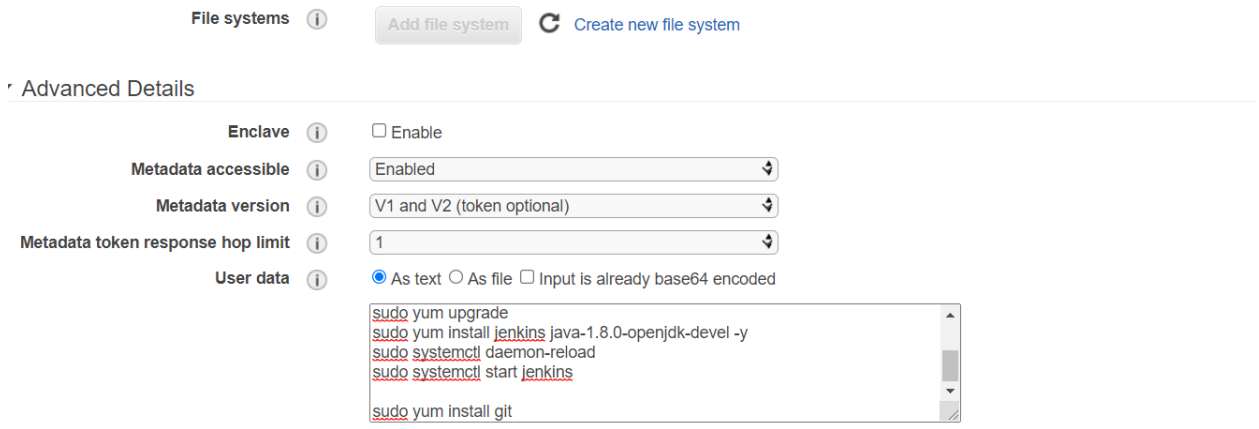
1. Create a new EC2 Amazon Linux instance using the following bootstrap script as shown below.

```
#!/bin/bash
sudo yum update -y
sudo wget -O /etc/yum.repos.d/jenkins.repo
\https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-
stable/jenkins.io.key
sudo yum upgrade
sudo yum install jenkins java-1.8.0-openjdk-devel -y
sudo systemctl daemon-reload
sudo systemctl start Jenkins

sudo yum install git
```

