

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

make test

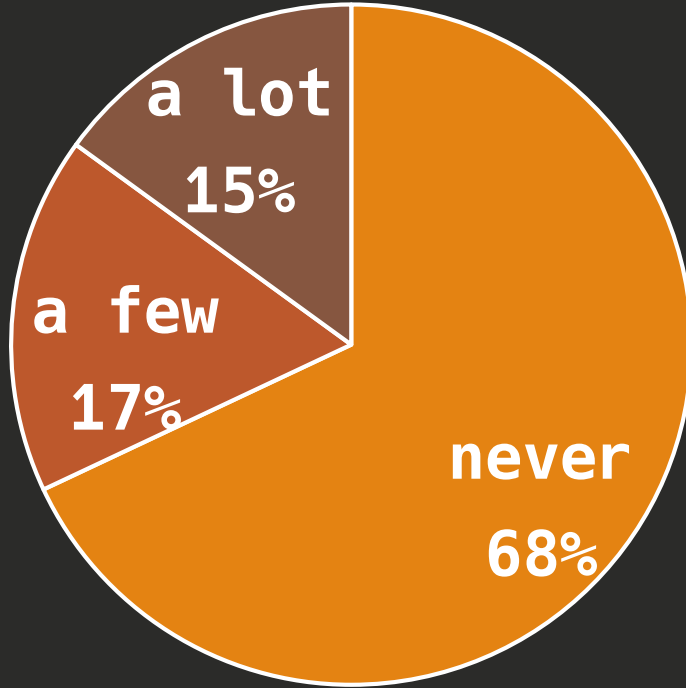
m-nishi

VimConf 2019, Nov 3rd 2019

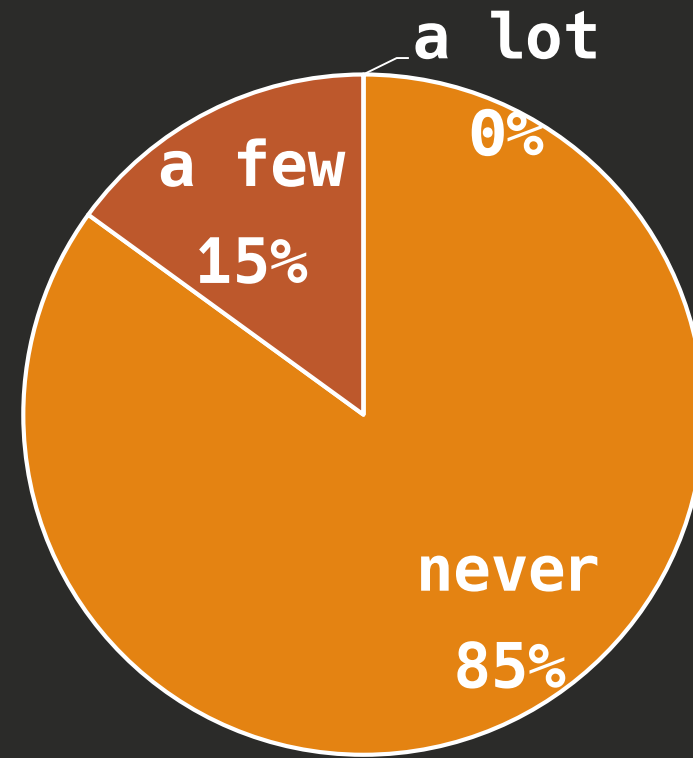
~
~
~
~
~
~
~
~
~
~

Have you ever...

built Vim?



run Vim's test?



Poll
on Twitter
Thank you!

2 2 2 2 2 2

About me



Name: `m-nishi`

Vim experience: 2 years

Embedded Software Engineer

Twitter: [mnishz0](https://twitter.com/mnishz0)

GitHub: [mnishz](https://github.com/mnishz)

I will talk about...

