**实验4 以太坊单节点搭建及合约部署**

**【实验介绍】**

本次实验首先在Ubuntu20.04系统上安装Go语言、相关依赖包和工具以搭建以太坊开发环境；之后编写并部署两个简单的智能合约。本实验旨在帮助读者对以太坊与智能合约有初步的理解，并通过部署两个智能合约了解以太坊对交易的处理流程。

在开始具体的实验之前，首先了解以太坊和智能合约的相关背景知识是十分必要的。

（1）以太坊及依赖环境

以太坊被称为区块链2.0，是一个可编写智能合约的开源区块链平台。除了可基于内置的以太币（ether）实现数字货币交易外，以太坊还提供了图灵完备的编程语言以编写智能合约，从而将智能合约应用到了区块链当中。诚然，在比特币中亦可使用基于栈的脚本语言来编写智能合约。但是这种脚本语言没有循环和递归结构，同时它比较难以理解和使用。以太坊通过引入solidity编程语言解决了这一问题。

本次实验中涉及的依赖环境主要包括：

①Node.js：它是一个跨平台的 JavaScript 运行环境，其构建在为了在服务器端运行JavaScript 代码而设计的 Chrome JavaScript 上，并通常被用来构建后端应用，同时它也是非常流行的全栈和前端解决方案；

②npm：它是 Node.js 的默认包管理工具，也是世界上最大的软件仓库；

③truffle：它是基于Javascript针对solidity编程语言的开发框架。它提供了许多优点，如可直接使用其合约抽象接口来操作合约函数，在控制台调用输出结果等；

④solc：它是solidity编程语言的编译工具，其将编写好的智能合约编译为EVM内的代码；

⑤Geth：它是使用Golang编写的以太坊官方提供的客户端，同时它也是目前最受欢迎的以太坊客户端。

在初步学习了以太坊的基本概念后，进一步了解以太坊对交易的处理流程对后续的实验是大有裨益的。具体来讲，该流程主要可以概括为如下的7个阶段，它们分别是：

①连接的建立：当以太坊节点启动后，它们通过P2P框架相互连接；

②区块与交易的同步：当某以太坊节点成功与其他节点建立连接后，它会与其余节点进行区块的同步。同时当有节点收到来自客户端的交易后，它会与其余节点进行交易的同步；

③交易的接收：以太坊节点接收来自客户端的交易，并对其进行校验后加入交易池中；

④交易的打包：以太坊节点根据不同的算法逻辑将一定数量的交易打包进区块，如PoW算法中为最长链节点打包交易；

⑤交易的执行：交易执行是由以太坊虚拟机完成的，在执行完毕后，会将诸如gasUsed等字段填入到区块中；

⑥区块的共识：以太坊采用PoW算法进行共识；

⑦区块的导入：以太坊将共识成功的区块落盘。

（2）智能合约

智能合约即运行在区块链平台上，实现了某种验证或执行功能的程序。Solidity是用来在以太坊上编写智能合约最受欢迎的编程语言。正如之前介绍的，solc是Solidity语言的编译工具，它可以将合约文件编译为.abi文件与.bin文件。前者描述了合约的接口，后者是一个由以太坊虚拟机指令集合构成的二进制文件。而合约的部署与调用会在之后的实验流程中展开详细的介绍。

**【实验要求】**

（1）熟悉以太坊依赖环境及相关库

（2）掌握合约编写、编译和部署的方法与流程

**【实验准备】**

**以太坊环境搭建**

(1) 安装go语言

下载二进制包并解压到指定位置：（如果卡在

，

图4-1 安装Golang可能出现的问题

可以尝试按回车，重新输入密码后系统会再次下载，下载成功后命令行显示：

，

图4-2 完成二进制包下载的命令行输出

同时在路径/usr/local中可以看到go文件夹）

|  |
| --- |
| **wget** -c https**://**golang.google.cn**/**dl**/**go1.15.7.linux-amd64.tar.gz -O **-** **|** **sudo** tar **-**xz -C **/**usr**/**local |

gedit ~/.bashrc在文件最后添加环境变量：

|  |
| --- |
| **export** PATH**=$PATH:/**usr**/**local**/**go**/**bin |

保存文件，重新加载bashrc配置文件：

|  |
| --- |
| **source** **~/.**bashrc |

查看go版本信息，测试安装是否完成：

|  |
| --- |
| go version |

(2) 安装Node.j和npm

若下载慢，可打开软件Software & Update切换apt国内镜像源下载：如果用下面的：**sudo** apt install npm **–**y出现如下错误：

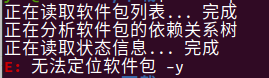


图4-3 安装npm可能出现的问题

删除目前安装的软件包：sudo apt-get remove nodejs npm

重新尝试安装：sudo apt install nodejs npm

|  |
| --- |
| **sudo** apt update  **sudo** apt install nodejs **-**y  **sudo** apt install npm **–**y  npm config set registry https**://**registry.npm.taobao.org # npm换淘宝源  npm get registry //查看npm源 |

查看版本信息：

|  |
| --- |
| node **-**v  npm **-**v |

(3) 安装truffle

|  |
| --- |
| **sudo** npm install -g truffle //出现warn不影响 |

(4) 安装solidity编译工具solc

安装并查看版本信息：

|  |
| --- |
| **sudo** add-apt-repository **-**y ppa**:**ethereum**/**ethereum  **sudo** apt update  **sudo** apt install solc **-**y  solc **--**version |

(5) 安装vscode编辑器

输入如下命令，使用apt安装vscode：

|  |
| --- |
| **sudo** apt update  **sudo** apt install **-**y software-properties-common apt-transport-https  **wget** **-**q https**://**packages.microsoft.com**/**keys**/**microsoft.asc **-**O- **|** **sudo** apt-key add **-**  **sudo** add-apt-repository "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main"  **sudo** apt install code |

(6) 安装以太坊客户端Geth

添加PPA源，安装go-ethereum稳定版（前面若执行前两步这里则无需执行）：

|  |
| --- |
| **sudo** add-apt-repository **-**y ppa**:**ethereum**/**ethereum  **sudo** apt update  **sudo** apt install ethereum **-**y |

查看版本信息：

|  |
| --- |
| geth version |

**【实验过程】**

**合约部署**

（1）启动Geth环境

新建一个eth00目录（两位数字为学号后两位）并在该文件夹下的命令行里输入：

|  |
| --- |
| mkdir eth00  cd eth00  geth **--**datadir testNet **--**dev console |

--datadir后面的参数是区块数据及密钥存放目录；--dev启用开发者网络（模式），使用POA共识，默认预分配一个开发者账户并自动开启挖矿；console启用一个交互式的Javascript环境，日志信息也会显示在终端中。