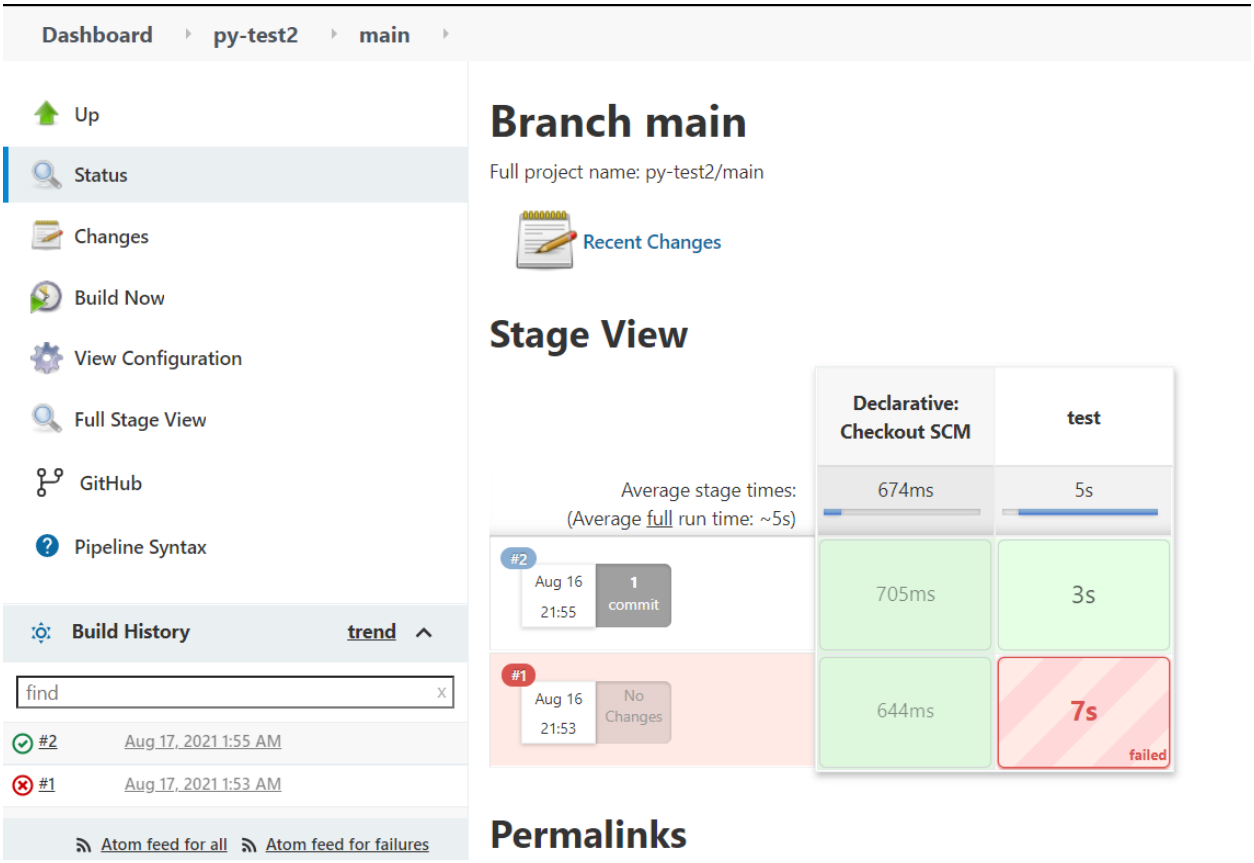


1. For security groups settings, set SSH port of 22 (My ip), a Custom TCP Rule of 8080 and HTTP port of 80.
2. Then launch your EC2 instance
3. After launching ssh into your instance, go to your terminal and enter the directory which contains your SSH keys and do the following:
   cd (directory where your key is located)
   ssh -i (EC2 keys) ec2-user@(public IPV4 address of your instance)
4. Check that Jenkins is running by using the command sudo systemctl status Jenkins.
5. Once Jenkins is running, enter your browser and enter the public IPV4 address of your instance with port 8080. Example 3.54.67.256:8080
6. Then after Jenkins loads, use the command sudo cat /var/lib/jenkins/secrets/initialAdminPassword to get that password you'll need to enter.
7. Paste the password that you got from previous step.
8. Now install suggested plugins for Jenkins

9. Setup up an admin account with your username, password and email.
10. Fork the DEPLOY03_TEST repository and edit your Jenkins file with the following code to create a pipeline with a build, test and deploy stage as shown below.

```
pipeline {
    agent any
    stages {
        stage('test') {
            steps {
                sh '''#! /bin/bash
                python3 -m venv test3
                source test3/bin/activate
                pip install pip --upgrade
                pip install pytest
                py.test --verbose --junit-xml test-reports/results.xml sources/test_calc.py
                '''
                }
                post {
                    always {
                        junit 'test-reports/results.xml'
                    }
                }
            }
    }
}
```

11. Create a py-test to your github repo for Jenkins to read your Jenkins file.
12. Go to your Github account settings and generate a personal access token, copy the value and save it somewhere safe.
13. Go to your Jenkins instance and head to the dashboard to make a new item.
14. Give it a name then select multibranch pipeline.
15. Then add the branch source as Github then click on add and after click on Jenkins.
16. Add your Github username into the username field, and then enter your personal access token into the password field. For ID field enter Jenkins-webhook-id.
17. Then set credentials next to the add button.
18. Then switch to Repository Scan, there adding your Github account user name as owner and selecting the repository needed.
19. Then go to configure to find Scan Repository triggers. There you will click Periodic to set it for 10 minute intervals.
20. Then after building check to make sure build was done successfully as shown below.