- How to run Vim's test
- Overview
- Contents of test script
- Coverage
 - How I wrote a Vim's test

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

How to run Vim's test

~
~
~
~
~
~
~
~
~
~

How to run Vim's test

```
$ cd src && make test  
libtool-bin is required
```

-> Demo

How to run Vim's test

Result

```
Test_zz2_terminal_guioptions_b  
test_windows_home.vim: only wo  
Test_timer_peek_and_get_char()
```

```
-----  
Executed: 2201 Tests  
Skipped: 35 Tests  
Failed: 0 Tests
```

ALL DONE

```
-----  
Executed: 2198 Tests  
Skipped: 35 Tests  
FAILED: 1 Tests
```

Failures:-

```
From test_assert.vim:  
Found errors in Test_assert_fa  
function RunTheTest[40]..Test_  
function RunTheTest[40]..Test_
```

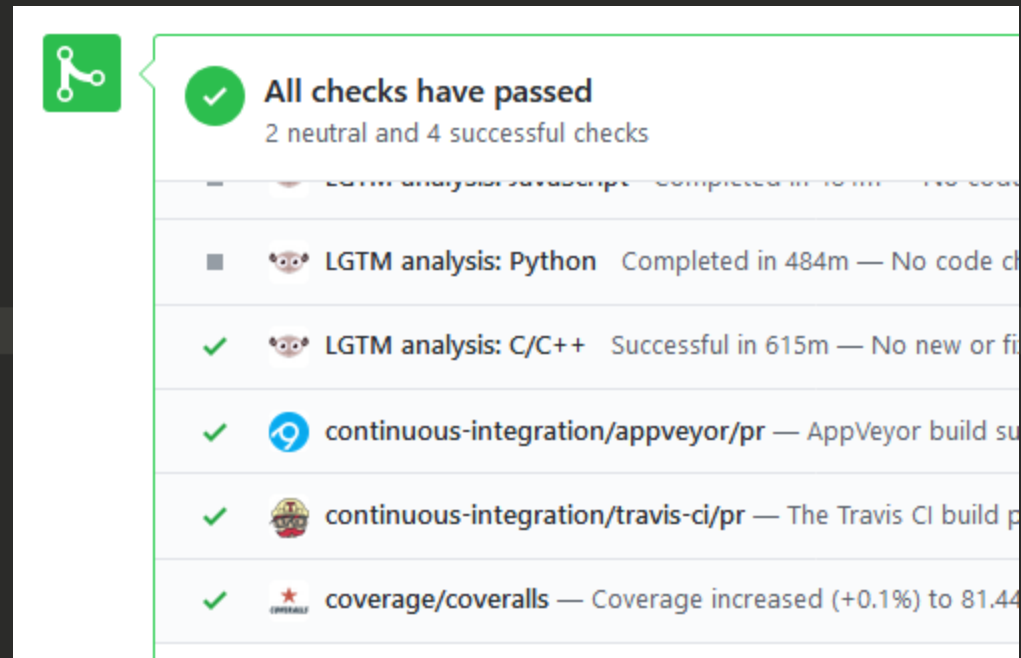
TEST FAILURE

Overview

- **Written in Vim script**
- Also executed by CI services
- `:help testing`` for details

Overview

- Written in Vim script
- Also executed by CI services
- `:help testing`` for details



Overview

- Written in Vim script
- Also executed by CI services
- `:help testing`` for details

Benefit of writing test

- Deep understanding of Vim
- Pull request may be merged quickly
- Contribution chance for the test!

