《智能合约技术》

实验报告

|  |  |
| --- | --- |
| 项目名称 | JS测试脚本自动测试Solidity智能合约 |
| 班 级 |  |
| 学生姓名 |  |
| 学生学号 |  |
| 指导教师 | 汪凌锋 |
| 成 绩 |  |

2024年 11月18 日 成都信息工程大学 区块链产业学院

# 实验基本信息

**被测试学生姓名**：

**被测试学生学号**：

# 实验目的

搭建Solidity智能合约JS测试环境、与结对同学相互测试第2章和第3章提交的智能合约代码。

# 实验环境

Windows10操作系统、智能合约集成开发环境Remix、项目管理平台Gitee、代码仓库管理工具Git/TortoiseGit、服务器端JS运行环境NodeJS、Hardhat Solidity开发测试环境。

# 需要提交的代码文件

tests/202x131xxxx/HelloWorldTest.js、tests/202x131xxx/HelloMyWorldTest.js

“202x131xxx”替换为自己的学号

# 实验八：同步源仓库代码到分叉仓库

## 实验步骤：

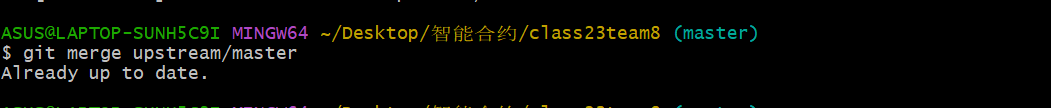
1. 小组长在“Git Bash”中本地小组分叉仓库路径下运行以下命令获取上游upstream仓库最新代码，保存命令运行输出结果截图。

git fetch upstream



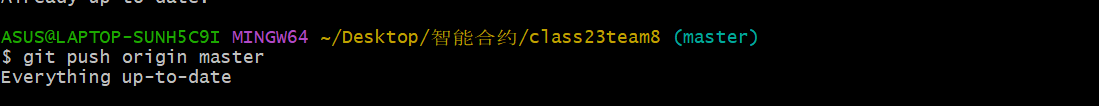
1. 小组长在“Git Bash”中运行以下命令合并upstream/master分支的最新代码到本地当前分支（master分支），保存命令运行输出结果截图（组员保存组长的截图）。

git merge upstream/master



1. 小组长在“Git Bash”中运行以下命令将本地分叉仓库master分支最新代码推送到远程分叉仓库master分支，保存命令运行输出结果截图（组员保存组长的截图）。

git push origin master

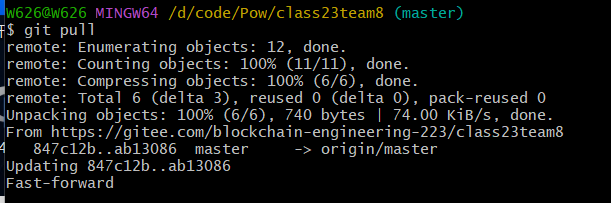


1. 组员在“Git Bash”中运行以下命令将远程分叉仓库master分支最新代码拉取到本地分叉仓库master分支，保存命令运行输出结果截图。

*git pull*

或

*git pull --progress -v --no-rebase "origin"*



## 实验结果：

**完成了什么实验？**

本次实验的目的是同步源仓库（upstream）的最新代码到小组的分叉仓库（fork）。实验步骤包括：从上游仓库获取最新代码、将这些代码合并到本地的 master 分支、推送到远程分叉仓库，并将远程分叉仓库的代码同步到组员的本地仓库。

**实验结论是什么？**

通过 git fetch 获取上游仓库的最新代码，并通过 git merge 合并到本地分支，再通过 git push 推送到远程分叉仓库，最终通过 git pull 同步远程分叉仓库的代码到本地，成功完成了代码同步操作。

**遇到了什么困难，怎么解决的？**

在合并上游仓库代码时，可能会遇到冲突。遇到冲突时，需要根据提示手动解决冲突，并提交解决后的代码。此外，还可能会遇到网络连接问题，解决方法是检查网络配置或更换镜像源。

