pytest-skippy

Release unknown

Jan 27, 2018

Contents

Automatically skip tests that don't need to run!

This library works with pytest to generate a complete import graph for your tests. If nothing in the import graph has changed (according to GIT), the test is skipped.

CHAPTER 1

Quickstart

1.1 Requirements

This package interacts with GIT and assumes that your code and tests both exist within the same git repository.

- GIT
- pytest>=2.3.4

1.2 Installation

```
pip install pytest-skippy
```

1.3 Usage

This command uses the default target branch of `origin/master`. The target branch is the branch that has been modified. Most users should not have to change the target branch when getting started.

```
py.test --skippy
```

CHAPTER 2

User Guide

2.1 Overview

pytest-skippy automatically skips tests that don't need to run!

The process for determining if a test needs to run involves

1. Generating a set of changed files by querying GIT
2. Generating an import graph for each test file using the python `ast` module and `pkgutil` importers/loaders.
3. If any files in the import graph have been modified, the test needs to run!

2.2 PyTest Command Line Options

2.2.1 `--skippy`

(*Default*: `False`)

Enables this plugin, automatically skipping tests when applicable.

2.2.2 `--skippy-target-branch`

(*Default*: `origin/master`)

The target branch is the branch that has been modified. This branch parameter is passed to `git merge-base` to determine which files have been changed.

For example, for a pull request using github and Travis-CI, the target branch is stored in an environment variable called `TRAVIS_PULL_REQUEST_BRANCH`.

2.2.3 `--skippy-safe`

(*Default*: `False`)

Safe mode changes the behavior of the skippy plugin so that any import that cannot be resolved forces a test run.

This makes sure that if a module is missing for any reason the test that would catch the missing import is run.

Using this mode is likely to result in many false positives, causing tests to run when it may not be necessary.

CHAPTER 3

Limitations

Since this library does not actually import any code (due to potential side effects of importing and performance), there are a number of limitations surrounding the accuracy of this plugin.

3.1 Runtime changes to `sys.path`

This library does not account for any changes to the python path at runtime. Modifications to the python path at runtime may change which file is loaded by an import statement.

If your code modifies `sys.path` at runtime, the use of *–skippy-safe* is recommended.

3.2 Importing modules with `from` statements

The type of literals that are imported with `from` style imports is ambiguous.

Consider the following statement:

```
from foo import bar
```

In the statement above, is `bar` an attribute of `foo` or a module?

It's possible that `foo.bar` is a module and the type of `bar` will be a module. In these cases, `import foo.bar` will succeed at runtime.

However, if `bar` is a function, `import foo.bar` will fail at runtime. This plugin does not attempt to determine the type of `bar`.

By default, pytest-skippy assumes that if `foo.bar` cannot be located, `bar` must be an attribute of the module `foo`. As a result, if `bar` is in fact a missing module, the test will be skipped by default.

Therefore, if module imports with the `from` statement are expected, *–skippy-safe* should be used.

CHAPTER 4

pytest_skippy

4.1 pytest_skippy.core

4.2 pytest_skippy.git

pytest_skippy.git.**detect_changed_files**(*target_branch*, *base_branch='HEAD'*, *git_repo_dir=None*)

Get a list of changed files in a git repo

Parameters

- **target_branch** (*str*) – The merge target for a branch.
- **base_branch** (*str*) – The branch that's being merged (default: 'HEAD')
- **git_repo_dir** (*None or str*) – The base directory of the git repository. None = current directory. (Default)

Returns A set of files that have changed in the git repository.

Return type set

4.3 pytest_skippy.imp

pytest_skippy.imp.**convert_module_to_filename**(*module_name*)

Find a module's file location

Returns the canonical path to the file that defines a module. The canonical path is retrieved using a call to `os.path.realpath()`

Parameters **module_name** (*str*) – Full path to a module.

Returns A string containing the filename of the requested module.

Return type str

Example:

```
>>> import os.path
>>> path = convert_module_to_filename('re')
>>> os.path.split(path)[-1]
're.py'
```

4.4 pytest_skippy.parse

pytest_skippy.parse.**get_imported_modules**(*filename*)

Return modules that are imported by a file

This function will extract all modules that are imported within a python file. The return value includes imports that happen at scopes other than the top level scope.

This parser does not execute any of the python code in the file.

The parser will also return a set of confirmed modules (strings that are guaranteed to be interpreted as module types at runtime)

```
from foo import bar  # foo is a confirmed module, bar is a candidate
import bar  # bar is a confirmed module
```

For example:

```
>>> import tempfile
>>> with tempfile.NamedTemporaryFile(delete=False) as f:
...     x = f.write(b'''
... import os
...
... def inner():
...    import re
... ''')
>>> modules, _ = get_imported_modules(f.name)
>>> print(sorted([m for m in modules]))
['os', 're']
>>> import os ; os.unlink(f.name)
```

Parameters **filename** (*str*) – File path to a python file

Returns (modules, confirmed_modules)

Return type tuple

4.5 pytest_skippy.util

pytest_skippy.util.**flatten_imports**(*imported_module*, *import_tree*)

Returns a set of all modules imported by imported_module

The import tree is a dict in the form {module: set(imported_by)}

Parameters

- **imported_module** (*str*) – A leaf module name
- **import_tree** (*str*) – A dict in the form of {module: set(imported_by)}

Returns A set of modules that import imported_module

Return type set

Example: A is imported by B

```
>>> flat = flatten_imports('A', {'A': {'B'}, 'B': set()})
>>> sorted(list(flat))
['A', 'B']
```

Python Module Index

p

Index