NOTE: If the code is entered incorrectly an error can occur as shown in this figure.

21. Figure below shows the original add2vals.py before it was edited.

main DEPLOY03_TEST / sources / add2vals.py / <> Jump to

kura-labs01 Add files via upload

1 contributor

25 lines (21 sloc) | 710 Bytes

```
'''
A simple command line tool that takes 2 values and adds them together using
the calc.py library's 'add2' function.
'''

import sys
import calc

argnumbers = len(sys.argv) - 1

if argnumbers == 2 :
    print("")
    print("The result is " + str(calc.add2(str(sys.argv[1]), str(sys.argv[2]))))
    print("")
    sys.exit(0)

if argnumbers != 2 :
    print("")
    print("You entered " + str(argnumbers) + " value/s.")
    print("")
    print("Usage: 'add2vals X Y' where X and Y are individual values.")
    print("       If add2vals is not in your path, usage is './add2vals X Y'.")
    print("       If unbundled, usage is 'python add2vals.py X Y'.")
    print("")
    sys.exit(1)
```

22. Created an extra 'if argunumbers' which I made equal to 3. Did this because I wanted to make a multiplication function.
23. Then created a print statement with the variable "cal.mul3" while adding and additional str(sys.argv[3]).
24. After I changed the other if statements to "elif" so that if the previous requirements aren't meant the code reads the next argunumber statements. This is shown in the figure below:

main DEPLOY03_TEST / sources / add2vals.py / Jump to

Mastercle Update add2vals.py Latest commi

2 contributors

31 lines (26 sloc) | 959 Bytes

```
'''
A simple command line tool that takes 2 values and adds them together using
the calc.py library's 'add2' function.
'''

import sys
import calc

argnumbers = len(sys.argv) - 1

if argunumbers == 3:
    print("")
    print("The result is " + str(calc.mul3(str(sys.argv[1]), str(sys.argv[2]), str(sys.argv[3]))))
    print("")
    sys.exit(0)

elif argnumbers == 2 :
    print("")
    print("The result is " + str(calc.add2(str(sys.argv[1]), str(sys.argv[2]))))
    print("")
    sys.exit(0)

elif argnumbers != 3 and argnumbers != 2:
    print("")
    print("You entered " + str(argnumbers) + " value/s.")
    print("")
    print("Usage: 'add2vals X Y' where X and Y are individual values. or 'mul3vals' XYZ where xyz are individual values.")
    print("       If add2vals is not in your path, usage is './add2vals X Y'.")
    print("       If unbundled, usage is 'python add2vals.py X Y'.")
    print("")
    sys.exit(1)
```

25. Figure below shows the original calc.py before it was edited.

main ▾ DEPLOY03_TEST / sources / calc.py / <> Jump to ▾

kura-labs01 Add files via upload

1 contributor

28 lines (26 sloc) | 1 KB

```
'''
The 'calc' library contains the 'add2' function that takes 2 values and adds
them together. If either value is a string (or both of them are) 'add2' ensures
they are both strings, thereby resulting in a concatenated result.
NOTE: If a value submitted to the 'add2' function is a float, it must be done so
in quotes (i.e. as a string).
'''

# If 'value' is not an integer, convert it to a float and failing that, a string.
def conv(value):
    try:
        return int(value)
    except ValueError:
        try:
            return float(value)
        except ValueError:
            return str(value)

# The 'add2' function itself
def add2(arg1, arg2):
    # Convert 'arg1' and 'arg2' to their appropriate types
    arg1conv = conv(arg1)
    arg2conv = conv(arg2)
    # If either 'arg1' or 'arg2' is a string, ensure they're both strings.
    if isinstance(arg1conv, str) or isinstance(arg2conv, str):
        arg1conv = str(arg1conv)
        arg2conv = str(arg2conv)
    return arg1conv + arg2conv
```