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

Contents of test script

~
~
~
~
~
~
~
~
~
~

Test sequence

```
src/testdir/test_****.vim
```

SetUp()



Test_A()



TearDown()

SetUp()



Test_B()



TearDown()

```
1
2
3
4 func SetUp()
5     new dummy
6     set nrformats&vim
7
8 endfunc
```

src/testdir/test_
increment.vim

```
9
10
11 func TearDown()
12     bwipe!
13
14 endfunc
```


```
15
16
17 func Test_visual_increment_01()
18     call setline(1, repeat(["foobaar-10"], 5))
19
20     call cursor(1, 1)
21     exec "norm! \<C-A>"
22     call assert_equal("foobaar-9", getline('.'))
23     call assert_equal([0, 1, 9, 0], getpos('.'))
```

```
1
2
3
4 func SetUp()
5     new dummy
6     set nrformats&vim
7
8 endfunc
```

src/testdir/test_
increment.vim

```
9
10
11 func TearDown()
12     bwipe!
13
14 endfunc
```

```
15
16
17 func Test_visual_increment_01()
18     call setline(1, repeat(["foobaar-10"], 5))
19
20     call cursor(1, 1)
21     exec "norm! \<C-A>"
22     call assert_equal("foobaar-9", getline('.'))
23     call assert_equal([0, 1, 9, 0], getpos('.'))
```




```
1
2
3
4 func SetUp()
5     new dummy
6     set nrformats&vim
7
8 endfunc
```

src/testdir/test_
increment.vim

```
9
10
11 func TearDown()
12     bwipe!
13
14 endfunc
```

```
15
16
17 func Test_visual_increment_01()
18     call setline(1, repeat(["foobaar-10"], 5))
19
20     call cursor(1, 1)
21     exec "norm! \<C-A>"
22     call assert_equal("foobaar-9", getline('.'))
23     call assert_equal([0, 1, 9, 0], getpos('.'))
```





```
1
2
3
4 func SetUp()
5     new dummy
6     set nrformats&vim
7
8 endfunc
```

src/testdir/test_
increment.vim

```
9
10
11 func TearDown()
12     bwipe!
13
14 endfunc
```

```
15
16
17 func Test_visual_increment_01()
18     call setline(1, repeat(["foobaar-10"], 5))
19
20     call cursor(1, 1)
21     exec "norm! \<C-A>"
22     call assert_equal("foobaar-9", getline('.'))
23     call assert_equal([0, 1, 9, 0], getpos('.'))
```



```
1
2
3
4 func SetUp()
5     new dummy
6     set nrformats&vim
7
8 endfunc
```

src/testdir/test_
increment.vim

```
9
10
11 func TearDown()
12     bwipe!
13
14 endfunc
```

```
15
16
17 func Test_visual_increment_01()
18     call setline(1, repeat(["foobaar-10"], 5))
19
20     call cursor(1, 1)
21     exec "norm! \<C-A>"
22     call assert_equal("foobaar-9", getline('.'))
23     call assert_equal([0, 1, 9, 0], getpos('.'))
```



```
1
2
3
4 func SetUp()
5     new dummy
6     set nrformats&vim
7
8 endfunc
```

src/testdir/test_
increment.vim

```
9
10
11 func TearDown()
12     bwipe!
13
14 endfunc
```

```
15
16
17 func Test_visual_increment_01()
18     call setline(1, repeat(["foobaar-10"], 5))
19
20     call cursor(1, 1)
21     exec "norm! \<C-A>"
22     call assert_equal("foobaar-9", getline('.'))
23     call assert_equal([0, 1, 9, 0], getpos('.'))
```



terminal-diff

It compares

- window size
- text
- color
- other attributes
 - (bold, underline...)

```
1-----
2-----
3      hello there      r one
4                    another two
5                    another three
6-----
                                           1,1
===== src/testdir/dumps/Test_popupwin_01.dump =====
-----
      bbbbbbbbbbbbbbbbbbb
-----
===== src/testdir/failed/Test_popupwin_01.dump =====
1-----
2-----
3      hello there      r one
4                    another two
5                    another three
6-----
                                           1,1
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

Coverage

~
~
~
~
~
~
~
~

Coverage

- <https://codecov.io/gh/vim/vim>
- Reached 80 % on this May
 - Bram sent a thank-you message
- Pull request for test looks welcome.

How I wrote a Vim's test

1. Found not tested code