如下图所示，创建了一个以太坊测试网络：

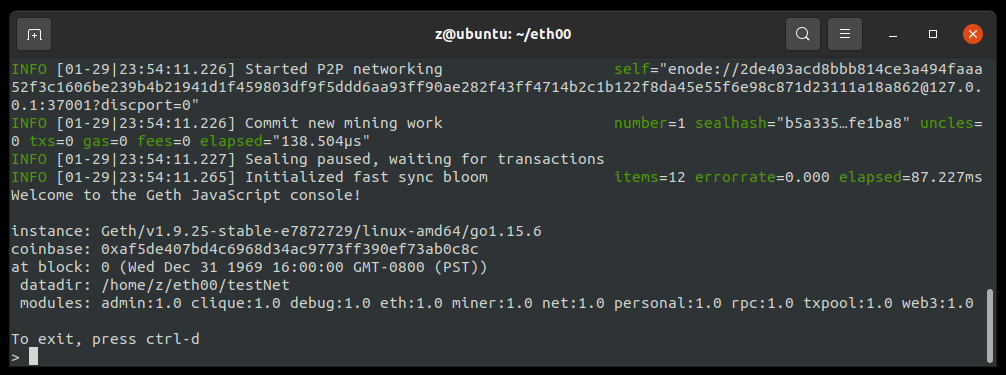


图4-4 成功创建以太坊测试网

输入eth，如下图可查看到当前以太坊的相关信息（格式为JSON），当前仅有创世区块0号块，只有一个账户，coinbase为矿工账户，即挖矿的奖励会进入这个账户，当前挖矿已开启，只要有交易就会打包成块。



图4-5 以太坊的当前相关信息

使用eth.accounts或personal.listAccounts查看账户列表，使用eth.getBalance(eth.accounts[0])查看账户余额，如下图所示：

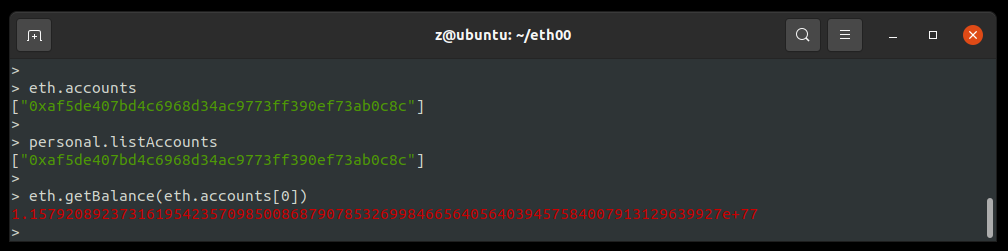


图4-6 成功获取以太坊账户列表及账户余额

（2）创建账户

开发模式下以太坊默认账户拥有大量的余额，如果用该账户来部署合约将无法看到余额变化，故创建一个新的账户进行下面的实验（**注意，图中的地址需为实际实验环境下的，后面也是如此，不多赘述**），“123”是新账户的密码，实验时可自行设定，可看到新账户的余额为0。



图4-7 成功创建新的以太坊账户

（3）转账

新账户余额为0，无法部署合约（需消耗以太币），故使用转账接口eth.sendTransaction()从默认账户转账1以太币到新用户，由于这是一次交易，可在下方日志信息中看到挖矿记录，其交易哈希为：0x55920603010638444af8f10bb3081ee99e89b6672c92e2e823debd7226e6a629

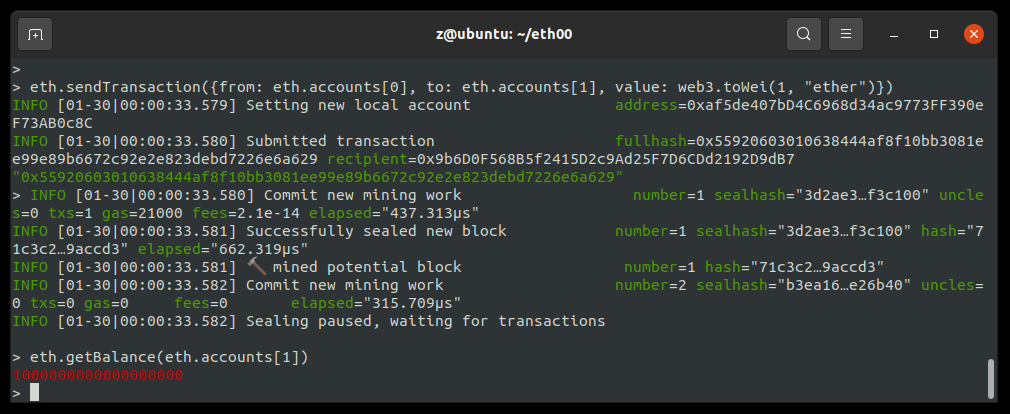


图4-8 日志信息中的挖矿记录

利用eth.getTransaction()接口可查看该交易信息，如下图所示，在返回的json数据中，我们能看到交易所属的块号及其哈希值，交易双方（from和to），交易值（value），消耗的gas值，v、r、s为交易签名相关字段。

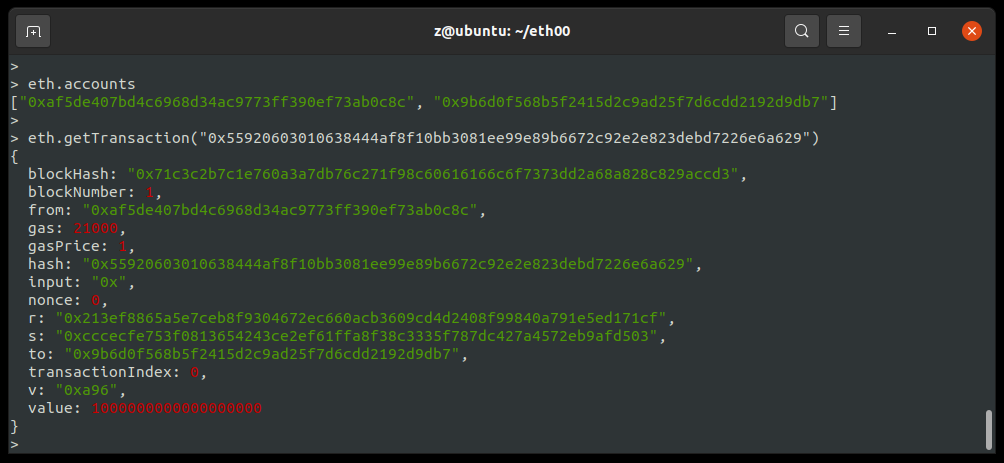
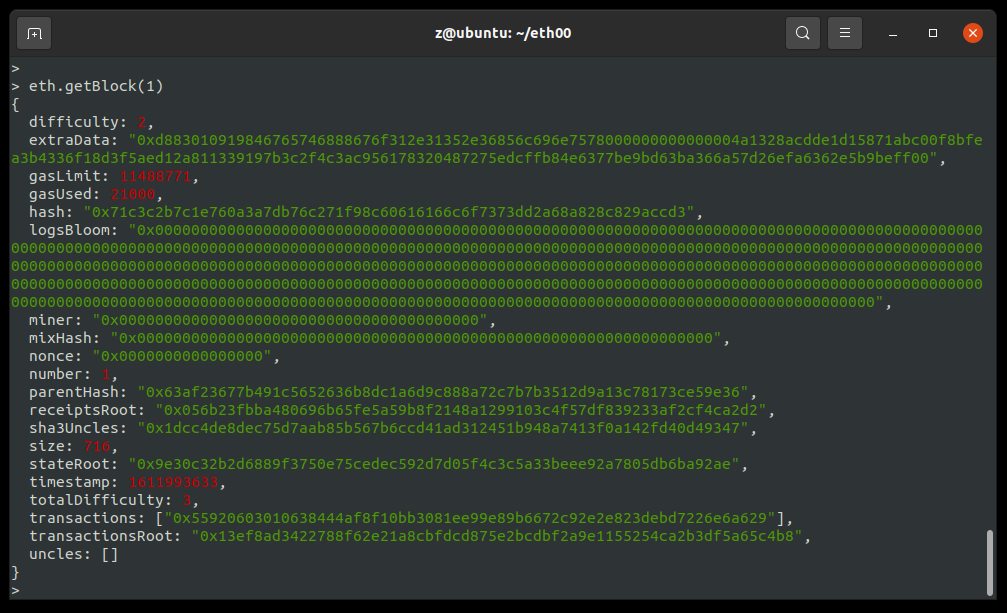


