



Objective: Generate a unit test to test the given configuration parameters that are used by the method. Module: Hadoop-common-project/hadoop-common ClassName: `/** * A {@link FileSystem} backed by an FTP client provided by <a href="http://commons.apache.org/net/">Apache Commons Net</a>. * </p> */ @InterfaceAudience.Public @InterfaceStability.Stable public class FTPFileSystem extends FileSystem { Configuration Parameter: <property> <name>fs.ftp.host</name> <value>0.0.0.0</value> <description>FTP filesystem connects to this server</description> </property> <property> <name>fs.ftp.host.port</name> <value>21</value> <description> FTP filesystem connects to fs.ftp.host on this port </description> </property>`

Dependency of the parameters: FTP filesystem connects to fs.ftp.host on this port. A and B, both parameters work together. Method Details: `*** Connect to the FTP server using configuration parameters *** @return An FTPClient instance * @throws IOException */ private FTPClient connect() throws IOException { FTPClient client = null; Configuration conf = getConf(); String host = conf.get(FS_FTP_HOST); int port = conf.getInt(FS_FTP_HOST_PORT, FTP.DEFAULT_PORT); String user = conf.get(FS_FTP_USER_PREFIX + host); String password = conf.get(FS_FTP_PASSWORD_PREFIX + host); client = new FTPClient(); client.connect(host, port); int reply = client.getReplyCode(); if (!FTPReply.isPositiveCompletion(reply)) { throw NetUtils.wrapException(host, port, NetUtils.UNKNOWN_HOST, 0, new ConnectException("Server response " + reply)); } else if (client.login(user, password)) { client.setFileTransferMode(getTransferMode(conf)); client.setFileType(FTP.BINARY_FILE_TYPE); client.setBufferSize(DEFAULT_BUFFER_SIZE); setTimeout(client, conf); setDataConnectionMode(client, conf); } else { throw new IOException("Login failed on server - " + host + ", port - " + port + " as user " + user + ""); } return client; }`

---



In Java, unit tests are commonly written using frameworks like JUnit or TestNG. Below is an example of how you might write a unit test for the `FTPFileSystem.connect()` method. This unit test will validate whether the method correctly uses the `fs.ftp.host` and `fs.ftp.host.port` configuration parameters to establish a connection to an FTP server.

First, you would need to add the JUnit library to your project's build path. Here's a simple unit test example using JUnit 5:

```
java
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

import org.apache.hadoop.conf.Configuration;
import org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPReply;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

public class FTPFileSystemTest {

    private static final String FS_FTP_HOST = "fs.ftp.host";
    private static final String FS_FTP_HOST_PORT = "fs.ftp.host.port";

    private FTPFileSystem ftpFileSystem;

    @Mock
    private FTPClient mockFtpClient;
    @Mock
    private Configuration mockConf;

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);
        ftpFileSystem = new FTPFileSystem() {
            @Override
            FTPClient createFTPClient() {
                return mockFtpClient;
            }

            @Override
            Configuration getConf() {
                return mockConf;
            }
        };
    }

    @Test
    public void testConnect() throws IOException {
        String host = "192.168.1.1";
        int port = 21;
        when(mockConf.get(FS_FTP_HOST)).thenReturn(host);
        when(mockConf.getInt(FS_FTP_HOST_PORT, FTP.DEFAULT_PORT)).thenReturn(port);
        when(mockFtpClient.getReplyCode()).thenReturn(FTPReply.COMMAND_OK);
    }
}
```

```

        FTPClient client = ftpFileSystem.connect();

        verify(mockFtpClient, times(1)).connect(host, port);
        assertNotNull(client, "FTPClient should not be null");
    }

    // Additional tests can be added to simulate and verify failure scenarios such as wrong host, port or credentials.
}

```

In this example, we use Mockito to mock dependencies, which is a common practice for isolating the code under test. You should note that in the `FTPFileSystem` class, the `connect()` method and `getConf()` method are private, which makes them not directly accessible for testing. This would require you to refactor your class to allow testing, such as by changing the visibility of methods, or by adding package-private methods that can be used to inject mocks.

Here's what the above test does:

- It initializes mocks for `FTPClient` and `Configuration`.
- It sets up the mock behavior to return specific values when methods like `get()` and `getInt()` are called.
- It calls the `connect()` method and verifies if the `connect()` method on the `mockFtpClient` was called with the correct parameters.

Additionally, you should test for negative cases, such as what happens if the configuration is incorrect, if the connection fails, if the server returns a negative reply, or if the login fails.

This test assumes you have getters or some way to inject the mocks since the original class did not provide for it. The `createFTPClient()` method is an assumed method that you might need to create in the actual `FTPFileSystem` class to be able to inject the `FTPClient` mock.

Please note that this is a simple unit test and does not cover all possible test cases or error handling you might need. You should expand upon this test to cover the full functionality of your method.



these are example test cases: `@Test public void testCreateWithWritePermissions() throws Exception { BaseUser user = server.addUser("test", "password", new WritePermission()); Configuration configuration = new Configuration(); configuration.set("fs.defaultFS", "ftp://"); configuration.set("fs.ftp.host", "localhost"); configuration.setInt("fs.ftp.host.port", server.getPort()); configuration.set("fs.ftp.user.hostname", user.getName()); configuration.set("fs.ftp.password.hostname", user.getPassword()); configuration.setBoolean("fs.ftp.impl.disable.cache", true); FileSystem fs = FileSystem.get(configuration); byte[] bytesExpected = "hello world".getBytes(StandardCharsets.UTF_8); try (FSDataOutputStream outputStream = fs.create(new Path("test1.txt"))) { outputStream.write(bytesExpected); } try (FSDaInputStream input = fs.open(new Path("test1.txt"))) { assertEquals(bytesExpected, equalTo(IOUtils.readFullyToByteArray(input))); } } @Test public void testCreateWithoutWritePermissions() throws Exception { BaseUser user = server.addUser("test", "password"); Configuration configuration = new Configuration(); configuration.set("fs.defaultFS", "ftp://"); configuration.set("fs.ftp.host", "localhost"); configuration.setInt("fs.ftp.host.port", server.getPort()); configuration.set("fs.ftp.user.hostname", user.getName()); configuration.set("fs.ftp.password.hostname", user.getPassword()); configuration.setBoolean("fs.ftp.impl.disable.cache", true); FileSystem fs = FileSystem.get(configuration); byte[] bytesExpected = "hello world".getBytes(StandardCharsets.UTF_8); LambdaTestUtils.intercept( IOException.class, "Unable to create file: test1.txt, Aborting", () -> { try (FSDaInputStream out = fs.create(new Path("test1.txt"))) { out.write(bytesExpected); } } ); }`



These test cases are designed to test the `FTPFileSystem` within the context of Hadoop's `FileSystem` abstraction. They appear to be functional tests rather than traditional unit tests, as they are interacting with an actual or emulated FTP server (`server` object) and performing read/write operations.