```
2573     static void
2574     f_cindent(typval_T *argvars UNUSED, typval_T *rettv)
2575     {
2576     #ifdef FEAT_CINDENT
2577         pos_T      pos;
2578         linenr_T   lnum;
2579
2580         pos = curwin->w_cursor;
2581         lnum = tv_get_lnum(argvars);
2582         if (lnum >= 1 && lnum <= curbuf->b_ml.ml_line_count)
2583         {
2584             curwin->w_cursor.lnum = lnum;
2585             rettv->vval.v_number = get_c_indent();
2586             curwin->w_cursor = pos;
```


How I wrote a Vim's test

2. Studied how the code is executed

`cindent({lnum})`

Get the amount of indent for line `{lnum}` according the C indenting rules, as with ['cindent'](#).
The indent is counted in spaces, the value of ['tabstop'](#) is relevant. `{lnum}` is used just like in [getline\(\)](#).
When `{lnum}` is invalid or Vim was not compiled the [+cindent](#) feature, -1 is returned.
See [C-indenting](#).

Can also be used as a [method](#):

`GetLnum()->cindent()`

How I wrote a Vim's test

3. Wrote a test script

```
104
105 + func Test_cindent_func()
106 +   new
107 +   setlocal cindent
108 +   call setline(1, ['int main(void)', '{', 'return 0;', '}'])
109 +   call assert_equal(cindent(0), -1)
110 +   call assert_equal(cindent(3), &sw)
111 +   call assert_equal(cindent(line('$')+1), -1)
112 +   bwipe!
113 + endfunc
114 +
115     " vim: shiftwidth=2 sts=2 expandtab
```

How I wrote a Vim's test

4. Checked coverage