图4-9 某笔交易的具体信息

通过eth.getBlcok()接口可查看区块信息，如下图所示，其包含上文交易的哈希值。

图4-10 某个区块的具体信息

（4）智能合约的编写

接下来编写第一个智能合约。

新开一个终端，输入如下命令使用vscode打开项目文件夹：

|  |
| --- |
| cd **~/**eth00**/**  code **.** |

安装solidity扩展：

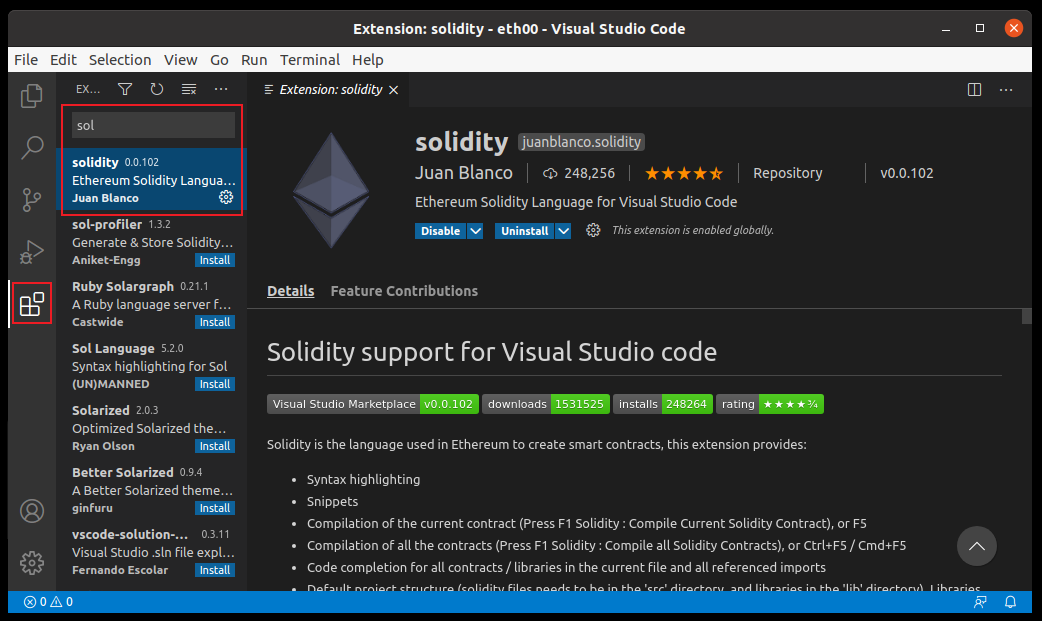
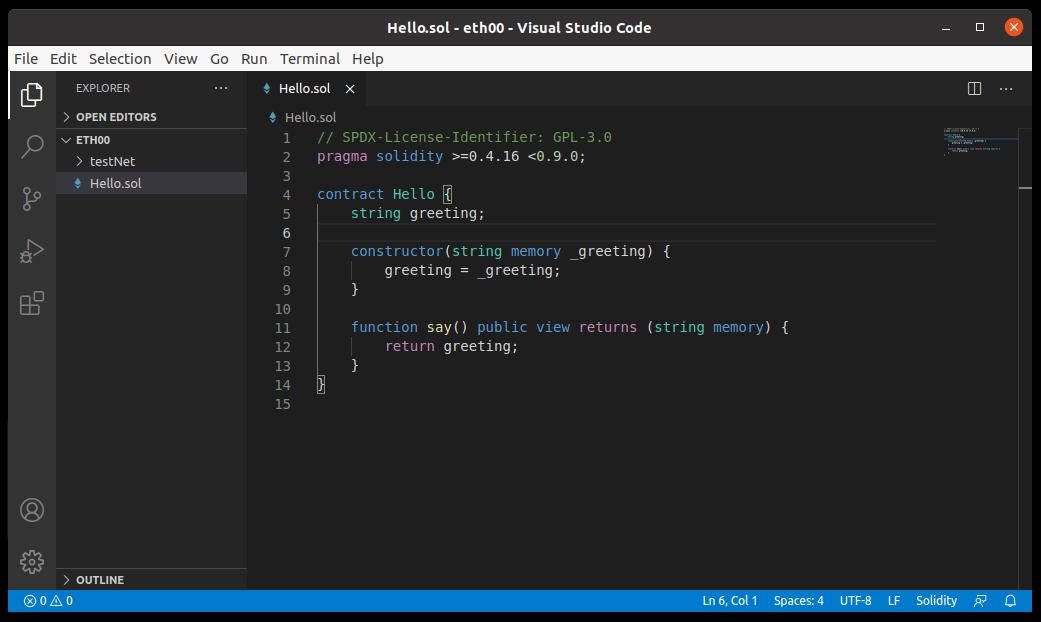


图4-11 Vscode中的solidity扩展

创建Hello.sol，编写solidity代码如下：

图4-12 第一个智能合约的代码

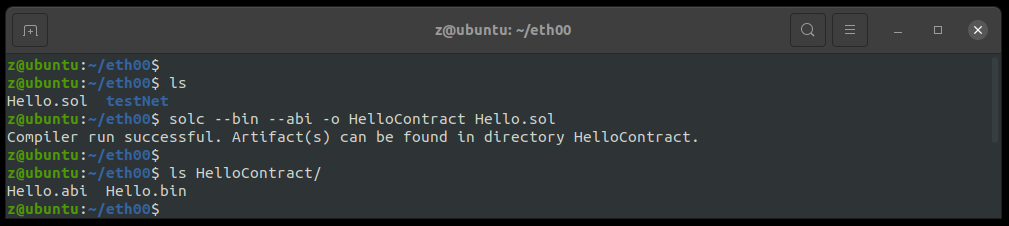
（5）编译智能合约

使用前文安装的solc进行编译：

|  |
| --- |
| solc **--**bin **--**abi -o HelloContract Hello.sol |

--bin要求输出字节码，即运行在以太坊虚拟机上的代码，类似于Java虚拟机上的字节码；--abi要求输出应用程序二进制接口（Application Binary Interface），即以太坊调用合约时的接口说明；-o HelloContract为输出文件目录。

