Measuring Test Coverage

Jamie Counsell
SOFTWARE DEVELOPER

@jamiecounsell   www.jamiecounsell.me

Test Coverage

Test coverage

- A valuable metric
- Not our primary goal

Coverage in Django

- Use standard Python test coverage
- coverage.py

Measuring Test Coverage

Metrics
90% - 95% is generally considered good coverage

Configuration
Good coverage is only possible with correct configuration

Persistence
Keeping test coverage up needs to be done from the start

`coverage.py`

Test coverage for Python

- Most popular tool for this purpose
- Generates text or HTML outputs
- Lots of third party support
- File-based configuration

```
$ coverage run manage.py test
```

Generating coverage

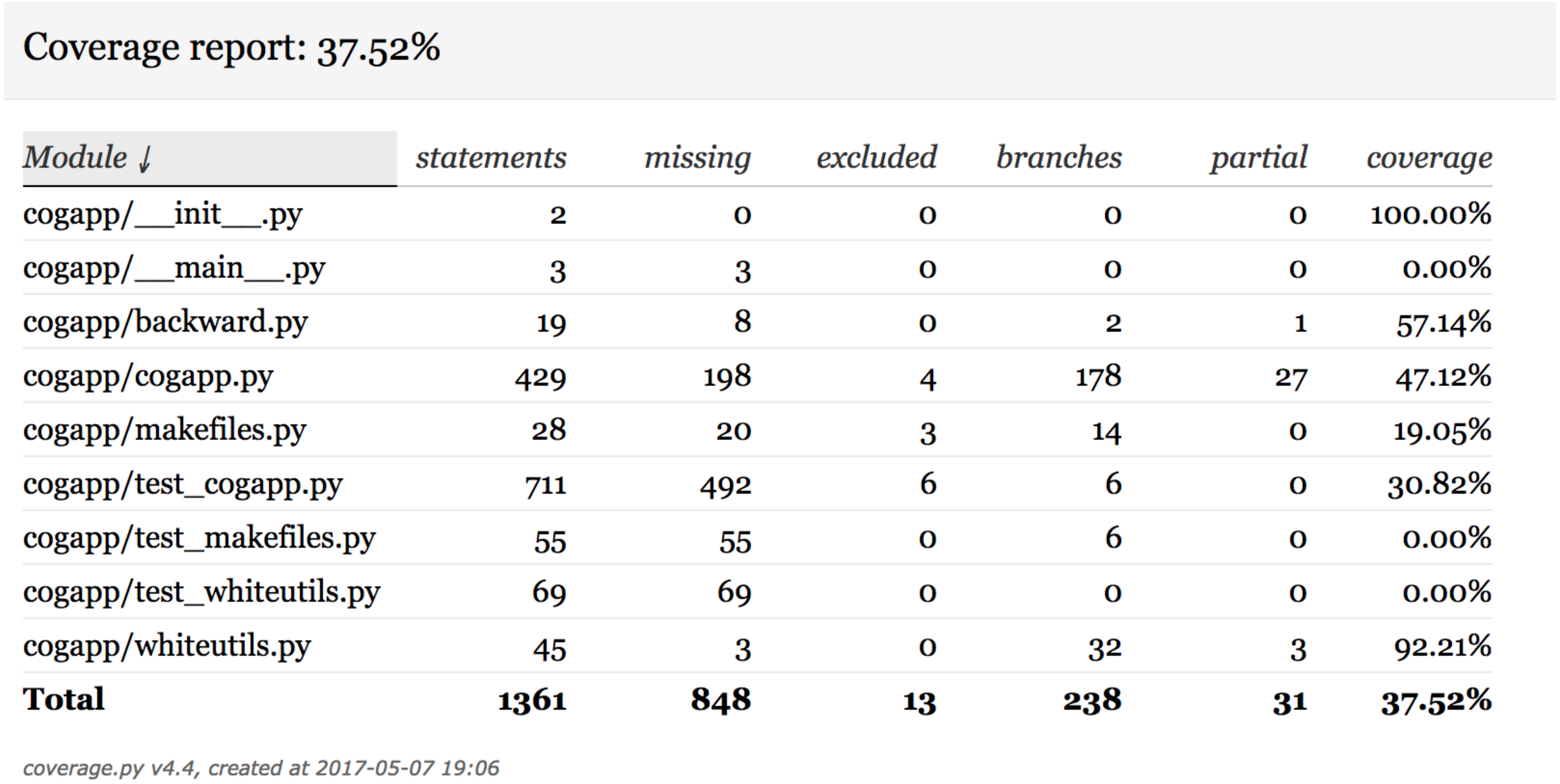
- Simple CLI provided to measure test coverage
- `coverage run` command run before command to measure

```
Name                  Stmts    Miss   Cover   Missing
-----------------------------------------------------
my_program.py            20       4     80%   33-35, 39
my_module.py             15       2     86%   8, 12
my_other_module.py       56       6     89%   17-23
-----------------------------------------------------
TOTAL                    91      12     87%
```

Coverage.py Output

Statements	Number of testable statements
Miss	Number not tested
Cover	% covered
Missing	Lines not reached by tests