26. Then add a method so that it uses the correct operations to the variables you'll be assigning.
27. In the figure below I created a method and named it "mul3" since I'll be taking 3 strings and multiplying them.

```
# The 'add2' function itself
def add2(arg1, arg2):
    # Convert 'arg1' and 'arg2' to their appropriate types
    arg1conv = conv(arg1)
    arg2conv = conv(arg2)
    # If either 'arg1' or 'arg2' is a string, ensure they're both strings.
    if isinstance(arg1conv, str) or isinstance(arg2conv, str):
        arg1conv = str(arg1conv)
        arg2conv = str(arg2conv)
    return arg1conv + arg2conv

def mul3(arg1, arg2, arg3):
    arg1conv = conv(arg1)
    arg2conv = conv(arg2)
    arg3conv = conv(arg3)

    if isinstance(arg1conv,str) or isinstance(arg2conv, str) or isinstance(arg3conv, str):
        arg1conv = str(arg1conv)
        arg2conv = str(arg2conv)
        arg3conv = str(arg3conv)
    return arg1conv * arg2conv * arg3conv
```

28. Figure below shows the original calc.py before it was edited

main DEPLOY03_TEST / sources / test_calc.py / <> Jump to

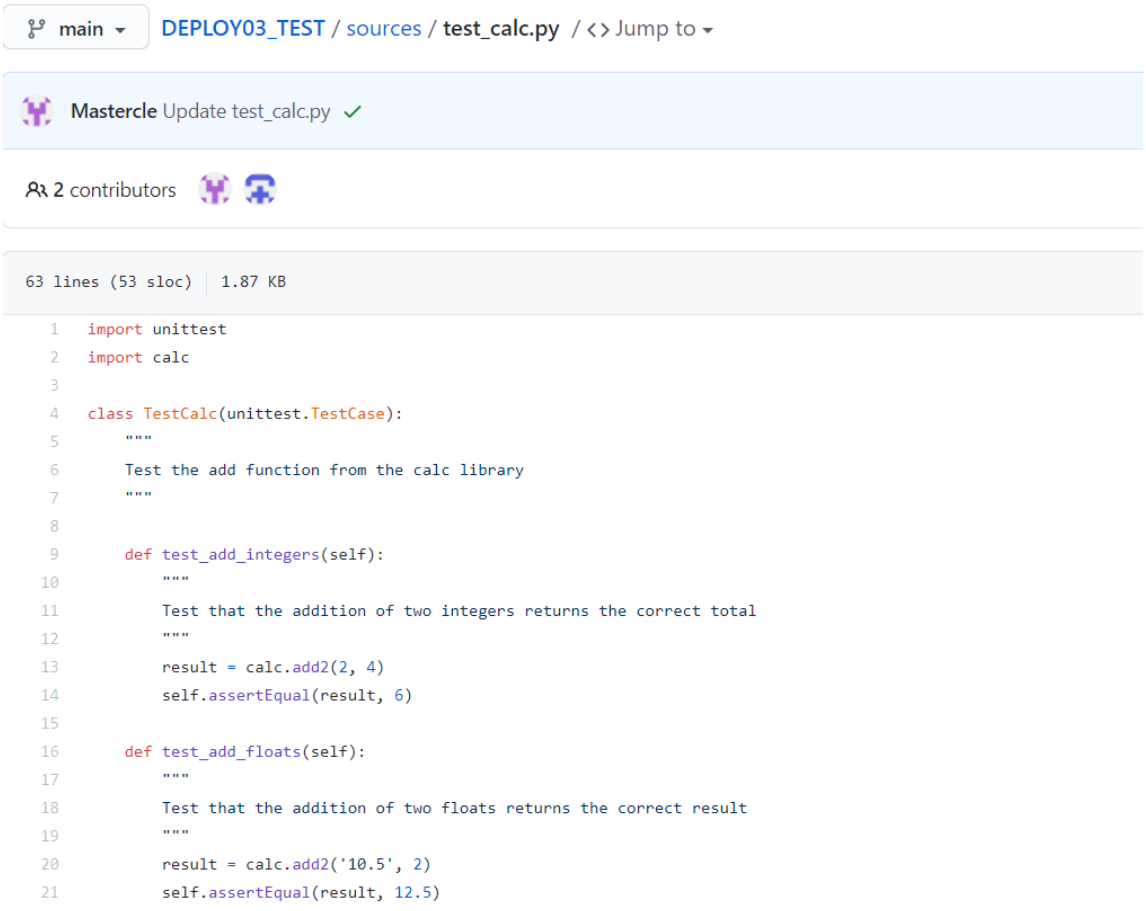
kura-labs01 Add files via upload Latest commit