**对实验有何认识？**

通过此次实验，深刻理解了如何在分叉仓库与上游仓库之间同步代码，掌握了 git fetch、git merge 和 git push 等常用 Git 命令的使用。实验也加深了对 Git 版本控制流程和团队协作开发中的代码同步操作的认识。

# 实验九：使用Hardhat测试智能合约

## 实验步骤：

1. 使用文件浏览器在电脑D盘Code路径下，以TortoiseGit图形界面方式全新克隆小组远程分叉仓库到本地，小组本地分叉仓库文件夹指定为"scptest"，并列出小组远程分叉仓库链接、小组本地分叉仓库地址、保存本地小组分叉仓库全路径截图。

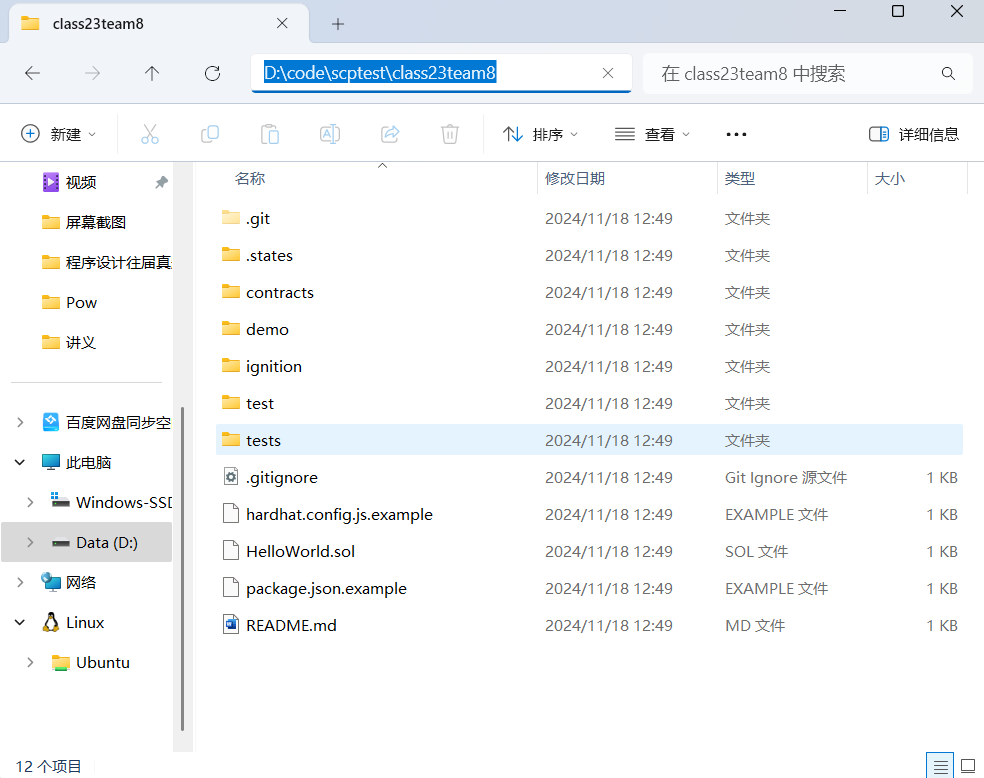
小组远程分叉仓库链接是：

https://gitee.com/blockchain-engineering-223/class23team8.git

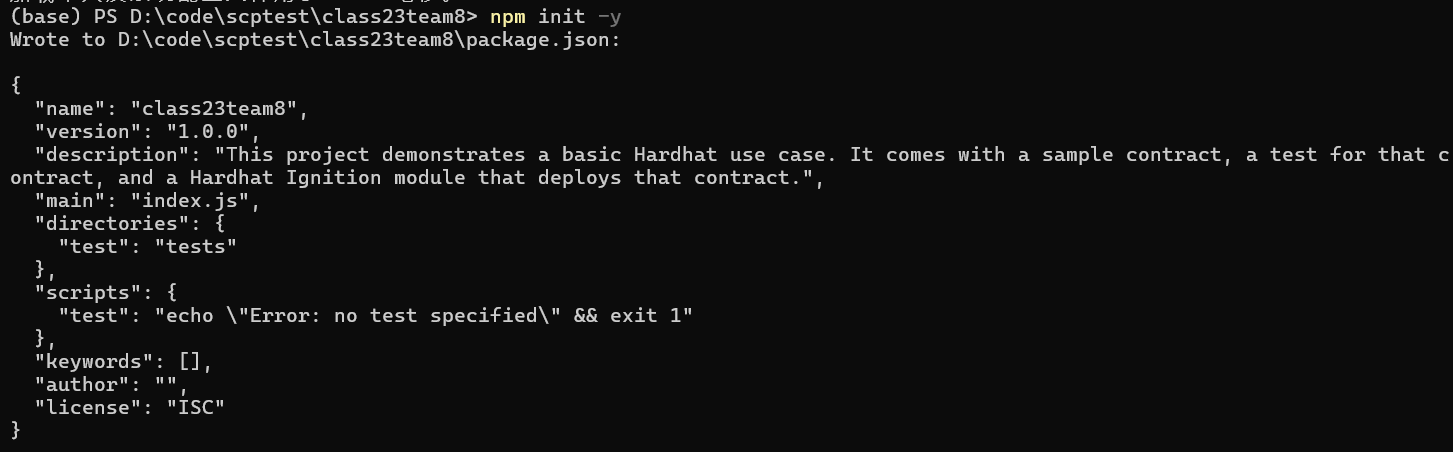
小组本地分叉仓库地址是：

D:\code\scptest\class23team8