```
2573     static void
2574 ① f_cindent(typval_T *argvars UNUSED, typval_T *rettv)
2575     {
2576     #ifdef FEAT_CINDENT
2577         pos_T      pos;
2578         linenr_T   lnum;
2579
2580 ②     pos = curwin->w_cursor;
2581 ②     lnum = tv_get_lnum(argvars);
2582 ②     if (lnum >= 1 && lnum <= curbuf->b_ml.ml_line_count)
2583     {
2584 ②         curwin->w_cursor.lnum = lnum;
2585 ②         rettv->vval.v_number = get_c_indent();
2586 ②         curwin->w_cursor = pos;
```

How I wrote a Vim's test

5. Sent a pull request!

The screenshot shows a GitHub pull request interface. At the top, a red 'Closed' badge is visible next to the title 'improve test coverage of evalfunc.c #4374'. Below the title, it states 'mnishz wants to merge 4 commits into vim:master from mnishz:improve_evalfunc_coverage'. A vertical timeline of activity is shown below:

- A comment from user 'mnishz' (profile picture of a fruit) dated '16 May'. The comment text reads: 'Thank you for your comment. I think the current patch is not too small and not too large to read, so I'll remove the draft annotation as is.' The comment box includes an 'Author' label, a smiley face icon, and a three-dot menu.
- A commit activity section for 'mnishz' dated '16 May' with two entries:
 - 'add an expand() test case for list' with commit hash 'e95677f' and a red 'X' icon.
 - 'merge Test_expand() into test_expand_func.vim' with commit hash '836c190' and a red 'X' icon.
- A final activity entry from user 'brammool' dated '17 May', stating 'closed this in 17aca70', with a red 'X' icon.

Short summary

- `$ make test`
- Written by Vim script
- Let's write test for untested code!
- Thanks for flying with me!

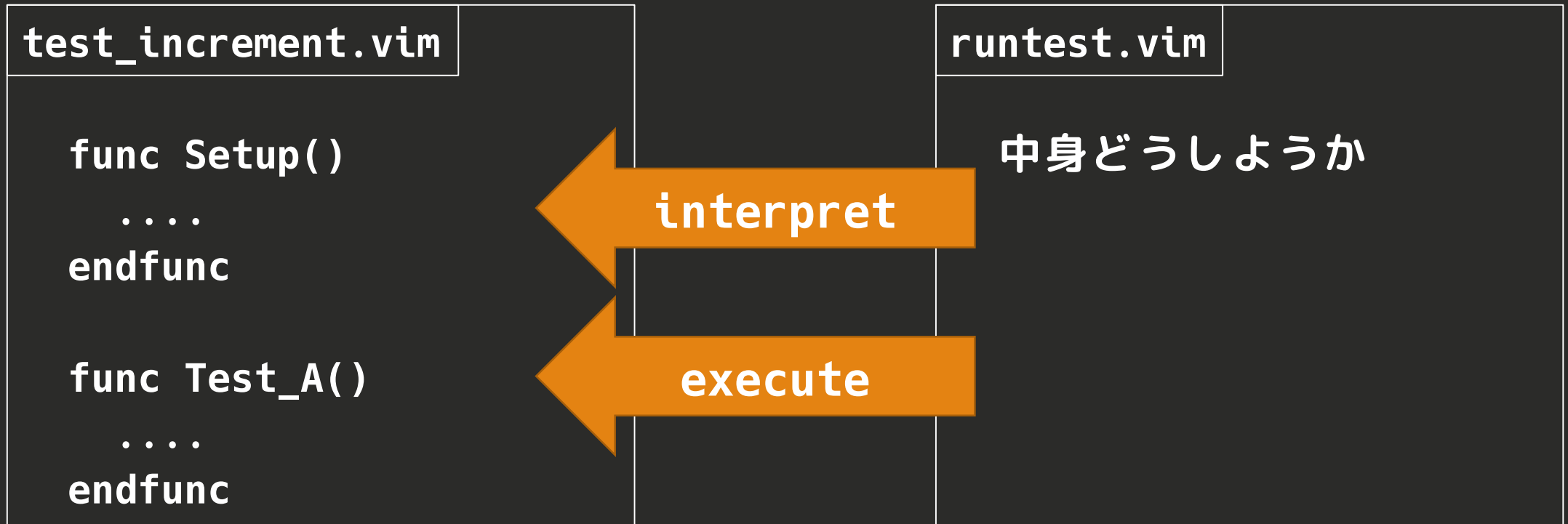
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

Backup

~
~
~
~
~
~
~
~

How the test scripts are executed

• `$ vim -S runtest.vim test_increment.vim`

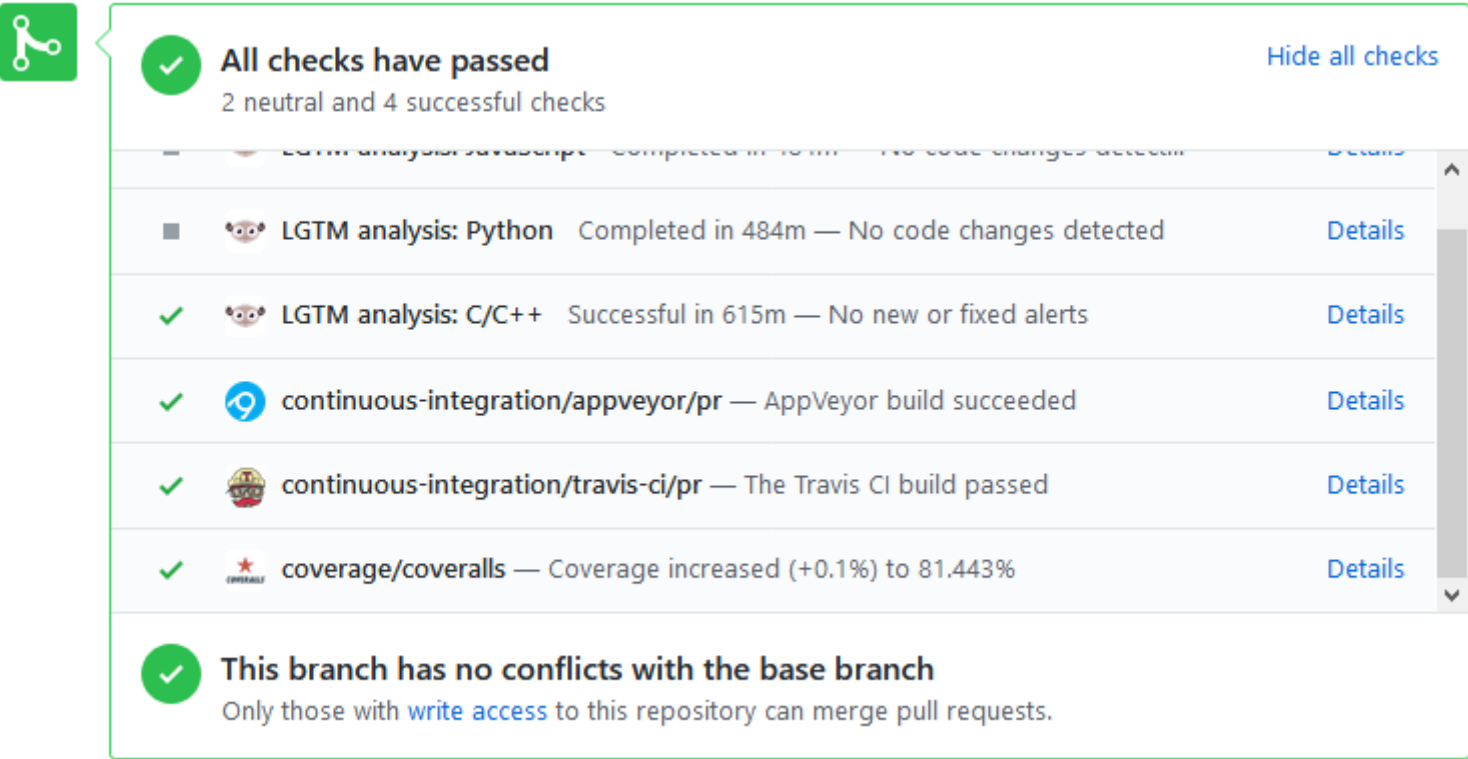


REDIR_TEST_TO_NULL


