- **alt_python** (*str*) – Path to the 'alternate' Python interpreter to use when `dual_wheels=True`.

  When `None` (the default) will be `python3` or `python2`, depending on the currently active interpreter.

- **check_desc** (*bool*) – Whether to run `setup.py check -r -s` (uses `readme_renderer`) before trying to publish - catches long_description bugs. Default: `False`.

`invocations.packaging.release.`**`push`**(*c*, *dry_run=False*)
Push current branch and tags to default Git remote.

`invocations.packaging.release.`**`status`**(*c*)
Print current release (version, changelog, tag, etc) status.

Doubles as a subroutine, returning the return values from its inner call to `_converge` (an `(actions, state)` two-tuple of Lexicons).

`invocations.packaging.release.`**`test_install`**(*c*, *directory*)
Test installation of previously built artifacts found in `directory`.

Uses the `venv` module to build temporary virtualenvs.

`invocations.packaging.release.`**`tidelift`**(*c*, *dry_run=False*)
Add current latest version to Tidelift & set changelog link.

`invocations.packaging.release.`**`upload`**(*c*, *directory*, *index=None*, *sign=False*, *dry_run=False*)
Upload (potentially also signing) all artifacts in `directory/dist`.

Parameters

- **index** (*str*) – Custom upload index/repository name.

  By default, uses whatever the invoked `pip` is configured to use. Modify your `pypirc` file to add new named repositories.

- **sign** (*bool*) – Whether to sign the built archive(s) via GPG.

- **dry_run** (*bool*) – Skip actual publication step (and dry-run actions like signing) if `True`.

  This also prevents cleanup of the temporary build/dist directories, so you can examine the build artifacts.

3.4.2 packaging.vendorize

Tasks for importing external code into a vendor subdirectory.

`invocations.packaging.vendorize.`**`vendorize`**(*c*, *distribution*, *version*, *vendor_dir*, *package=None*, *git_url=None*, *license=None*)
Vendorize Python package `distribution` at version/SHA `version`.

Specify the vendor folder (e.g. `<mypackage>/vendor`) as `vendor_dir`.

For Crate/PyPI releases, `package` should be the name of the software entry on those sites, and `version` should be a specific version number. E.g. `vendorize('lexicon', '0.1.2')`.

For Git releases, `package` should be the name of the package folder within the checkout that needs to be vendorized and `version` should be a Git identifier (branch, tag, SHA etc.) `git_url` must also be given, something suitable for `git clone <git_url>`.

For SVN releases: xxx.

For packages where the distribution name is not the same as the package directory name, give `package='name'`.

By default, no explicit license seeking is done – we assume the license info is in file headers or otherwise within the Python package vendorized. This is not always true; specify `license=/path/to/license/file` to trigger copying of a license into the vendored folder from the checkout/download (relative to its root.)

3.5 pytest

Pytest-using variant of testing.py. Will eventually replace the latter.

`invocations.pytest.`**`coverage`**(*c*, *report='term'*, *opts=''*, *tester=None*, *codecov=False*)

Run pytest with coverage enabled.

Assumes the `pytest-cov` pytest plugin is installed.

Parameters

- **report** (*str*) – Coverage report style to use. If 'html', will also open in browser.
- **opts** (*str*) – Extra runtime opts to pass to pytest.
- **tester** – Specific test task object to invoke. If `None` (default), uses this module's local *test*.
- **codecov** (*bool*) – Whether to build XML and upload to Codecov. Requires `codecov` tool. Default: `False`.

`invocations.pytest.`**`integration`**(*c*, *opts=None*, *pty=True*, *x=False*, *k=None*, *verbose=True*, *color=True*, *capture='sys'*, *module=None*)

Run the integration test suite. May be slow!

See `pytest.test` for description of most arguments.

`invocations.pytest.`**`test`**(*c*, *verbose=True*, *color=True*, *capture='sys'*, *module=None*, *k=None*, *x=False*, *opts=''*, *pty=True*, *warnings=True*)

Run pytest with given options.

Parameters

- **verbose** (*bool*) – Whether to run tests in verbose mode.
- **color** (*bool*) – Whether to request colorized output (typically only works when `verbose=True`.)
- **capture** (*str*) – What type of stdout/err capturing pytest should use. Defaults to `sys` since pytest's own default, `fd`, tends to trip up subprocesses trying to detect PTY status. Can be set to `no` for no capturing / useful print-debugging / etc.
- **module** (*str*) – Select a specific test module to focus on, e.g. `main` to only run `tests/main.py`. (Note that this is a specific idiom aside from the use of `-o '-k pattern'`.) Default: `None`.
- **k** (*str*) – Convenience passthrough for `pytest -k`, i.e. test selection. Default: `None`.
- **x** (*bool*) – Convenience passthrough for `pytest -x`, i.e. fail-fast. Default: `False`.
- **opts** (*str*) – Extra runtime options to hand to `pytest`.
- **pty** (*bool*) – Whether to use a pty when executing pytest. Default: `True`.

- **warnings** (*bool*) – Inverse alias for the pytest `--disable_warnings` flag; when this is False (i.e. called on CLI as `--no-warnings`), `--disable-warnings` will be given. Default: `True`.

  New in version 2.0.

Python Module Index

i

Index