本地分叉仓库完整路径截图：



1. 在小组本地分叉仓库的根路径下，使用Windows命令行工具运行命令"npm init -y" 初始化NPM工程，保存初始化成功的截图



1. 将小组本地分叉仓库根路径下package.json文件"directories"下的字段"test"的值改为" tests/2020131000"，其中"2020131000"替换为自己的学号。
2. 修改package.json，添加"devDependencies"字段，"devDependencies"字段添加以下灰色字体的依赖关系，并保存package.json文件修改"test"字段和添加"devDependencies"字段后完整内容的截图。

*"devDependencies": {*

*"@nomiclabs/hardhat-ethers": "^2.0.2",*

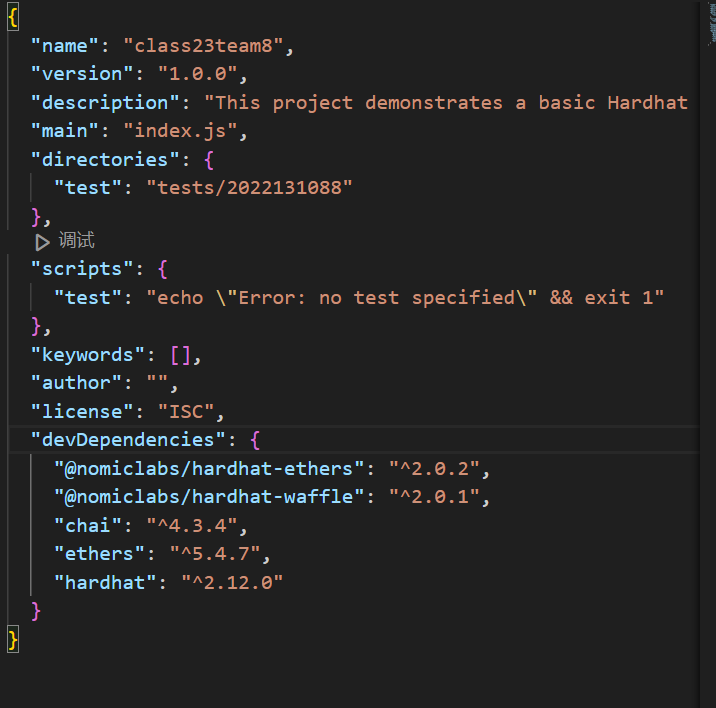
*"@nomiclabs/hardhat-waffle": "^2.0.1",*