```
diff --git a/src/testdir/Makefile b/src/testdir/Makefile
index bcf2f8c37..0ffdbc228 100644
--- a/src/testdir/Makefile
+++ b/src/testdir/Makefile
@@ -12,7 +12,7 @@ SCRIPTSOURCE = ../../runtime
 # Comment out this line to see the verbose output of tests.
 #
 # Catches SwapExists to avoid hanging at the ATTENTION prompt.
-REDIR_TEST_TO_NULL = --cmd 'au SwapExists * let v:swapchoice = "e" > /dev/null
+REDIR_TEST_TO_NULL = --cmd 'au SwapExists * let v:swapchoice = "e"







# Uncomment this line to use valgrind for memory leaks and extra warnings.
#   The output goes into a file "valgrind.testN"
```



Test by CI services



The screenshot shows a GitHub Actions CI status for a pull request. At the top, a green checkmark icon is followed by the text "All checks have passed" and "2 neutral and 4 successful checks". A "Hide all checks" link is visible in the top right. Below this, a list of checks is shown, each with a status icon, a check name, a description, and a "Details" link. The checks are: "LGTM analysis: JavaScript" (neutral), "LGTM analysis: Python" (neutral), "LGTM analysis: C/C++" (successful), "continuous-integration/appveyor/pr" (successful), "continuous-integration/travis-ci/pr" (successful), and "coverage/coveralls" (successful). At the bottom, a green checkmark icon is followed by the text "This branch has no conflicts with the base branch" and "Only those with write access to this repository can merge pull requests."


 **All checks have passed** [Hide all checks](#)
2 neutral and 4 successful checks

-  LGTM analysis: JavaScript — Completed in 10m — No code changes detected [Details](#)
-  LGTM analysis: Python — Completed in 484m — No code changes detected [Details](#)
-  LGTM analysis: C/C++ — Successful in 615m — No new or fixed alerts [Details](#)
-  continuous-integration/appveyor/pr — AppVeyor build succeeded [Details](#)
-  continuous-integration/travis-ci/pr — The Travis CI build passed [Details](#)
-  coverage/coveralls — Coverage increased (+0.1%) to 81.443% [Details](#)

 **This branch has no conflicts with the base branch**
Only those with [write access](#) to this repository can merge pull requests.

Coverage, Codecov

README.md



The editor

build passing build failing codecov 82% coverity passed code quality: c/c++ A+ Debian Testing 2:8.1.0875-5+b3

in repositories 52

What is Vim?

Coverage, Codecov