Here's a breakdown of what each test case is doing:

#### 1. `testCreateWithWritePermissions`:

- It sets up a user on the FTP server with write permissions.
- It creates a `Configuration` object and sets various parameters, including FTP host details and user credentials.
- It gets an instance of the `FileSystem` and writes "hello world" to a file named "test1.txt".
- Then it reads the file back to verify the content is as expected.

#### 2. `testCreateWithoutWritePermissions`:

- It sets up a user on the FTP server without write permissions.
- It configures the `FileSystem` similar to the previous test.
- It attempts to write to "test1.txt" and expects an `IOException` to be thrown because the user doesn't have write permission.

To make these test cases work correctly, you need to ensure the `server` object is properly initialized and represents the FTP server interface for the tests. You'll also need to ensure the `LambdaTestUtils.intercept` utility is available to intercept and assert exceptions.

It's worth noting that these test cases assume that the `server.addUser` method is a way to configure an underlying FTP server to add a user with or without write permissions. This suggests that these tests might be a part of an integration testing suite where an in-memory or local FTP server is being used for testing purposes.

For a complete test, you'd also need to include setup and teardown methods that start and stop the FTP server before and after the tests. Here's a skeleton of how those might look with JUnit 5 annotations:

```
java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;

// Other imports...

public class FTPFileSystemTest {

    private Server server; // This would be your FTP server test harness

    @BeforeEach
    public void setUp() throws Exception {
        // Start your FTP server and perform any setup
        server = new Server();
        server.start();
    }

    @AfterEach
    public void tearDown() throws Exception {
        // Stop the FTP server and clean up resources
        if (server != null) {
            server.stop();
        }
    }

    // ... (your test methods here)
}
```

Ensure that the Server class in your tests correctly manages the lifecycle of the FTP server, with methods like `start()` and `stop()`. If you're using an embedded FTP server for testing, like Apache MINA or MockFtpServer, these would typically be provided by the server class.



You:

can you rewrite the entire test suite including all information above



ChatGPT:

Certainly! Here's an example of how the test suite could be written, including both test cases provided above. This suite uses JUnit 5 annotations and assumes you're using some kind of in-memory or local FTP server (referred to here as `Server`) for testing purposes. Make sure to replace `Server` with the actual FTP server class you're using for tests.

```
java
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.equalTo;

import java.io.IOException;
import java.nio.charset.StandardCharsets;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hdfs.server.namenode.permission.BaseUser;
import org.apache.hadoop.hdfs.server.namenode.permission.WritePermission;
import org.apache.hadoop.io.IOUtils;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class FTPFileSystemTest {

    private Server server; // Replace with your actual FTP server class
    private Configuration configuration;
```

```

@BeforeEach
public void setUp() throws Exception {
    // Initialize and start the FTP server
    server = new Server(); // Server should be your mock or in-memory FTP server
    server.start();

    // Common configuration setup for both tests
    configuration = new Configuration();
    configuration.set("fs.defaultFS", "ftp:///");
    configuration.set("fs.ftp.host", "localhost");
    configuration.setInt("fs.ftp.host.port", server.getPort());
    configuration.setBoolean("fs.ftp.impl.disable.cache", true);
}

@AfterEach
public void tearDown() throws Exception {
    // Stop the FTP server and clean up resources
    if (server != null) {
        server.stop();
    }
}

@Test
public void testCreateWithWritePermissions() throws Exception {
    BaseUser user = server.addUser("test", "password", new WritePermission());
    configuration.set("fs.ftp.user.localhost", user.getName());
    configuration.set("fs.ftp.password.localhost", user.getPassword());

    FileSystem fs = FileSystem.get(configuration);
    byte[] bytesExpected = "hello world".getBytes(StandardCharsets.UTF_8);
    try (FSDatOutputStream outputStream = fs.create(new Path("test1.txt"))) {
        outputStream.write(bytesExpected);
    }
    try (FSDatInputStream input = fs.open(new Path("test1.txt"))) {
        byte[] bytesRead = IOUtils.readFullyToByteArray(input);
        assertThat(bytesExpected, equalTo(bytesRead));
    }
}

@Test
public void testCreateWithoutWritePermissions() throws Exception {
    BaseUser user = server.addUser("test", "password");
    configuration.set("fs.ftp.user.localhost", user.getName());
    configuration.set("fs.ftp.password.localhost", user.getPassword());

    FileSystem fs = FileSystem.get(configuration);
    byte[] bytesExpected = "hello world".getBytes(StandardCharsets.UTF_8);

    IOException exception = assertThrows(IOException.class, () -> {
        try (FSDatOutputStream out = fs.create(new Path("test1.txt"))) {
            out.write(bytesExpected);
        }
    });
    String errorMessage = "Unable to create file: test1.txt, Aborting";
    assertTrue(exception.getMessage().contains(errorMessage));
}
}

```

Some important notes for this code:

- Server should be replaced with whatever FTP server implementation you're using for testing.
- The `server.addUser` method is assumed to be a method that adds a user to your test FTP server with appropriate permissions.
- `WritePermission` is a hypothetical class or object that signifies the user has write permissions.
- The `LambdaTestUtils.intercept` method you mentioned is not a standard JUnit or Mockito method. Instead, I've used `assertThrows` from JUnit 5 to expect an `IOException`.
- The error message expected in `testCreateWithoutWritePermissions` should be adjusted to match the actual message thrown by your `FTPFileSystem`.
- This suite assumes that the setup and teardown of the FTP server (`server.start()` and `server.stop()`) will correctly initialize and clean up the server for each test.