*"chai": "^4.3.4",*

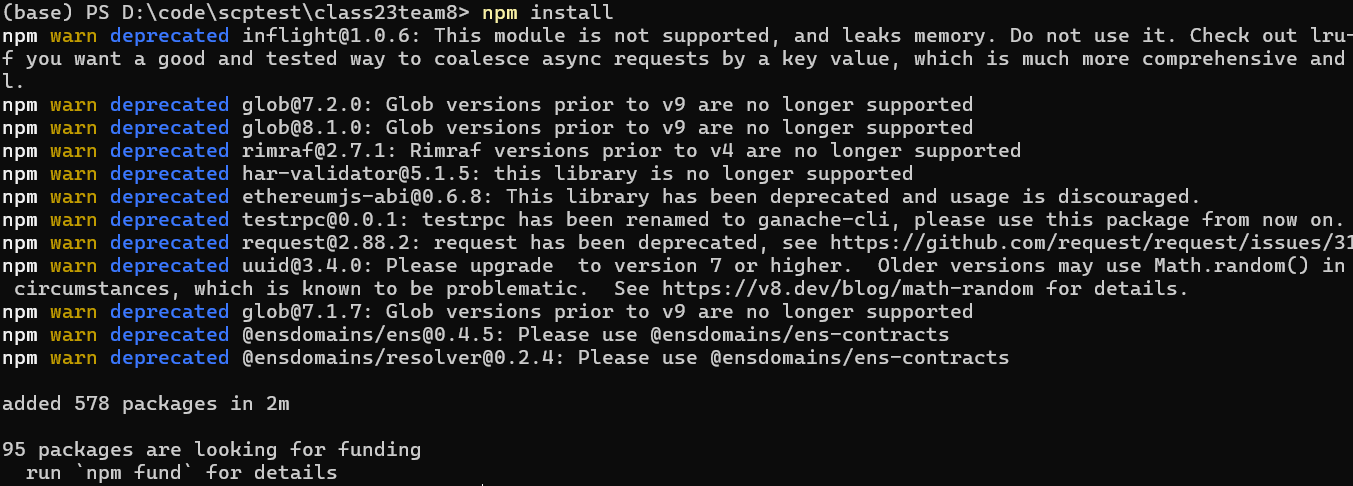
*"ethers": "^5.4.7",*

*"hardhat": "^2.12.0"*

*}*



1. 在本地分叉仓库的根路径下，使用Windows命令行方式运行命令”npm install”安装第4步增加的依赖关系包，保存安装成功的截图。



1. 在本地分叉仓库根路径下创建hardhat配置文件hardhat.config.js，在文件中输入以下灰色字体内容并保存，”sources”字段内容中的学号改为被测试学生的学号，”tests”字段内容中的学号改为自己的学号。