1 contributor

48 lines (41 sloc) | 1.43 KB

```
import unittest
import calc

class TestCalc(unittest.TestCase):
    """
    Test the add function from the calc library
    """

    def test_add_integers(self):
        """
        Test that the addition of two integers returns the correct total
        """
        result = calc.add2(1, 2)
        self.assertEqual(result, 3)

    def test_add_floats(self):
        """
        Test that the addition of two floats returns the correct result
        """
        result = calc.add2('10.5', 2)
        self.assertEqual(result, 12.5)

    def test_add_strings(self):
        """
        Test the addition of two strings returns the two strings as one
        concatenated string
        """
        result = calc.add2('abc', 'def')
        self.assertEqual(result, 'abcdef')

    def test_add_string_and_integer(self):
        """
        Test the addition of a string and an integer returns them as one
        concatenated string (in which the integer is converted to a string)
        """
        result = calc.add2('abc', 3)
        self.assertEqual(result, 'abc3')

    def test_add_string_and_number(self):
        """
        Test the addition of a string and a float returns them as one
        concatenated string (in which the float is converted to a string)
        """
        result = calc.add2('abc', '5.5')
        self.assertEqual(result, 'abc5.5')

if __name__ == '__main__':
    unittest.main()
```

29. Add a function, in this case I added a new function to test the addition of 2 integers.



main DEPLOY03_TEST / sources / test_calc.py / Jump to

Mastercle Update test_calc.py ✓

2 contributors

63 lines (53 sloc) | 1.87 KB

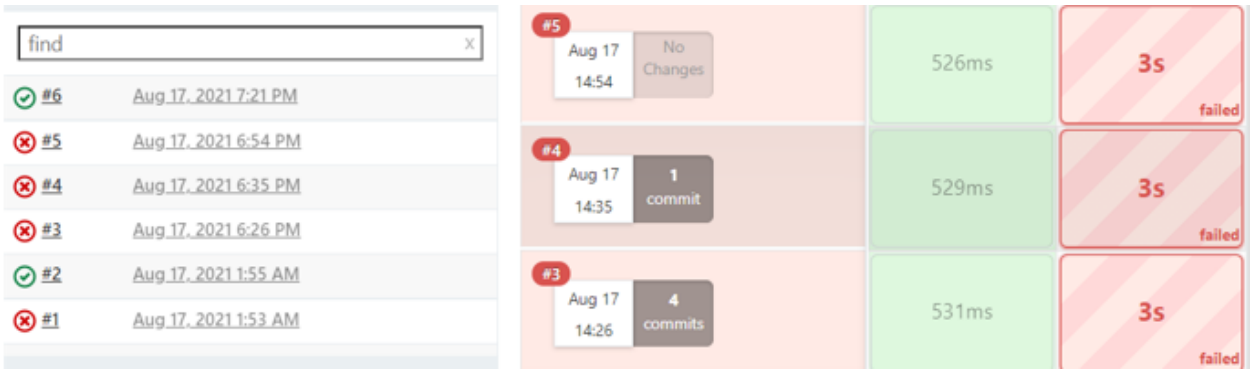
```
1  import unittest
2  import calc
3
4  class TestCalc(unittest.TestCase):
5      """
6      Test the add function from the calc library
7      """
8
9      def test_add_integers(self):
10         """
11         Test that the addition of two integers returns the correct total
12         """
13         result = calc.add2(2, 4)
14         self.assertEqual(result, 6)
15
16     def test_add_floats(self):
17         """
18         Test that the addition of two floats returns the correct result
19         """
20         result = calc.add2('10.5', 2)
21         self.assertEqual(result, 12.5)
```