Remember to import `assertThrows` and `assertTrue` from the `org.junit.jupiter.api.Assertions` package if you decide to use those assertions.



```
[ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[60,38]
cannot find symbol symbol: class IOException location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-
common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[45,27] cannot find symbol symbol: method
openMocks(org.apache.hadoop.fs.ftp.FTPFileSystemTest) location: class org.mockito.MockitoAnnotations [ERROR] /home/nvadde2/hadoop/hadoop-
common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[53,27] getConf() in <anonymous
org.apache.hadoop.fs.ftp.FTPFileSystemTest$1> cannot implement getConf() in org.apache.hadoop.conf.Configurable attempting to assign weaker
access privileges; was public [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-
common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[47,13] method does not override or implement a method from a supertype
[ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[64,48]
cannot find symbol symbol: variable FTP location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-
project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[67,41] connect() has private access in
org.apache.hadoop.fs.ftp.FTPFileSystem [INFO] 6 errors [INFO] ----- [INFO] ----- [INFO] BUILD FAILURE [INFO] ----- [INFO] Total time: 33.580 s [INFO] Finished at:
2023-11-02T14:52:26-05:00 [INFO] ----- [ERROR] Failed to execute goal
org.apache.maven.plugins:maven-compiler-plugin:3.10.1:testCompile (default-testCompile) on project hadoop-common: Compilation failure: Compilation
failure: [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:
[60,38] cannot find symbol [ERROR] symbol: class IOException [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR]
/home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[45,27] cannot find
symbol [ERROR] symbol: method openMocks(org.apache.hadoop.fs.ftp.FTPFileSystemTest) [ERROR] location: class org.mockito.MockitoAnnotations
[ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[53,27]
getConf() in <anonymous org.apache.hadoop.fs.ftp.FTPFileSystemTest$1> cannot implement getConf() in org.apache.hadoop.conf.Configurable [ERROR]
attempting to assign weaker access privileges; was public [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-
common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[47,13] method does not override or implement a method from a supertype
[ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[64,48]
cannot find symbol [ERROR] symbol: variable FTP [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR]
/home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[67,41] connect() has
private access in org.apache.hadoop.fs.ftp.FTPFileSystem
```



It looks like you're experiencing several compilation issues. Let's go through each of the errors and correct them:

- 1. Cannot find symbol: IOException** You need to import the IOException class. Add the following import statement at the top of your test file:

```
java
import java.io.IOException;
```

- 2. Cannot find symbol: openMocks** The openMocks method is part of the Mockito library, and it's used to initialize mocks. It seems you're not using it in the provided code. If you are not using Mockito, you can remove any reference to openMocks. If you are using Mockito and want to initialize mocks, ensure you've imported the correct method from Mockito:

```
java
import org.mockito.MockitoAnnotations;
```

- 3. getConf() attempting to assign weaker access privileges; was public** This error indicates that there's a mismatch in the visibility of the getConf() method. The method getConf() in your anonymous inner class or lambda must match the visibility of the method it's trying to override. Ensure you're not reducing the visibility of an overridden method.

- 4. Method does not override or implement a method from a supertype** This error suggests there's an @Override annotation on a method that doesn't actually override a method from a superclass or interface. Make sure that your method signatures match exactly with those of the superclass/interface you're intending to override.

- 5. Cannot find symbol: FTP** You must import the FTP class or correctly reference the FTP constant. If FTP is a constant in FTPFileSystem, you should use it like FTPFileSystem.FTP, otherwise, ensure you have the right import:

```
java
import org.apache.commons.net.ftp.FTP; // or the correct path to your FTP constant
```

- 6. connect() has private access in FTPFileSystem** The connect() method is private and therefore cannot be accessed from outside the FTPFileSystem class. If you need to test this method, you will need to either make the method package-private or protected, or you will need to use reflection to access it. However, the latter is not recommended for regular unit testing practices as it breaks encapsulation.

If you've provided a direct snippet from your test file, it also seems like you're trying to call connect() directly from your test which won't be possible with its private access modifier.

Here is the revised import section for your test class:

```
java
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.equalTo;
```

```

import java.io.IOException;
import java.nio.charset.StandardCharsets;

// ... other import statements ...

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.commons.net.ftp.FTP; // Import your FTP constants or classes correctly

// ... potentially more imports ...

```

Please make sure that all classes and methods used in your test file are imported correctly and that you're not trying to access private methods or reduce method visibility in overridden methods. If the FTP constant is within your `FTPFileSystem`, you'll have to use it with the class name prefix, i.e., `FTPFileSystem.FTP`.



You:

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.10.1:testCompile (default-testCompile) on project hadoop-common: Compilation failure: Compilation failure: [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[21,27] cannot find symbol [ERROR] symbol: class Matchers [ERROR] location: package org.hamcrest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[21,1] static import only from classes and interfaces [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[31,57] package org.apache.hadoop.hdfs.server.namenode.permission does not exist [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[32,57] package org.apache.hadoop.hdfs.server.namenode.permission does not exist [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[40,13] cannot find symbol [ERROR] symbol: class Server [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[46,22] cannot find symbol [ERROR] symbol: class Server [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[67,9] cannot find symbol [ERROR] symbol: class BaseUser [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[67,64] cannot find symbol [ERROR] symbol: class WritePermission [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[78,39] cannot find symbol [ERROR] symbol: method equalTo(byte[]) [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[84,9] cannot find symbol [ERROR] symbol: class BaseUser [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[91,33] cannot find symbol [ERROR] symbol: method assertThrows(java.lang.Class<java.io.IOException>,()->{ try [...] ; } }) [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[98,9] cannot find symbol [ERROR] symbol: method assertTrue(boolean) [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest
```



These errors indicate that several classes and methods cannot be found by the compiler. This usually means that the required import statements are missing, incorrect, or that the relevant dependencies are not included in your project's build path. Here's how you can resolve each error:

- 1. Cannot find symbol: Matchers** You need to ensure that Hamcrest is included in your project dependencies and that you import the Matchers class correctly. The correct import statement should be:

```
java
import static org.hamcrest.Matchers.equalTo;
```

- 2. Static import only from classes and interfaces** This means that you are trying to statically import methods or fields from a package rather than a class or an interface. Make sure your static imports are pointing to a class or interface.

- 3. Package does not exist** The package `org.apache.hadoop.hdfs.server.namenode.permission` does not exist. Ensure that you are importing the correct packages. If these are custom classes, you need to check if the package path is correct and that they are included in the classpath.

- 4. Cannot find symbol: Server** The compiler cannot find the Server class. You must import this class or ensure it is defined somewhere in your project.

- 5. Cannot find symbol: BaseUser, WritePermission** It seems like you are referencing BaseUser and WritePermission classes that the compiler cannot find. Ensure that these classes exist in your project and are properly imported.

6. **Cannot find symbol: equalTo(byte[])** The equalTo method is a Hamcrest matcher that should be statically imported. The correct import statement would be as mentioned above in point 1.
7. **Cannot find symbol: assertThrows and assertTrue** These are methods from the JUnit framework. Make sure you have JUnit as a dependency in your project and import these methods correctly:

```
java
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
```

Here's an example of how your imports might look:

```
java
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.equalTo;

import org.junit.jupiter.api.Test;
// ... other necessary imports ...