*/\*\**

*\* @type import('hardhat/config').HardhatUserConfig*

*\*/*

*require("@nomiclabs/hardhat-waffle");*

*module.exports = {*

*solidity: "0.8.16",*

*defaultNetwork: "hardhat",*

*networks: {*

*hardhat: {*

*}*

*},*

*paths: {*

*sources: "./contracts/2020131000",*

*tests: "./tests/2020131000",*

*cache: "./cache",*

*artifacts: "./artifacts"*

*},*

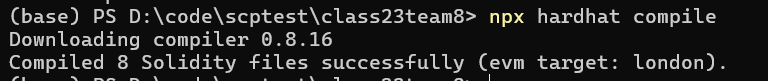
*};*

1. 在小组本地分叉仓库的根路径下，使用Windows命令行方式运行命令”npx hardhat compile”编译智能合约。

若编译失败，保存编译失败的截图：

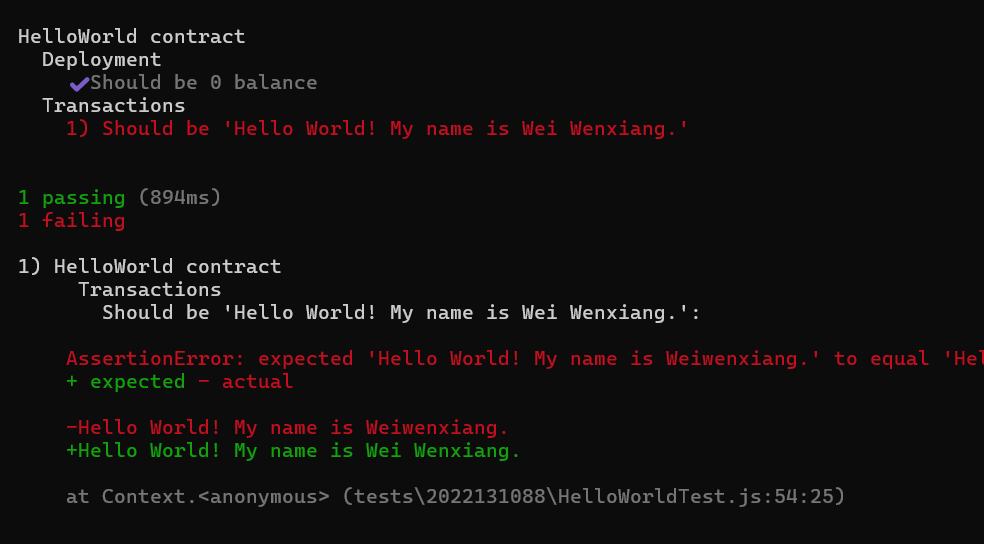
解决编译错误后，保存编译成功的截图：

解决编译告警后，保存编译成功且没有Warning的截图：

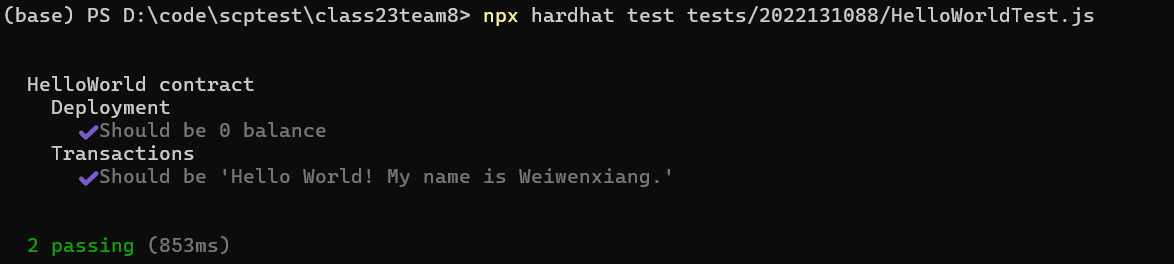


1. 在小组本地分叉仓库中创建以自己学号命名的文件夹（例：”tests/2020131000”， “2020131000”替换为自己的学号），将测试脚本文件”tests/2020131000/HelloWorldTest.js”拷贝到自己学号命名的新文件夹下，测试脚本文件第2个测试用例字符串变量expectedString值中的名字拼音改为被测试学生的名字拼音（例：”Hello World! My name is Zhang San.”）。
2. 在本地分叉仓库的根路径下，使用Windows命令行方式运行命令”npx hardhat test tests/2020131000/HelloWorldTest.js”执行自己的智能合约测试用例，其中”2020131000”替换为自己的学号。