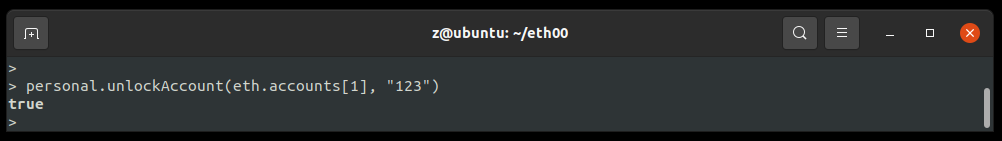
编译后，HelloContract目录下有Hello.abi文件和Hello.bin文件。

图4-13 智能合约编译后产生的文件

（6）解锁账户

部署合约前需先解锁账户（如同银行转账需输入密码），“123”为前面创建账户设置的密码，输入下面命令后得到“true”即为解锁成功。

|  |
| --- |
| personal.unlockAccount(eth.accounts[1], "123") |

图4-14 成功解锁以太坊账户

（7）部署智能合约

如下图所示，将文件里的字节码和abi粘贴到geth环境中，分别创建HelloAbi和HelloBin变量（**注意**HelloBin**需加双引号和0x**）。

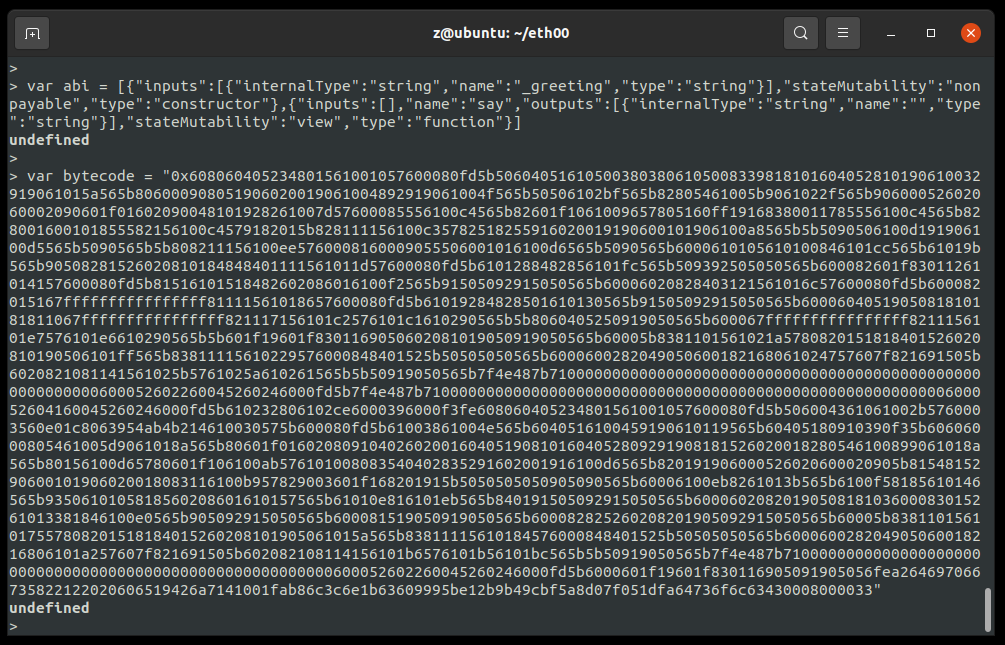


图4-15 第一个智能合约的部署过程

如下图所示，使用eth.contract()创建一个合约实例HelloContract，然后在用HelloContract创建一个HelloContract实例，参数为其构造函数参数和交易参数，此时已部署一个新的合约。部署本身也是一笔交易，发送方为eth.accounts[1]），发送数据为合约字节码，消耗一定的gas，故下图日志信息中矿工挖出一个块，块中交易为此次合约部署。**注意等待该交易上链再进行下面的操作，因为只能在合约实例获得地址后才可以调用方法。**