// Make sure all custom classes are imported or defined in your project
```

If you are unsure about the dependencies or where the classes like BaseUser, WritePermission, and Server are coming from, you will need to investigate your project structure or consult with your project documentation to find the correct packages or dependencies that need to be included in your build path.



[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.10.1:testCompile (default-testCompile) on project hadoop-common: Compilation failure: Compilation failure: [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[21,27] cannot find symbol [ERROR] symbol: class Matchers [ERROR] location: package org.hamcrest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[21,1] static import only from classes and interfaces [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[31,57] package org.apache.hadoop.hdfs.server.namenode.permission does not exist [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[32,57] package org.apache.hadoop.hdfs.server.namenode.permission does not exist [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[40,27] cannot find symbol [ERROR] symbol: class Matchers [ERROR] location: package org.hamcrest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[40,1] static import only from classes and interfaces [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[45,13] cannot find symbol [ERROR] symbol: class Server [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[51,22] cannot find symbol [ERROR] symbol: class Server [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[72,9] cannot find symbol [ERROR] symbol: class BaseUser [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[72,64] cannot find symbol [ERROR] symbol: class WritePermission [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[83,39] cannot find symbol [ERROR] symbol: method equalTo(byte[]) [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[89,9] cannot find symbol [ERROR] symbol: class BaseUser [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest



The errors you are encountering during the Maven compilation process for the Hadoop project suggest a few issues with missing dependencies and perhaps some incorrect import statements.

Here's how you can approach resolving each of these errors:

1. **Cannot find symbol: Matchers** This class is part of the Hamcrest library, which is often used alongside JUnit for assertions in tests. Ensure that you have Hamcrest as a dependency in your Maven pom.xml file. You should also check that you are importing the class correctly, as follows:

```
java
import org.hamcrest.Matchers;
// or for a static import
import static org.hamcrest.Matchers.*;
```

Add the Hamcrest dependency to your pom.xml if it's missing:

```

xml
<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest</artifactId>
  <version>2.2</version> <!-- Use the appropriate version -->
  <scope>test</scope>
</dependency>

```

2. **Static import only from classes and interfaces** Make sure you are statically importing methods or fields from a class or interface. You cannot statically import from a package.
3. **Package does not exist** The errors relating to the org.apache.hadoop.hdfs.server.namenode.permission package not existing may be due to a few reasons. You might be working with a version of Hadoop that does not include those packages, the packages might be from a custom module that hasn't been included in the classpath, or you might have incorrect package names in your import statements. Check your project structure and the Hadoop documentation for the correct package names.
4. **Cannot find symbol: Server, BaseUser, WritePermission** If these classes are part of third-party libraries, make sure those libraries are included as dependencies in your pom.xml. If they are part of your project but in different modules, make sure those modules are correctly referenced. If these are classes within the same module, check that the package names and class names are correct in your import statements and that the classes are in the correct directory according to their package name.
5. **Cannot find symbol: equalTo(byte[])** This method is part of the Hamcrest library and should be resolved by correctly importing the Matchers class as shown above. Make sure that the method is being used correctly as well, with the appropriate arguments.
6. **Maven Dependency Issues** If dependencies are missing from your pom.xml, you'll need to add them. For example, if Server, BaseUser, and WritePermission are classes from a library you are using, you need to find out which library that is and add it as a dependency in your pom.xml. If these are custom classes that are a part of the Hadoop codebase, ensure they are compiled and packaged correctly.
7. **Compilation Environment** Ensure that the compilation environment is correctly set up with all the necessary environment variables and paths (like JAVA\_HOME and M2\_HOME), and that you are using the correct Maven and JDK versions for compiling the Hadoop project.

Make sure you run mvn clean install or mvn clean package to rebuild the project after making changes to the pom.xml file or updating your imports. If these classes are custom and supposed to be part of the hadoop-common project itself, then it might be a project structure issue or the classes are not located where the build expects them to be.



You:  
can you rewrite test so you do not change the pom.xml



Certainly, to rewrite the test without changing the pom.xml, you would need to:

- Remove the dependencies on the missing classes.
- Replace any utility methods from these classes with equivalent code that does not rely on them.

However, I must emphasize that if your test relies on certain behaviors or utilities provided by these classes, then removing them could change the nature of the test. Ideally, you should have all the necessary dependencies properly declared in your build configuration.

Here is a simplified example of how you might adjust a test that's failing due to missing dependencies. Please note that without the actual test code, I can only provide a general direction:

```

java
// Assuming that Matchers and assertThrows are not available.
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

// Other imports ...

public class FTPFileSystemTest {

    // ... other test code ...

    @Test
    public void testSomeFunctionality() {
        byte[] expected = ...; // Whatever the expected bytes are
        byte[] actual = ...; // Get the actual result from the code under test

        // Instead of using Matchers.equalTo, directly use assertions
        assertTrue(Arrays.equals(expected, actual));
    }
}

```

```

// Instead of using assertThrows, use a try-catch block
try {
    // Some code that is expected to throw an IOException
    fail("IOException was expected to be thrown");
} catch (IOException expectedException) {
    // Possibly check the message or other properties of the exception
    assertEquals("Expected exception message", expectedException.getMessage());
}

// Continue with the test code...
}

// ... other test methods ...
}

```

This example uses only JUnit's assertions and a manual try-catch to handle expected exceptions, which should be available without additional dependencies.

For other missing classes like Server, BaseUser, and WritePermission, if they are custom classes that are part of the hadoop-common project, you would need to mock or stub out their behavior, or include the necessary code directly in your test if possible.

Remember, by doing this, you might be changing the intent or reducing the coverage of the test, which is not a best practice. The proper fix is to have all necessary dependencies declared and available.



You:

please rewrite the entire test case again



ChatGPT:

Rewriting the entire test case without changing the pom.xml means that we need to provide alternatives for all the missing classes and methods while still trying to maintain the integrity of the test. Below is a pseudo-code example based on the error messages you've provided. Keep in mind that without the actual content of the FTPFileSystemTest.java file, I can only provide an approximate rewrite that assumes certain behaviors of the original test.