30. Then added 2 additional functions to test for the multiplication of numbers, both as integers and floats. This is shown in the figure below

```
        concatenated string
        """
        result = calc.add2('abc', 'def')
        self.assertEqual(result, 'abcdef')

    def test_add_string_and_integer(self):
        """
        Test the addition of a string and an integer returns them as one
        concatenated string (in which the integer is converted to a string)
        """
        result = calc.add2('abc', 3)
        self.assertEqual(result, 'abc3')

    def test_add_string_and_number(self):
        """
        Test the addition of a string and a float returns them as one
        concatenated string (in which the float is converted to a string)
        """
        result = calc.add2('abc', '5.5')
        self.assertEqual(result, 'abc5.5')

    def test_multiply_integers(self):
        """
        Test that the multiplication of two integers returns the correct total
        """
        result = calc.mul3(1, 2, 3)
        self.assertEqual(result, 6)

    def test_multiply_floats(self):
        """
        Test that the multiplication of two floats returns the correct result
        """
        result = calc.mul3('5.5', 2, 3)
        self.assertEqual(result, 33)


if __name__ == '__main__':
    unittest.main()
```

Testing and Results

Figure below shows some fails

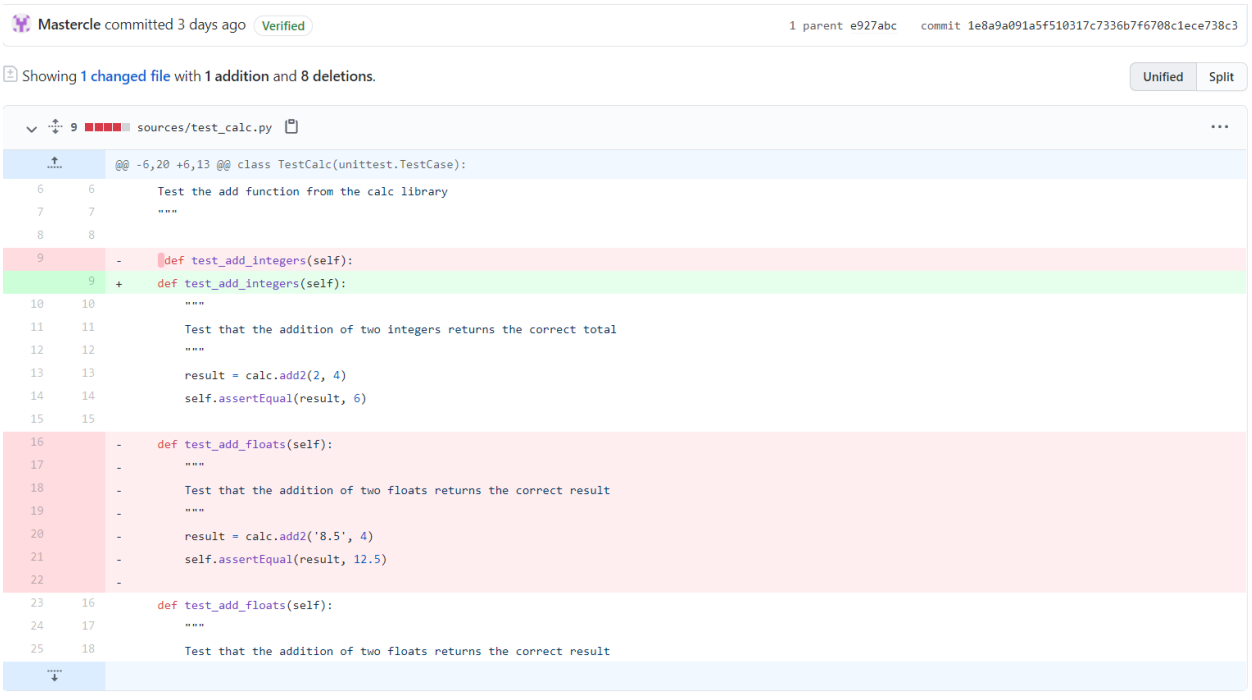


Reasons for fails

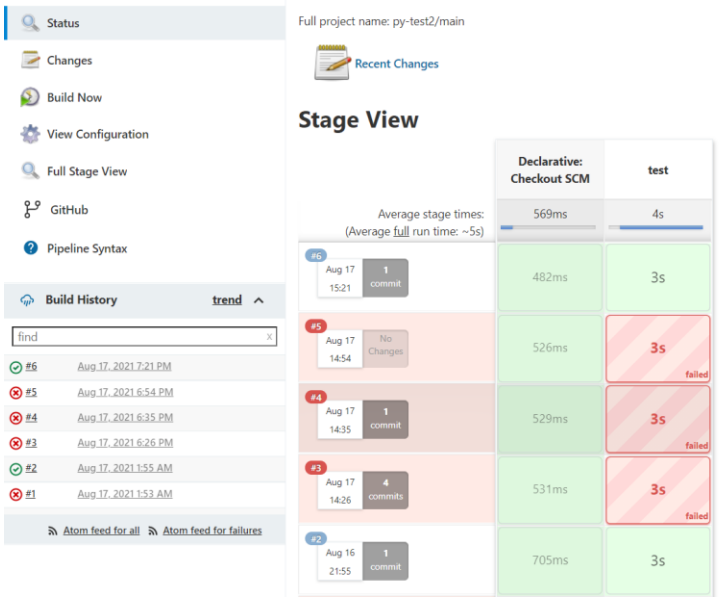
1. Test failed because strings can't be multiplied.
2. Indentation error

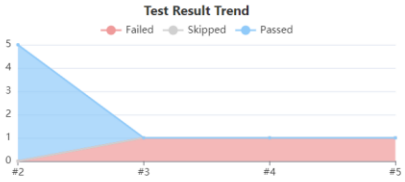
How the fails were fixed:

1. Only tested for integers and floats
2. Indentation and where it was fixed



END RESULT


Status
Changes
Build Now
View Configuration
Full Stage View
GitHub
Pipeline Syntax
Build History
trend
find
#6 Aug 17, 2021 7:21 PM
#5 Aug 17, 2021 6:54 PM
#4 Aug 17, 2021 6:35 PM
#3 Aug 17, 2021 6:26 PM
#2 Aug 17, 2021 1:55 AM
#1 Aug 17, 2021 1:53 AM
Atom feed for all
Atom feed for failures
Full project name: py-test2/main
Recent Changes
Stage View
Declarative: Checkout SCM
test
Average stage times:
(Average full run time: ~5s)
569ms
4s
#6 Aug 17 15:21 1 commit 482ms 3s
#5 Aug 17 14:54 No Changes 526ms 3s failed
#4 Aug 17 14:35 1 commit 529ms 3s failed
#3 Aug 17 14:26 4 commits 531ms 3s failed
#2 Aug 16 21:55 1 commit 705ms 3s


Test Result Trend
Failed
Skipped
Passed
5
4
3
2
1
0
#2
#3
#4
#5