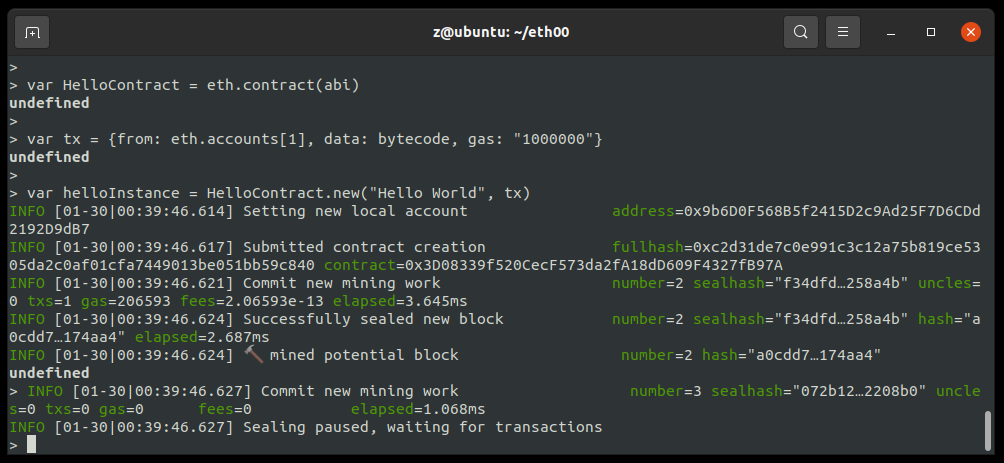


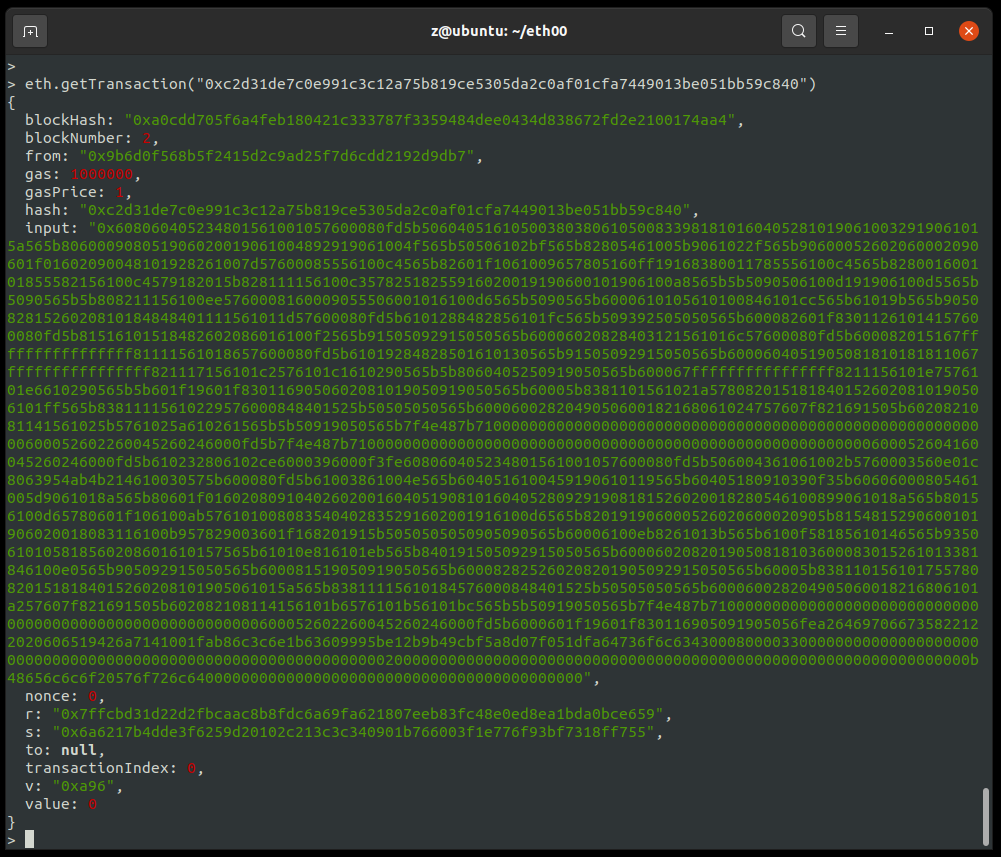
图4-16 创建智能合约(HelloContract)的实例

如下图所示，打印这个实例，可看到合约abi、合约地址、交易哈希和相关的接口方法，前文构造实例时我们传入了参数“Hello World”，这里我们调用say()方法打印该合约的存储信息，**注意此处使用** helloContract.say.call()**来打印信息，使用**helloContract.say()**只会产生交易。**



图4-17 智能合约(HelloContract)实例的有关信息

使用eth.getTransaction()查看该部署交易信息：

图4-18 部署交易的信息

（8）部署第二个简单智能合约

编写智能合约代码如下：

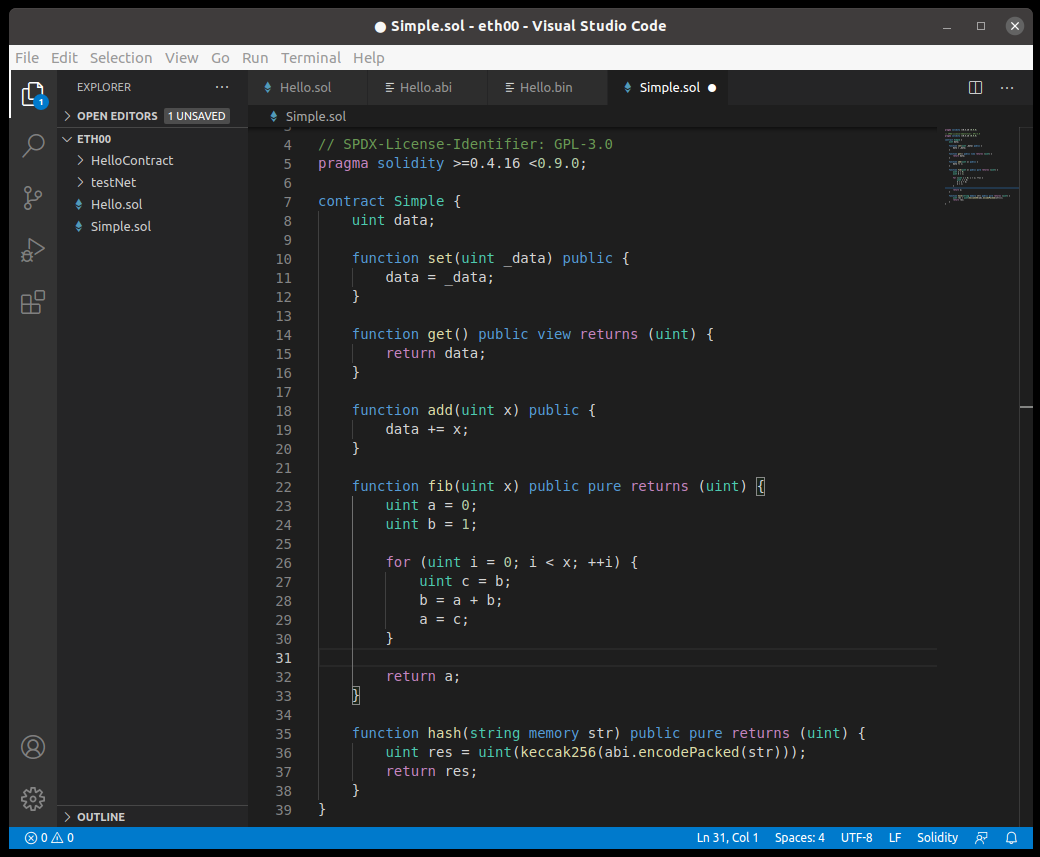


图4-19 第二个智能合约的代码

编译智能合约：

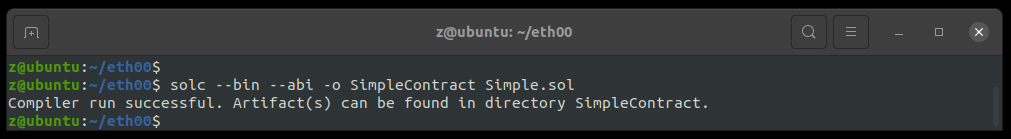


图4-20 智能合约的编译

在geth中创建相应变量，将abi和字节码复制进去：

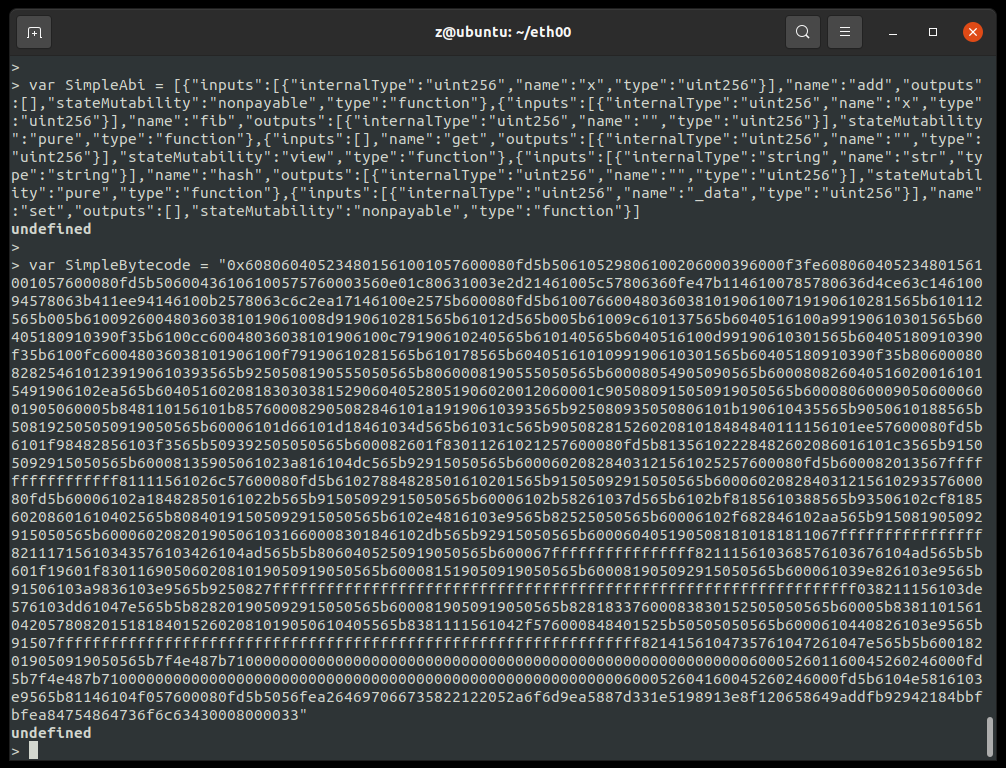
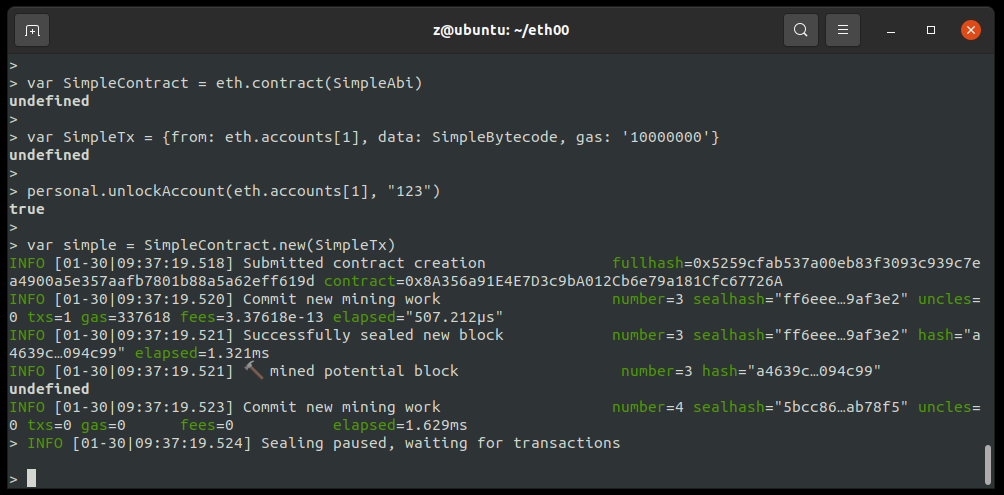