```

java
import org.apache.hadoop.fs.ftp.FTPFileSystem;
import org.junit.Before;
import org.junit.Test;
import org.junit.Assert;

import java.io.IOException;

// Other necessary imports...

public class FTPFileSystemTest {

    private FTPFileSystem ftpFileSystem;

    @Before
    public void setUp() {
        // Initialize your FTPFileSystem here
        ftpFileSystem = new FTPFileSystem();
        // Do any necessary configuration
    }

    @Test
    public void testFileOperations() {
        // Simulate some file operations, like creating a file, writing to it, etc.

        // Verify operations - replace Matchers.equalTo with direct assertions
        byte[] expectedContent = {/* ... expected content ... */};
        byte[] actualContent = {/* ... actual content, probably read from a file ... */};
        Assert.assertArrayEquals("The content of the file does not match the expected content.", expectedContent, actualContent);

        // If there are permissions related checks, you will have to simulate them without using the unavailable classes
    }

    @Test
    public void testExceptionHandling() {

```

```

// If there are tests for exceptions, replace assertThrows with try-catch blocks
try {
    // Code that is expected to throw an exception
    Assert.fail("Expected IOException was not thrown.");
} catch (IOException e) {
    // Assert that the message or other properties of the exception are what you expect
    Assert.assertEquals("Expected exception message", e.getMessage());
}

@Test
public void testFilePermission() {
    // Since the classes for permissions are not available, you might need to mock the behavior
    // if those classes are used to check permissions.

    // Assume that we have a method checkWritePermission() to test
    boolean hasPermission = /* call the method or simulate the permission check */;
    Assert.assertTrue("User should have write permission", hasPermission);
}

// Add more test methods as needed
}

```

Please adjust the pseudo-code to fit the actual test logic. The aim is to replace all external dependencies with standard Java/JUnit functionality. This may not be straightforward for all cases, especially if the original tests relied heavily on the functionality provided by the external classes. In some situations, rewriting the tests may involve writing your own utility methods or significantly altering the test logic. Remember, the best practice would still be to include the necessary dependencies in your pom.xml to ensure that your tests are thorough and reliable.



You:

write an actual test case that i can use to compile and test the configuration parameters listed



ChatGPT:

Certainly, we can write a test case for testing configuration parameters in the FTPFileSystem without relying on external matchers and permission classes. However, without specific information on what methods and configuration parameters are available or need to be tested within the FTPFileSystem class, the following example will focus on a general approach to testing getters and setters for configuration parameters.

This example will assume that your FTPFileSystem class has methods to set and get configuration parameters. I will also include a simple method to simulate establishing a connection to an FTP server that will be tested for throwing an IOException.

```

java
import org.apache.hadoop.fs.ftp.FTPFileSystem;
import org.junit.Before;
import org.junit.Test;
import org.junit.Assert;

import java.io.IOException;

public class FTPFileSystemTest {

    private FTPFileSystem ftpFileSystem;

    @Before
    public void setUp() {
        // Initialize your FTPFileSystem here
        ftpFileSystem = new FTPFileSystem();
        // Configure the object with some default test parameters if necessary
        ftpFileSystem.setConf(new Configuration());
    }

    @Test
    public void testConfigurationParameters() {
        // Assuming there are setter and getter methods for configuration parameters.
        String expectedHost = "localhost";
        ftpFileSystem.setHost(expectedHost);
        String actualHost = ftpFileSystem.getHost();
        Assert.assertEquals("The host configuration parameter was not set or retrieved correctly.", expectedHost, actualHost

        int expectedPort = 21;
    }
}

```

```

        ftpFileSystem.setPort(expectedPort);
        int actualPort = ftpFileSystem.getPort();
        Assert.assertEquals("The port configuration parameter was not set or retrieved correctly.", expectedPort, actualPort
    }

    @Test(expected = IOException.class)
    public void testFtpConnectionFailure() throws IOException {
        // This test assumes there's a connect method that throws IOException on failure
        // For the test, we configure it with invalid parameters to ensure the exception is thrown
        ftpFileSystem.setHost("invalidhost");
        ftpFileSystem.setPort(-1); // assuming invalid port to simulate connection failure
        ftpFileSystem.connect();
        // If an IOException is thrown, the test will pass.
    }

    // Other tests can be added similarly to cover different aspects of the FTPFileSystem functionality.
}

```

Please note that I had to assume method names like `setHost`, `getHost`, `setPort`, `getPort`, and `connect`. You would need to replace these with the actual methods available in your `FTPFileSystem` class.

Also, for more sophisticated behavior, you might have to mock the underlying FTP client or server, depending on what exactly you need to test, which might require additional libraries like Mockito.

If you have the actual method names and functionalities, please replace my assumed methods with the real ones, and the test case can be further refined to suit your needs.



You:

[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.10.1:testCompile (default-testCompile) on project hadoop-common: Compilation failure: Compilation failure: [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[123,35] cannot find symbol [ERROR] symbol: class Configuration [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[130,22] cannot find symbol [ERROR] symbol: method setHost(java.lang.String) [ERROR] location: variable ftpFileSystem of type org.apache.hadoop.fs.ftp.FTPFileSystem [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[131,42] cannot find symbol [ERROR] symbol: method getHost() [ERROR] location: variable ftpFileSystem of type org.apache.hadoop.fs.ftp.FTPFileSystem [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[135,22] cannot find symbol [ERROR] symbol: method setPort(int) [ERROR] location: variable ftpFileSystem of type org.apache.hadoop.fs.ftp.FTPFileSystem [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[136,39] cannot find symbol [ERROR] symbol: method getPort() [ERROR] location: variable ftpFileSystem of type org.apache.hadoop.fs.ftp.FTPFileSystem [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[144,22] cannot find symbol [ERROR] symbol: method setHost(java.lang.String) [ERROR] location: variable ftpFileSystem of type org.apache.hadoop.fs.ftp.FTPFileSystem [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[145,22] cannot find symbol [ERROR] symbol: method setPort(int) [ERROR] location: variable ftpFileSystem of type org.apache.hadoop.fs.ftp.FTPFileSystem [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[146,22] connect() has private access in org.apache.hadoop.fs.ftp.FTPFileSystem [ERROR] -> [Help 1] [ERROR] [ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch. [ERROR] Re-run Maven using the -X switch to enable full debug logging. [ERROR] [ERROR] For more information about the errors and possible solutions, please read the following articles: [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException



ChatGPT:

It seems there are several issues with the provided code snippet:

1. The `Configuration` class cannot be found, likely because the import statement for it is missing.
2. Methods such as `setHost`, `getHost`, `setPort`, `getPort`, and `connect` are not found or not accessible in the `FTPFileSystem` class. This could mean that such methods do not exist, are named differently, or `connect` method is private and therefore not accessible from the test.

Without knowing the exact API of the `FTPFileSystem` class, I can only provide a very generic example. I'll revise the test to not use any methods that don't exist and ensure the `Configuration` class is properly imported. Since we're not sure about the configuration methods, I will omit these.

For the `connect` method, if it's private and you need to test it, you would have to use reflection to access it, which is not a recommended practice for unit tests. Instead, you should test the public methods that use `connect` internally.

Here is a revised version without assuming the existence of these methods:

```

java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.junit.Before;
import org.junit.Test;
import org.junit.Assert;

```

```

import java.io.IOException;
import java.net.URI;

public class FTPFileSystemTest {

    private FileSystem fs;

    @Before
    public void setUp() throws IOException {
        // Set up the FTPFileSystem with a URI and configuration if needed.
        URI ftpUri = URI.create("ftp://user:password@localhost");
        fs = FileSystem.get(ftpUri, new Configuration());
    }

    @Test
    public void testConfigurationParameters() {
        // Assume we want to check if the file system is the correct type.
        Assert.assertTrue("FileSystem is not an FTPFileSystem", fs instanceof FTPFileSystem);
    }

    // Additional tests go here to cover other aspects of the FTPFileSystem functionality.
}

```

Note that the `setUp` method now creates a `FileSystem` object using a `URI` that points to an `FTP` location. This is a more typical way of working with file systems in Hadoop, where you don't directly set host and port but rather work with `URIs`. Please adjust the `URI` to match the `FTP` server you intend to use for testing.

Please ensure that the rest of your `FTP` file system class and its capabilities are being tested through their public API. If there are specific methods that you are aware of and want to test, you should include those in your test cases while ensuring they are publicly accessible.

---



You:

For more context on the test case: import java.io.FileNotFoundException; import java.io.IOException; import java.io.InputStream; import java.io.OutputStream; import java.net.ConnectException; import java.net.URI; import org.apache.hadoop.classification.VisibleForTesting; import org.apache.hadoop.util.Preconditions; import org.apache.commons.net.ftp.FTP; import org.apache.commons.net.ftp.FTPClient; import org.apache.commons.net.ftp.FTPFile; import org.apache.commons.net.ftp.FTPReply; import org.apache.hadoop.classification.InterfaceAudience; import org.apache.hadoop.classification.InterfaceStability; import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.FSDaInputStream; import org.apache.hadoop.fs.FSDaOutputStream; import org.apache.hadoop.fs.FileAlreadyExistsException; import org.apache.hadoop.fs.FileStatus; import org.apache.hadoop.fs.FileSystem; import org.apache.hadoop.fs.ParentNotDirectoryException; import org.apache.hadoop.fs.Path; import org.apache.hadoop.fs.permission.FsAction; import org.apache.hadoop.fs.permission.FsPermission; import org.apache.hadoop.io.IOUtils; import org.apache.hadoop.net.NetUtils; import org.apache.hadoop.util.Progressable; import org.slf4j.Logger; import org.slf4j.LoggerFactory; /\* \* A {@link FileSystem} backed by an FTP client provided by <a href="http://commons.apache.org/net/">Apache Commons Net</a>. \*/ @InterfaceAudience.Public @InterfaceStability.Stable public class FTPFileSystem extends FileSystem { public static final Logger LOG = LoggerFactory.getLogger(FTPFileSystem.class); public static final int DEFAULT\_BUFFER\_SIZE = 1024 \* 1024; public static final int DEFAULT\_BLOCK\_SIZE = 4 \* 1024; public static final long DEFAULT\_TIMEOUT = 0; public static final String FS\_FTP\_USER\_PREFIX = "fs.ftp.user."; public static final String FS\_FTP\_HOST = "fs.ftp.host"; public static final String FS\_FTP\_HOST\_PORT = "fs.ftp.host.port"; public static final String FS\_FTP\_PASSWORD\_PREFIX = "fs.ftp.password"; public static final String FS\_FTP\_DATA\_CONNECTION\_MODE = "fs.ftp.data.connection.mode"; public static final String FS\_FTP\_TRANSFER\_MODE = "fs.ftp.transfer.mode"; public static final String E\_SAME\_DIRECTORY\_ONLY = "only same directory renames are supported"; public static final String FS\_FTP\_TIMEOUT = "fs.ftp.timeout"; private URI uri; /\* \* Return the protocol scheme for the FileSystem. \*/ @Override protected String getScheme() { return "ftp"; } /\* \* Get the default port for this FTPFileSystem. \*/ @Override protected int getDefaultPort() { return FTP.DEFAULT\_PORT; } @Override public void initialize(URI uri, Configuration conf) throws IOException { // get super.initialize(uri, conf); // get host information from uri (overrides info in conf) String host = uri.getHost(); host = (host == null) ? conf.get(FS\_FTP\_HOST, null) : host; if (host == null) { throw new IOException("Invalid host specified"); } conf.set(FS\_FTP\_HOST, host); // get port information from uri, (overrides info in conf) int port = uri.getPort(); if (port == -1) { port = conf.getInt(FS\_FTP\_HOST\_PORT, FTP.DEFAULT\_PORT); } conf.setInt(FS\_FTP\_HOST\_PORT, port); // get user/password information from URI (overrides info in conf) String userAndPassword = uri.getUserInfo(); if (userAndPassword == null) { userAndPassword = (conf.get(FS\_FTP\_USER\_PREFIX + host, null) + ":" + conf.get(FS\_FTP\_PASSWORD\_PREFIX + host, null)); } String[] userPasswdInfo = userAndPassword.split(":"); Preconditions.checkNotNull(userPasswdInfo[0].length > 1, "Invalid username / password"); conf.set(FS\_FTP\_USER\_PREFIX + host, userPasswdInfo[0]); conf.set(FS\_FTP\_PASSWORD\_PREFIX + host, userPasswdInfo[1]); setConf(conf); this.uri = uri; } /\* \* Connect to the FTP server using configuration parameters \*/ @Override protected void connect() throws IOException { FTPClient client = null; Configuration conf = getConf(); String host = conf.get(FS\_FTP\_HOST); int port = conf.getInt(FS\_FTP\_HOST\_PORT, FTP.DEFAULT\_PORT); String user = conf.get(FS\_FTP\_USER\_PREFIX + host); String password = conf.get(FS\_FTP\_PASSWORD\_PREFIX + host); client = new FTPClient(); client.connect(host, port); int reply = client.getReplyCode(); if (!FTPReply.isPositiveCompletion(reply)) { throw NetUtils.wrapException(host, port, NetUtils.UNKNOWN\_HOST, 0, new ConnectException("Server response " + reply)); } else if (client.login(user, password)) { client.setFileTransferMode(getTransferMode(conf)); client.setFileType(FTP.BINARY\_FILE\_TYPE); client.setBufferSize(DEFAULT\_BUFFER\_SIZE); setTimeout(client, conf); setDataConnectionMode(client, conf); } else { throw new IOException("Login failed on server - " + host + ", port - " + port + " as user " + user + ""); } return client; } can you fix the errors on the test case now.

---



With the provided context of the `FTPFileSystem` class, it looks like the test case needs to use the `Configuration` object to set and retrieve the FTP server's configuration details. Here's an updated test case that checks the FTP server configuration parameters using the `Configuration` object and asserts that they are set correctly.

The test case will be updated to use the `initialize` method, which seems to be the correct approach for setting up the FTP configuration. Here's the updated test case:

```
java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.ftp.FTPFileSystem;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.net.URI;

public class FTPFileSystemTest {

    private FTPFileSystem ftpFileSystem;
    private Configuration conf;
    private URI ftpUri;

    @Before
    public void setUp() {
        // Initialize configuration and FTP file system
        conf = new Configuration();
        ftpFileSystem = new FTPFileSystem();
        ftpUri = URI.create("ftp://user:password@localhost:21");
    }

    @Test
    public void testFtpConfigurationParameters() throws Exception {
        // Initialize FTPFileSystem with the given URI and configuration
        ftpFileSystem.initialize(ftpUri, conf);

        // Retrieve configuration details
        String host = conf.get(FTPFileSystem.FS_FTP_HOST);
        int port = conf.getInt(FTPFileSystem.FS_FTP_HOST_PORT, FTP.DEFAULT_PORT);
        String user = conf.get(FTPFileSystem.FS_FTP_USER_PREFIX + host);
        String password = conf.get(FTPFileSystem.FS_FTP_PASSWORD_PREFIX + host);

        // Assert configuration details are set correctly
        Assert.assertEquals("Host is not set correctly", "localhost", host);
        Assert.assertEquals("Port is not set correctly", 21, port);
        Assert.assertEquals("User is not set correctly", "user", user);
        Assert.assertEquals("Password is not set correctly", "password", password);
    }

    // Additional test methods to cover other functionalities of FTPFileSystem...
}
```

This test case sets up an `FTPFileSystem` and `Configuration` object, then initializes the `FTPFileSystem` with a `URI` that includes the user info and port. It then uses assertions to ensure the host, port, user, and password are set correctly in the `Configuration` object.

Make sure your test environment has access to the necessary Hadoop and FTP classes. If you face any further issues, please ensure that your classpath includes the relevant Hadoop and Apache Commons Net FTP client libraries.

---

 You:

[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.10.1:testCompile (default-testCompile) on project hadoop-common: Compilation failure [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[49,64] cannot find symbol [ERROR] symbol: variable FTP [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest

---

 ChatGPT:

The error message indicates that the `FTP` variable cannot be found within the `FTPFileSystemTest` class. The `FTP` variable is a constant from the Apache Commons Net library, which defines the default FTP port.

To resolve the compilation error, you should ensure that you have the correct import statement for the FTP class at the top of your `FTPFileSystemTest` class. Here's how you can import it:

```
java
import org.apache.commons.net.ftp.FTP;
```

Ensure this import statement is added to your test class. If this symbol is meant to be a part of your own classes (and not the Apache Commons Net FTP class), you will need to ensure that it is correctly defined and accessible within the scope of your test class.

With the correct import, the `DEFAULT_PORT` constant should be recognized. Here's the corrected portion of the test code that includes the import:

```
java
import org.apache.commons.net.ftp.FTP;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.ftp.FTPFileSystem;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.net.URI;

public class FTPFileSystemTest {
    // ... other parts of the test class remain unchanged ...

    @Test
    public void testFtpConfigurationParameters() throws Exception {
        // Initialize FTPFileSystem with the given URI and configuration
        ftpFileSystem.initialize(ftpUri, conf);

        // Retrieve configuration details
        String host = conf.get(FTPFileSystem.FS_FTP_HOST);
        int port = conf.getInt(FTPFileSystem.FS_FTP_HOST_PORT, FTP.DEFAULT_PORT); // Now FTP should be recognized
        String user = conf.get(FTPFileSystem.FS_FTP_USER_PREFIX + host);
        String password = conf.get(FTPFileSystem.FS_FTP_PASSWORD_PREFIX + host);

        // Assert configuration details are set correctly
        Assert.assertEquals("Host is not set correctly", "localhost", host);
        Assert.assertEquals("Port is not set correctly", 21, port);
        Assert.assertEquals("User is not set correctly", "user", user);
        Assert.assertEquals("Password is not set correctly", "password", password);
    }

    // ... additional test methods ...
}
```

Please ensure that your project's `pom.xml` file includes the dependency for the Apache Commons Net library if it is not already present. If the `FTP` class is part of your project and not the Apache Commons Net library, you will need to ensure that it is defined and available in the classpath.

---

 You:

[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.10.1:testCompile (default-testCompile) on project hadoop-common: Compilation failure [ERROR] /home/nvadde2/hadoop/hadoop-common-project/hadoop-common/src/test/java/org/apache/hadoop/fs/ftp/FTPFileSystemTest.java:[49,64] cannot find symbol [ERROR] symbol: variable FTP [ERROR] location: class org.apache.hadoop.fs.ftp.FTPFileSystemTest

---

 ChatGPT:

The error message you're encountering indicates that the `FTP` class or its member `DEFAULT_PORT` is not found in the context where it's being referenced in the `FTPFileSystemTest` class. This could happen due to several reasons:

1. The Apache Commons Net library is not included in the classpath.
2. The import statement for the `FTP` class is missing in the `FTPFileSystemTest.java` file.
3. The `FTP` class or the `DEFAULT_PORT` constant does not exist within the imported package.
4. There's a typographical error in the variable name or the import statement.

Given that the error points to a variable `FTP` and assuming that it refers to the `FTP` class from the Apache Commons Net library, here's what you need to verify:

1. Ensure that the Apache Commons Net library is declared as a dependency in your `pom.xml` file. It should look similar to this:

```
xml
<dependencies>
  <!-- other dependencies -->
  <dependency>
    <groupId>commons-net</groupId>
    <artifactId>commons-net</artifactId>
    <version>YOUR_VERSION_HERE</version>
  </dependency>
</dependencies>
```

Replace YOUR\_VERSION\_HERE with the version you're using. Make sure it's the correct and compatible version with your project.

- Check that you have the correct import statement at the top of the `FTPFileSystemTest` class:

```
java
import org.apache.commons.net.ftp.FTP;
```

- If you have done both steps above and the error persists, there might be an issue with your Maven setup. Try refreshing your project dependencies in your IDE or running `mvn clean install` from the command line to refresh the local dependencies.

If the `FTP` referred to here is actually a custom static variable that you have defined in another class, you need to make sure that:

- It's properly declared and initialized in that class.
- You have the correct import statement for that class in your `FTPFileSystemTest` class.
- The variable `FTP` is declared as public or package-private (with the test class being in the same package) and static.

Please confirm that these checks are correctly implemented, and if the error still persists, we might need to inspect other parts of your setup or codebase.



You:

use this: `@InterfaceAudience.Public @InterfaceStability.Stable public class FTPFileSystem extends FileSystem { public static final Logger LOG = LoggerFactory.getLogger(FTPFileSystem.class); public static final int DEFAULT_BUFFER_SIZE = 1024 * 1024; public static final int DEFAULT_BLOCK_SIZE = 4 * 1024; public static final long DEFAULT_TIMEOUT = 0; public static final String FS_FTP_USER_PREFIX = "fs.ftp.user."; public static final String FS_FTP_HOST = "fs.ftp.host"; public static final String FS_FTP_HOST_PORT = "fs.ftp.host.port"; public static final String FS_FTP_PASSWORD_PREFIX = "fs.ftp.password."; public static final String FS_FTP_DATA_CONNECTION_MODE = "fs.ftp.data.connection.mode"; public static final String FS_FTP_TRANSFER_MODE = "fs.ftp.transfer.mode"; public static final String E_SAME_DIRECTORY_ONLY = "only same directory renames are supported"; public static final String FS_FTP_TIMEOUT = "fs.ftp.timeout";}`



ChatGPT:

Based on the information provided, it seems like you're trying to compile a test case that uses the `FTPFileSystem` class. The error indicates that there's an issue with the `FTP.DEFAULT_PORT` not being found.

First, check to ensure that your imports in the `FTPFileSystemTest` class include the `FTP` class from the Apache Commons Net library where the `DEFAULT_PORT` is supposed to be defined:

```
java
import org.apache.commons.net.ftp.FTP;
```

If you have this import and the error persists, it could be due to one of the following reasons:

- The version of the Apache Commons Net library you are using does not have a `DEFAULT_PORT` constant in the `FTP` class. You should confirm by checking the library's API documentation for the version you are using.
- There is an issue with the classpath in your Maven setup, and the library is not being properly included during compilation.

Assuming that the `FTPFileSystem` class is correctly using `FTP.DEFAULT_PORT` from the Apache Commons Net library, you should verify that the dependency is correctly set up in your `pom.xml`. For example:

```
xml
<dependencies>
  <!-- Other dependencies -->

  <dependency>
    <groupId>commons-net</groupId>
    <artifactId>commons-net</artifactId>
    <version>3.8.0</version> <!-- Make sure you use the correct version -->
  </dependency>
</dependencies>
```

Make sure to replace 3.8.0 with the version that's compatible with your project and that the version you're using contains the `FTP` class with a `DEFAULT_PORT` field.

Once you've verified that the dependency is correct and the import statement is in place, try to rebuild your project. If you are using an IDE, make sure to refresh the Maven project to update the dependencies. If you're using the command line, you can run:

```
sh
mvn clean install
```

This will clean your target directory, re-download dependencies, and attempt to rebuild your project.

If you've done all of this and the error still shows up, please make sure that there isn't a typo or mismatch in the class name or variable reference in your test class.



You:

no rewrite the test so as not to modify the pom



ChatGPT:

If you need a test case that uses the `FTPFileSystem` class without altering the `pom.xml`, you would have to rely only on the classes and methods already available in the existing project dependencies.

Since you're getting a "cannot find symbol" error for `FTP.DEFAULT_PORT`, it suggests that either the `FTP` class is not imported or the `DEFAULT_PORT` field is not available in the version of the Apache Commons Net library that you are using. Instead of referring to `FTP.DEFAULT_PORT`, you can hardcode the default FTP port (which is typically 21) directly into your test class.

Below is an example of how you might rewrite the test to avoid using `FTP.DEFAULT_PORT`:

```
java
package org.apache.hadoop.fs.ftp;

import org.apache.hadoop.conf.Configuration;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
import java.net.URI;

public class FTPFileSystemTest {

    private static final int FTP_DEFAULT_PORT = 21; // The default FTP port is 21
    private Configuration conf;
    private FTPFileSystem fileSystem;

    @Before
    public void setUp() throws IOException {
        conf = new Configuration();
        fileSystem = new FTPFileSystem();
        // Assume that the host and credentials are set in the configuration
        URI uri = URI.create("ftp://user:password@localhost/test");
        fileSystem.initialize(uri, conf);
    }

    @Test
    public void testFtpDefaultPort() {
        // Validate that the file system uses the default port when none is specified
        Assert.assertEquals(FTP_DEFAULT_PORT, fileSystem.getUri().getPort());
    }

    // Add additional test methods here

    // Remember to include any other necessary test setup or teardown methods.
}
```

This code assumes that you have appropriate host and credentials specified in your `Configuration` object. You'll need to replace `localhost`, `user`, and `password` with actual values for your FTP server.

In your actual test methods, instead of checking against `FTP.DEFAULT_PORT`, you would check against the `FTP_DEFAULT_PORT` constant defined in your test class.

Ensure that this test is in the correct directory corresponding to its package structure, typically something like `src/test/java/org/apache/hadoop/fs/ftp/` for a Maven project.

Please note that using hardcoded values in place of constants is not a best practice and should be avoided if possible. This approach is only a workaround given the constraint of not modifying the `pom.xml`.

---