`

tests/test\_dependency\_manager.py

import unittest

import os

from unittest.mock import patch

from utils.dependency\_manager import ensure\_dependencies

class TestDependencyManager(unittest.TestCase):

@patch('utils.dependency\_manager.check\_command\_exists')

@patch('utils.dependency\_manager.run\_subprocess')

def test\_ensure\_dependencies\_all\_present(self, mock\_run\_subprocess, mock\_check\_command\_exists):

# Simulate all dependencies being present

mock\_check\_command\_exists.return\_value = True

config = {}

ensure\_dependencies(config)

mock\_run\_subprocess.assert\_not\_called()

@patch('utils.dependency\_manager.check\_command\_exists')

@patch('utils.dependency\_manager.run\_subprocess')

def test\_ensure\_dependencies\_some\_missing(self, mock\_run\_subprocess, mock\_check\_command\_exists):

# Simulate some dependencies being missing

def side\_effect(cmd):

# Return True if the command is 'git' or 'docker', False otherwise

return cmd in ['git', 'docker']

mock\_check\_command\_exists.side\_effect = side\_effect

config = {}

ensure\_dependencies(config)

mock\_run\_subprocess.assert\_called()

```

In this corrected code, the `side\_effect` method returns a boolean value indicating whether the command exists. If the command is 'git' or 'docker', it returns `True`; otherwise, it returns `False`. This allows the test to function correctly and verify that the `ensure\_dependencies` method behaves as expected.