图4-21 第一个智能合约的部署过程

创建合约实例SimpleContract和SimpleContract实例simple:

图4-22 创建智能合约(SimpleContract)的实例

打印simple，如果address为undefined且下面无方法，可将上面的实例命令再执行一遍，重新创建一个simple实例。如果合约上链可以看到最后面的接口方法：

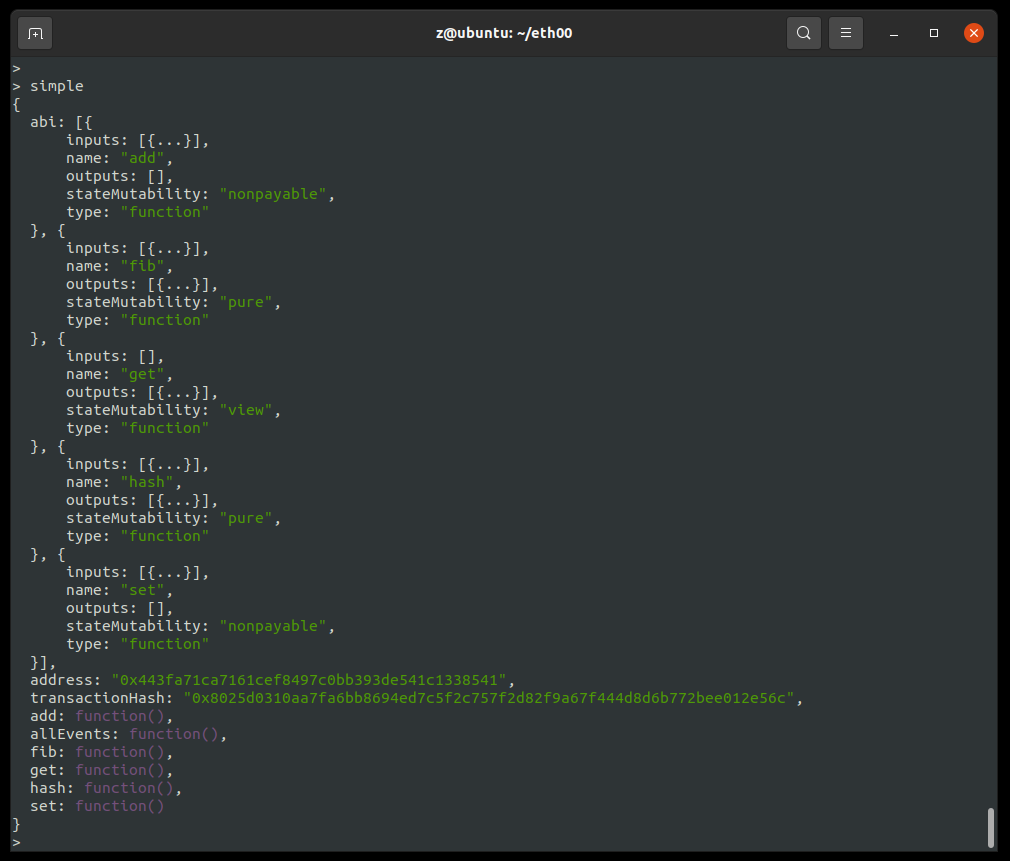


图4-23 智能合约(SimpleContract)实例的有关信息

执行simple.set()方法，需设置默认账户（执行合约的账户）或解锁账户（解锁一段时间后会重新锁定，需再次解锁）:

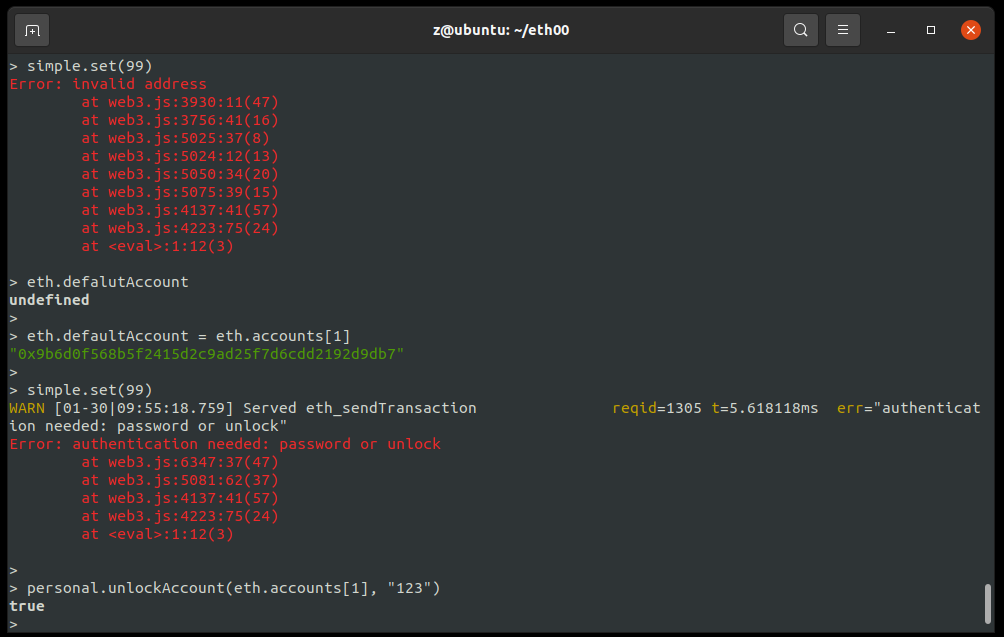


图4-24 账号的重新锁定

重新执行，结果如下图所示，该方法产生一笔交易，并被矿工放入块中成功上链，通过其哈希可查看相应信息，注意其中的input值，前面的0x60fe47b1是set的函数签名，后面的0x63即为set的参数值。这里我们可以知道区块链像日志一样将相应的操作都保存在链上，合约中的数据皆可溯源之。