若测试用例执行失败，保存测试失败的截图：



解决问题后，保存测试成功的截图：

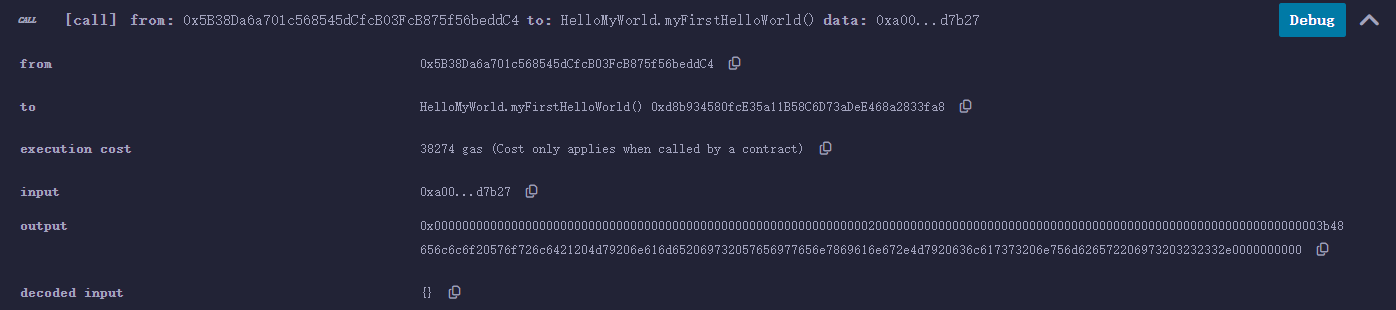


1. 在Remix IDE中部署被测试学生的HelloMyWorld合约，调用HelloMyWorld. myFirstHelloWorld函数，查看调用该函数的交易实际消耗燃料数。

在金山文档中记录实际消耗的燃料数：

38274

保存查看实际消耗燃料数的截图：



1. 按照HelloWorldTest.js测试脚本的模式，在自己测试学号文件夹下创建HelloMyWorldTest.js测试脚本，在测试场景”Transactions”中，编写被测试学生HelloMyWorld合约的测试用例，执行个人测试文件夹下两个测试脚本，并保存测试用例执行成功后的截图。

## 实验结果：

**完成了什么实验？**

（描述是否按照实验步骤成功编写并运行测试脚本，被测试学生提交的第2章、第3章的代码是否未做修改一次性通过测试，被测试代码修改后最终是否通过测试，调用被测试代码HelloMyWorld.myFirstHelloWorld的交易实际消耗的燃料值是多少）

本次实验的目的是使用 Hardhat 测试智能合约。实验步骤包括：从远程分叉仓库克隆代码、初始化 NPM 项目、修改 package.json 文件以添加依赖关系、安装依赖包、编译智能合约、创建并修改测试脚本、执行测试用例以及在 Remix IDE 中部署合约并查看交易的燃料消耗。最终成功执行了测试用例，并确认被测试学生的第2章、第3章的代码未做修改一次性通过测试。在测试过程中，成功调用了 HelloMyWorld.myFirstHelloWorld 函数，交易实际消耗的燃料值为 38274。

**实验结论是什么？**

通过该实验，掌握了使用 Hardhat 进行智能合约开发和测试的流程。成功编写并执行了智能合约的测试脚本，确保了合约功能的正确性。通过 Remix 部署并调用合约函数，确认了合约的燃料消耗情况。

**遇到了什么困难，怎么解决的？**

在测试脚本执行过程中，遇到了一些依赖包安装错误。通过检查 package.json 中的依赖版本以及重新运行 npm install 命令解决了问题。编译合约时也曾遇到小的语法错误，解决方法是仔细检查合约代码并根据错误提示进行修正。

**对实验有何认识？**

通过本次实验，了解了如何使用 Hardhat 和 Remix 进行智能合约的开发、测试和部署，深刻认识到编写和测试智能合约的复杂性。掌握了在团队协作中，如何通过 Git 来同步和管理项目代码，确保各成员的代码正确性。