Changing standard (Python) test discovery

Ignore paths during test collection

You can easily ignore certain test directories and modules during collection by passing the `--ignore=path` option on the cli. `pytest` allows multiple `--ignore` options. Example:

```
tests/
|-- example
|   |-- test_example_01.py
|   |-- test_example_02.py
|   '-- test_example_03.py
|-- foobar
|   |-- test_foobar_01.py
|   |-- test_foobar_02.py
|   '-- test_foobar_03.py
'-- hello
    '-- world
        |-- test_world_01.py
        |-- test_world_02.py
        '-- test_world_03.py
```

Now if you invoke `pytest` with `--ignore=tests/foobar/test_foobar_03.py --ignore=tests/hello/`, you will see that `pytest` only collects test-modules, which do not match the patterns specified:

```
========= test session starts ==========
platform darwin -- Python 2.7.10, pytest-2.8.2, py-1.4.30, pluggy-0.3.1
rootdir: $REGENDOC_TMPDIR, inifile:
collected 5 items

tests/example/test_example_01.py .
tests/example/test_example_02.py .
tests/example/test_example_03.py .
tests/foobar/test_foobar_01.py .
tests/foobar/test_foobar_02.py .

======= 5 passed in 0.02 seconds =======
```

Keeping duplicate paths specified from command line

Default behavior of `pytest` is to ignore duplicate paths specified from the command line. Example:

```
py.test path_a path_a

...
collected 1 item
...
```

Just collect tests once.

To collect duplicate tests, use the `--keep-duplicates` option on the cli. Example:

```
py.test --keep-duplicates path_a path_a

...
collected 2 items
...
```

As the collector just works on directories, if you specify twice a single test file, `pytest` will still collect it twice, no matter if the `--keep-duplicates` is not specified. Example:

```
py.test test_a.py test_a.py

...
collected 2 items
...
```

Changing directory recursion

You can set the **norecursedirs** option in an ini-file, for example your `pytest.ini` in the project root directory:

```
# content of pytest.ini
[pytest]
norecursedirs = .svn _build tmp*
```

This would tell `pytest` to not recurse into typical subversion or sphinx-build directories or into any `tmp` prefixed directory.

Changing naming conventions

You can configure different naming conventions by setting the **python_files**, **python_classes** and **python_functions** configuration options. Example:

```
# content of pytest.ini
# can also be defined in tox.ini or setup.cfg file, although the section
# name in setup.cfg files should be "tool:pytest"
[pytest]
python_files=check_*.py
python_classes=Check
python_functions=*_check
```

This would make `pytest` look for tests in files that match the `check_* .py` glob-pattern, `Check` prefixes in classes, and functions and methods that match `*_check`. For example, if we have:

```
# content of check_myapp.py
class CheckMyApp(object):
    def simple_check(self):
        pass
    def complex_check(self):
        pass
```

then the test collection looks like this:

```
$ pytest --collect-only
======= test session starts ========
platform linux -- Python 3.x.y, pytest-3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile: pytest.ini
collected 2 items
<Module 'check_myapp.py'>
  <Class 'CheckMyApp'>
    <Instance '()'>
      <Function 'simple_check'>
      <Function 'complex_check'>

======= no tests ran in 0.12 seconds ========
```

> **Note:**
>
> the `python_functions` and `python_classes` options has no effect for `unittest.TestCase` test discovery because pytest delegates detection of test case methods to unittest code.

Interpreting cmdline arguments as Python packages

You can use the `--pyargs` option to make `pytest` try interpreting arguments as python package names, deriving their file system path and then running the test. For example if you have unittest2 installed you can type:

```
pytest --pyargs unittest2.test.test_skipping -q
```

which would run the respective test module. Like with other options, through an ini-file and the **`addopts`** option you can make this change more permanently:

```
# content of pytest.ini
[pytest]
addopts = --pyargs
```

Now a simple invocation of `pytest NAME` will check if NAME exists as an importable package/module and otherwise treat it as a filesystem path.

Finding out what is collected

You can always peek at the collection tree without running tests like this:

```
. $ pytest --collect-only pythoncollection.py
======= test session starts ========
platform linux -- Python 3.x.y, pytest-3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile: pytest.ini
collected 3 items
<Module 'CWD/pythoncollection.py'>
  <Function 'test_function'>
  <Class 'TestClass'>
    <Instance '()'>
      <Function 'test_method'>
      <Function 'test_anothermethod'>

======= no tests ran in 0.12 seconds ========
```

customizing test collection to find all .py files

You can easily instruct `pytest` to discover tests from every python file:

```
# content of pytest.ini
[pytest]
python_files = *.py
```

However, many projects will have a `setup.py` which they don't want to be imported. Moreover, there may files only importable by a specific python version. For such cases you can dynamically define files to be ignored by listing them in a `conftest.py` file:

```
# content of conftest.py
import sys

collect_ignore = ["setup.py"]
if sys.version_info[0] > 2:
    collect_ignore.append("pkg/module_py2.py")
```

And then if you have a module file like this:

```
# content of pkg/module_py2.py
def test_only_on_python2():
    try:
```

```
        assert 0
    except Exception, e:
        pass
```

and a setup.py dummy file like this:

```
# content of setup.py
0/0  # will raise exception if imported
```

then a pytest run on Python2 will find the one test and will leave out the setup.py file:

```
#$ pytest --collect-only
====== test session starts ======
platform linux2 -- Python 2.7.10, pytest-2.9.1, py-1.4.31, pluggy-0.3.1
rootdir: $REGENDOC_TMPDIR, inifile: pytest.ini
collected 1 items
<Module 'pkg/module_py2.py'>
  <Function 'test_only_on_python2'>

====== no tests ran in 0.04 seconds ======
```

If you run with a Python3 interpreter both the one test and the setup.py file will be left out:

```
$ pytest --collect-only
======= test session starts ========
platform linux -- Python 3.x.y, pytest-3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile: pytest.ini
collected 0 items

======= no tests ran in 0.12 seconds ========
```