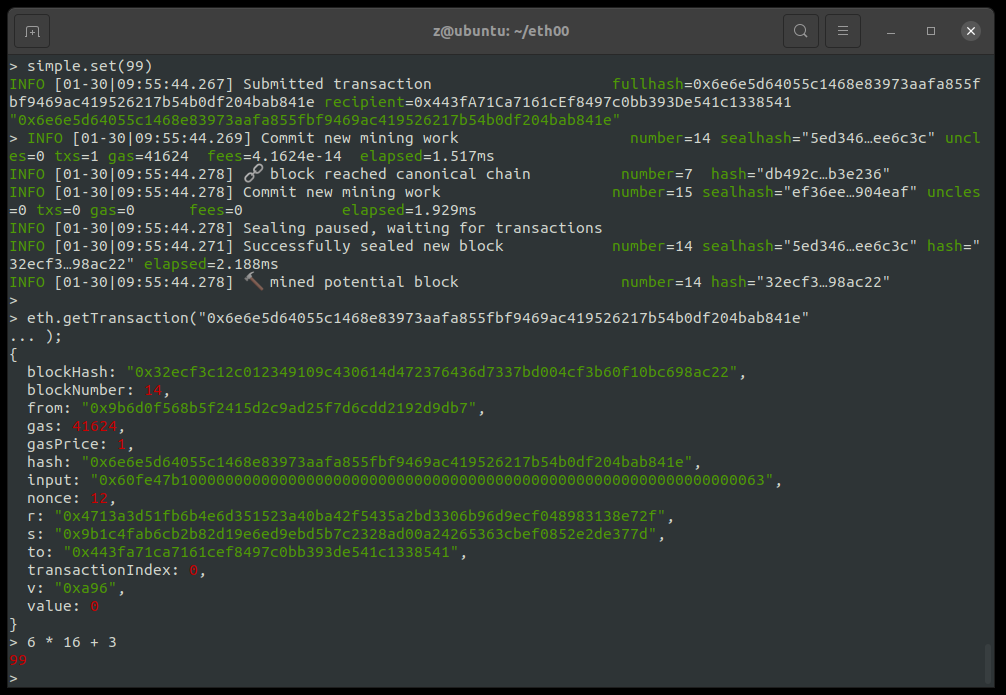


图4-25 合约方法(set( ))的调用及对应的交易信息

通过call()，可以调用相应方法，但无法对合约数据作修改，不创建交易。如果所示，可看到合约中的data已被set为99，其他方法也可以调用，如果不通过call()则会创建交易。

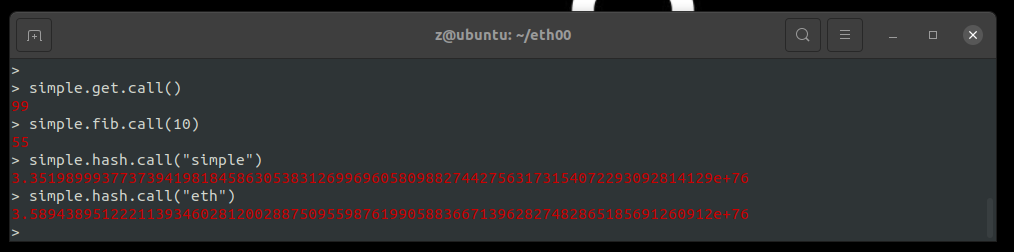
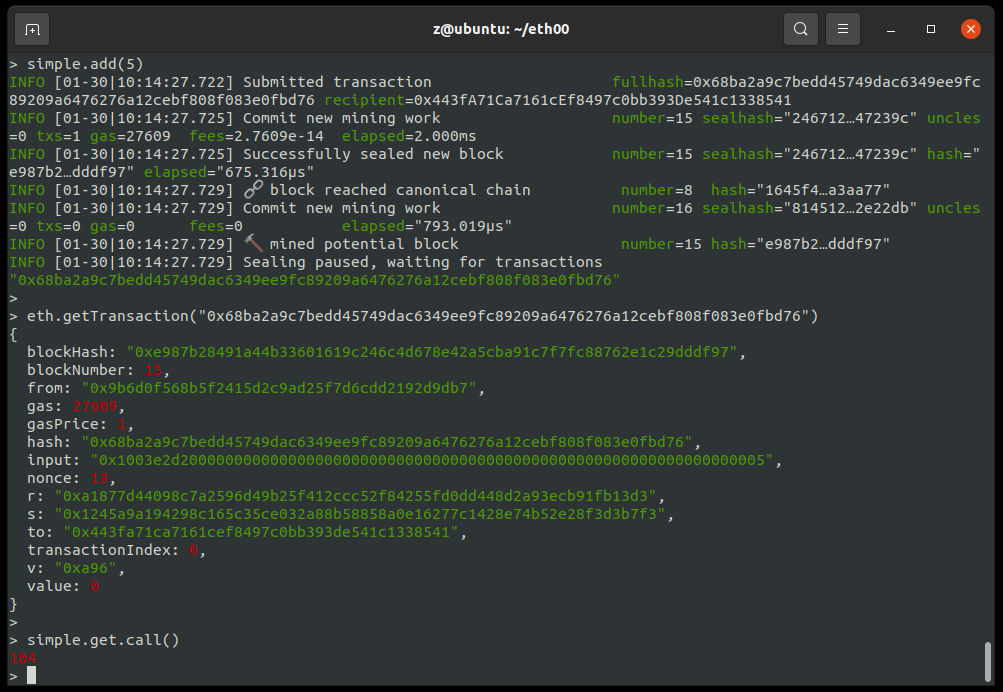