Rich Output

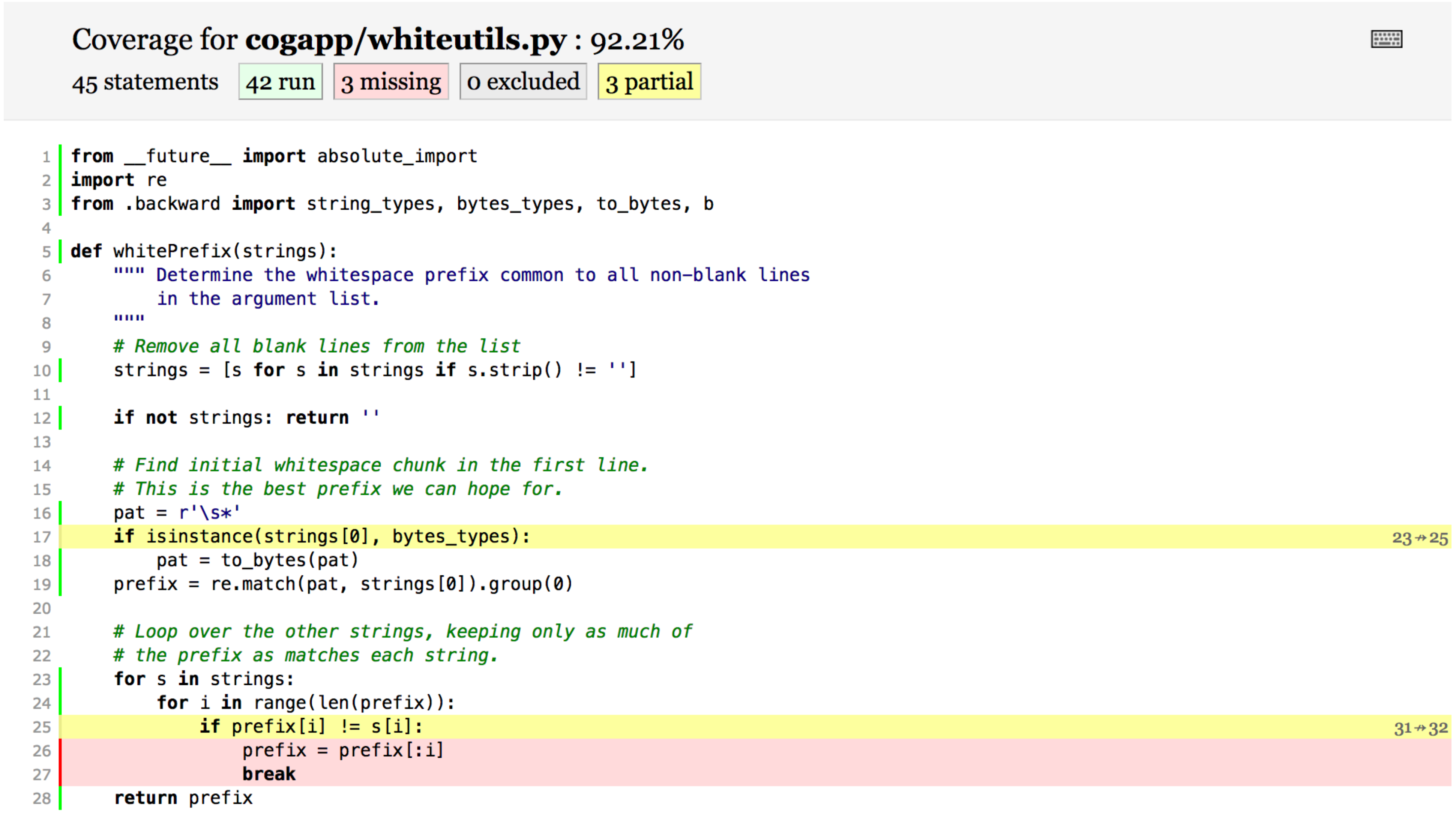


Coverage report: 37.52%

Module ↓	*statements*	*missing*	*excluded*	*branches*	*partial*	*coverage*
cogapp/__init__.py	2	0	0	0	0	100.00%
cogapp/__main__.py	3	3	0	0	0	0.00%
cogapp/backward.py	19	8	0	2	1	57.14%
cogapp/cogapp.py	429	198	4	178	27	47.12%
cogapp/makefiles.py	28	20	3	14	0	19.05%
cogapp/test_cogapp.py	711	492	6	6	0	30.82%
cogapp/test_makefiles.py	55	55	0	6	0	0.00%
cogapp/test_whiteutils.py	69	69	0	0	0	0.00%
cogapp/whiteutils.py	45	3	0	32	3	92.21%
Total	**1361**	**848**	**13**	**238**	**31**	**37.52%**

coverage.py v4.4, created at 2017-05-07 19:06

Rich Output



Coverage for **cogapp/whiteutils.py** : 92.21%

45 statements | 42 run | 3 missing | 0 excluded | 3 partial

```
from __future__ import absolute_import
import re
from .backward import string_types, bytes_types, to_bytes, b

def whitePrefix(strings):
    """ Determine the whitespace prefix common to all non-blank lines
        in the argument list.
    """
    # Remove all blank lines from the list
    strings = [s for s in strings if s.strip() != '']

    if not strings: return ''

    # Find initial whitespace chunk in the first line.
    # This is the best prefix we can hope for.
    pat = r'\s*'
    if isinstance(strings[0], bytes_types):                                    23↛25
        pat = to_bytes(pat)
    prefix = re.match(pat, strings[0]).group(0)

    # Loop over the other strings, keeping only as much of
    # the prefix as matches each string.
    for s in strings:
        for i in range(len(prefix)):
            if prefix[i] != s[i]:                                              31↛32
                prefix = prefix[:i]
                break
    return prefix
```

Demo

Run coverage report

- Generate report
- Generate HTML
- Exclude certain files

Summary

Measuring test coverage

- Analyzing output
- Configuring for accuracy
- Setting us up to succeed early