```
1514 void
1515 2 set_context_for_expression(
1516     expand_T *xp,
1517     char_u *arg,
1518     cmdidx_T cmdidx)
1519 {
1520 3 int got_eq = FALSE;
1521 int c;
1522 char_u *p;
1523
1524 3 if (cmdidx == CMD_let)
1525 {
1526 3 xp->xp_context = EXPAND_USER_VARS;
1527 3 if (vim_strpbrk(arg, (char_u *)"\\"'+-*/%.?!?~|&${[<>,#"") == NULL)
1528 {
1529     /* ":let var1 var2 ...": find last space. */
1530     for (p = arg + STRLEN(arg); p >= arg; )
1531     {
1532         xp->xp_pattern = p;
1533         MB_PTR_BACK(arg, p);
1534         if (VIM_ISWHITE(*p))
1535             break;
1536     }
1537 2 return;
1538     }
1539 }
1540 else
```

How to get test coverage

```
674 # TEST COVERAGE - Uncomment the two lines below the explanation to get code
675 # coverage information. (provided by Yegappan Lakshmanan)
676 # 1. make clean, run configure and build Vim as usual.
677 # 2. Generate the baseline code coverage information:
678 #>-----$ lcov -c -i -b . -d objects -o objects/coverage_base.info
679 # 3. Run "make test" to run the unit tests. The code coverage information will
680 # be generated in the src/objects directory.
681 # 4. Generate the code coverage information from the tests:
682 #>-----$ lcov -c -b . -d objects/ -o objects/coverage_test.info
683 # 5. Combine the baseline and test code coverage data:
684 #>-----$ lcov -a objects/coverage_base.info -a objects/coverage_test.info -o obj
685 # 6. Process the test coverage data and generate a report in html:
686 #>-----$ genhtml objects/coverage_total.info -o objects
687 # 7. Open the objects/index.html file in a web browser to view the coverage
688 # information.
689 #
690 # PROFILE_CFLAGS=-g -O0 -fprofile-arcs -ftest-coverage
691 # LDFLAGS=--coverage
```

Makefiles

- `src/Makefile`

- `unittests`, `test_libvterm`, `scripttests`

- `src/testdir/Makefile`

- `.vim.res`

unittest

- # Unittest files
- JSON_TEST_SRC = json_test.c
- JSON_TEST_TARGET = json_test\$(EXEEXT)
- KWORD_TEST_SRC = kword_test.c
- KWORD_TEST_TARGET = kword_test\$(EXEEXT)
- MEMFILE_TEST_SRC = memfile_test.c
- MEMFILE_TEST_TARGET = memfile_test\$(EXEEXT)
- MESSAGE_TEST_SRC = message_test.c
- MESSAGE_TEST_TARGET = message_test\$(EXEEXT)

How to run single test

- Single test file

- `$ make test_****`

- Single test or filtering

- `$ export TEST_FILTER=Test_****`

How to create reference data for terminal dump

- Reference data are located in `src/testdir/dumps`
- `term_dumpwrite()`

Links

- <http://rbtnn.hateblo.jp/entry/2019/03/24/054919>
- <https://daisuzu.hatenablog.com/entry/2017/12/02/174002>
- https://qiita.com/m_nish/items/865751bedd1f7d28a740