图4-26 合约方法的调用

调用add()，产生交易并上链，可看到不同的input参数，结果为99 + 5 = 104。

图4-27 合约方法(add( ))的调用及对应的交易信息

（9）挖矿和交易池

使用miner.stop()停止挖矿，此时创建数个交易将保留在交易池中，合约中的数据不会改变，只有矿工去挖矿才会执行这些交易以改变合约中的数据。

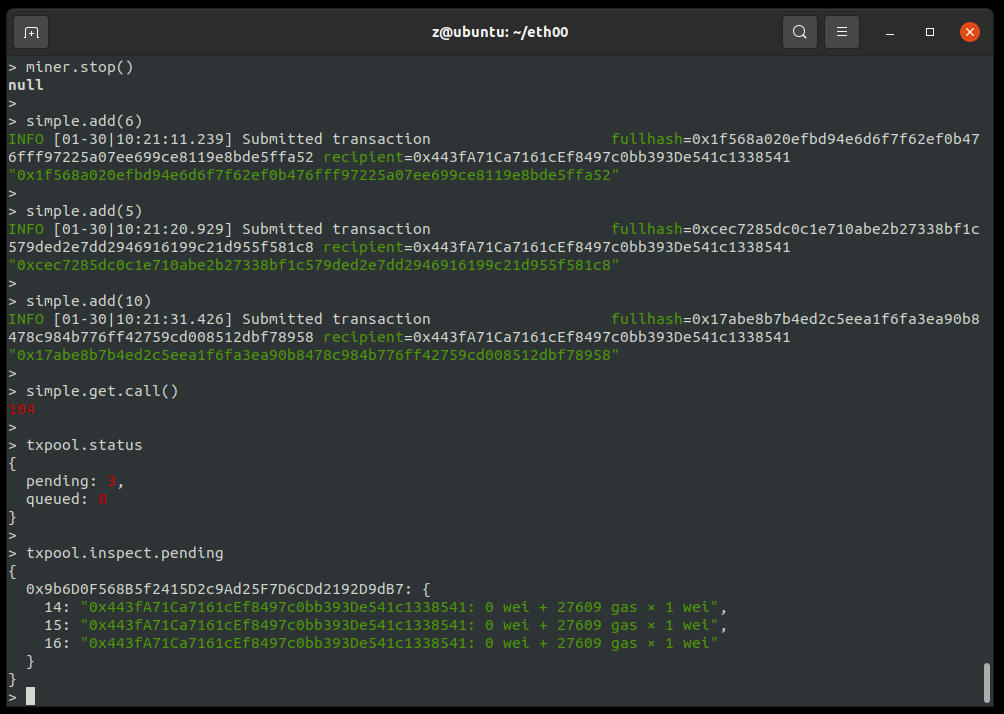


图4-28 停止挖矿对交易执行的影响

使用miner.start()开启挖矿，查看最新块信息，如图所示，这四个交易都被打包至一个块中。

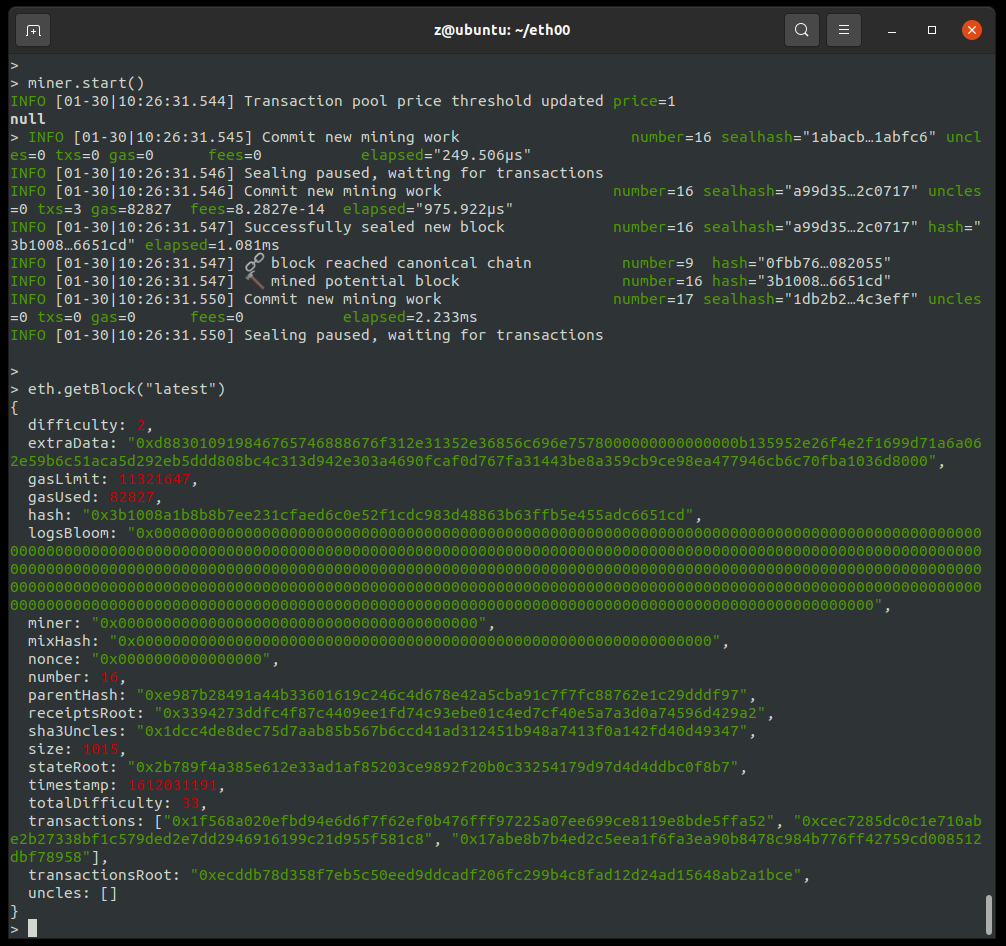


图4-29 重启挖矿对交易执行的影响

simple.get()查看，交易上链，改变了合约数据：

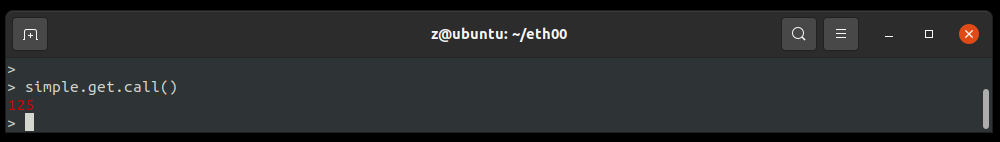


图4-30 合约数据的查看

暂停挖矿，通过不同的方法和参数产生不同的交易，因其消耗资源（计算和存储）不同，所花费的gas值也不同，如下图所示：



图4-31 查看gas的消耗情况

**【实验小结】**

本次实验主要介绍了以太坊的相关知识，具体包括智能合约的编写与部署方法、使用客户端Geth查看以太坊的信息，如账户列表、挖矿日志记录、区块与交易信息等。智能合约是区块链系统中最重要的概念之一，其也是以太坊成为区块链2.0的原因。经过本次实验，读者会对智能合约的生命周期及以太坊对其的处理流程有初步的了解，这为后续的实验打下了基础。

同时通过动手安装配置以太坊环境及实操客户端Geth，读者对理论知识会有更进一步的理解，这也为后续实验积累了实践经验。

**【习题】**

1. 在配置以太坊环境，安装依赖包和有关工具时，你有没有碰到什么问题？如果有，你是怎么解决的？

2. 在第一个智能合约中，当想调用say()方法时，为什么使用helloContract.say.call()？如果只使用helloContract.say()会怎么样？

3. 请你谈谈对以太坊中的gas的理解？

4. 以太坊中引入gas机制有什么优点？

**【参考文献】**

1. Buterin V. A next-generation smart contract and decentralized application platform[J]. white paper, 2014, 3(37).
2. Wood G. Ethereum: A secure decentralised generalised transaction ledger[J]. Ethereum project yellow paper, 2014, 151(2014): 1-32.
3. 闫莺. 以太坊技术详解与实战[M]. 机械工业出版社, 2018.