实验四 以太坊单节点搭建及合约部署

【实验介绍】

在Ubuntu20.04系统上搭建以太坊开发环境，然后使用geth工具开启一个单节点，同时部署两个简单的solidity合约。

以太坊（英文Ethereum）是一个开源的有智能合约功能的公共区块链平台，通过其专用加密货币以太币（Ether，简称“ETH”）提供去中心化的以太虚拟机（Ethereum Virtual Machine）来处理点对点合约。（摘自[百度百科](https://baike.baidu.com/item/%E4%BB%A5%E5%A4%AA%E5%9D%8A)）

以太坊目的之一是实现“世界计算机”，即这是一台全世界共有的计算机，每个节点都保存一致的计算机状态副本。当你编写一份智能合约（可看作一份代码），上传到这台“计算机”并被同步到全世界的节点后，你可以发送一个交易（可看作代码中相应函数的参数）触发该合约，某个矿工在他的节点上执行这份代码，改变相应的状态（可看作改变计算机存储区的值），并记录到一个区块中，随后他开始“挖矿”，当他找到相符的随机数后（这里设定是POW共识），他向全网广播他的区块，其他的节点接受他的区块，检查后同步自己的以太坊区块链，此时智能合约的状态变更（可看作代码的运行结果）就被全世界的以太坊节点记录下来。当然，无论是计算还是存储，这台“计算机”任何的资源消耗都是需要花费以太币的。

智能合约贯彻“Code is law!”的理念，如上所说，让这样一台“计算机”去跑普通的程序是没有意义的，但它去中心化且自动执行的特点让其成为商务协议合同的极佳选择。智能合约是计算机协议的抽象概念，本次实验我们使用的是其一种具体实现——[solidity语言](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/articles/3_features/35_contract/solidity_basic_features.html)。

【实验要求】

1. 熟悉以太坊依赖环境及相关库
2. 掌握solidity合约的编写、编译和部署

【实验过程】

1. 以太坊环境搭建（已完成）
2. 安装go语言

下载二进制包并解压到指定位置：

|  |
| --- |
| **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 |

1. 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：

|  |
| --- |
| **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 |

查看版本信息：

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

1. 安装truffle

|  |
| --- |
| **sudo** npm install -g truffle |

1. 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 |

1. 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 |

1. 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 |

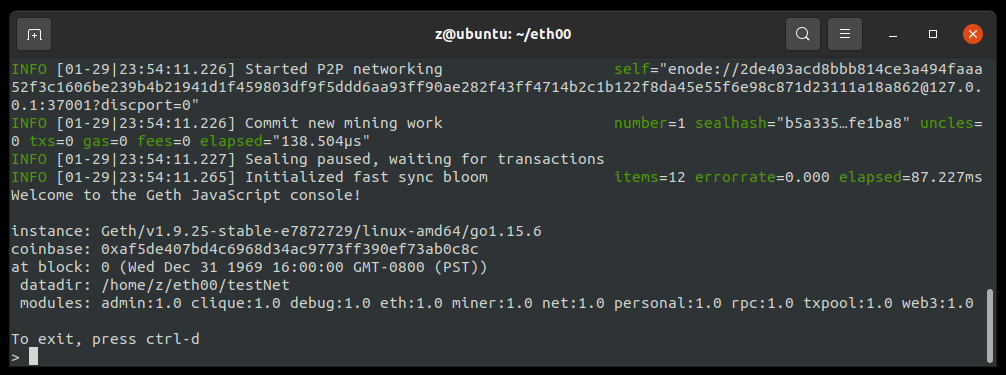
1. 合约部署
2. 启动Geth环境

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。

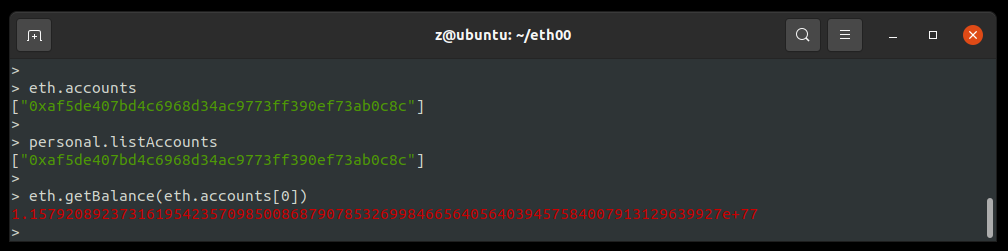
如下图所示，创建了一个以太坊测试网络：



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。



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：



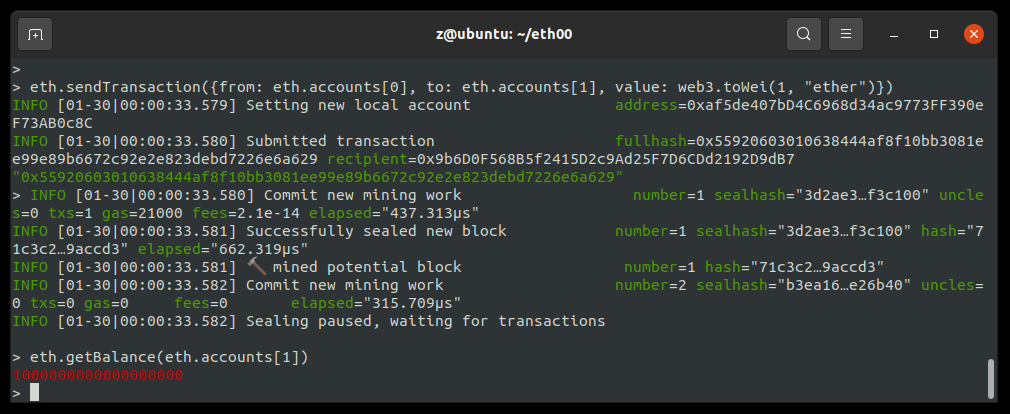
1. 创建账户

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。

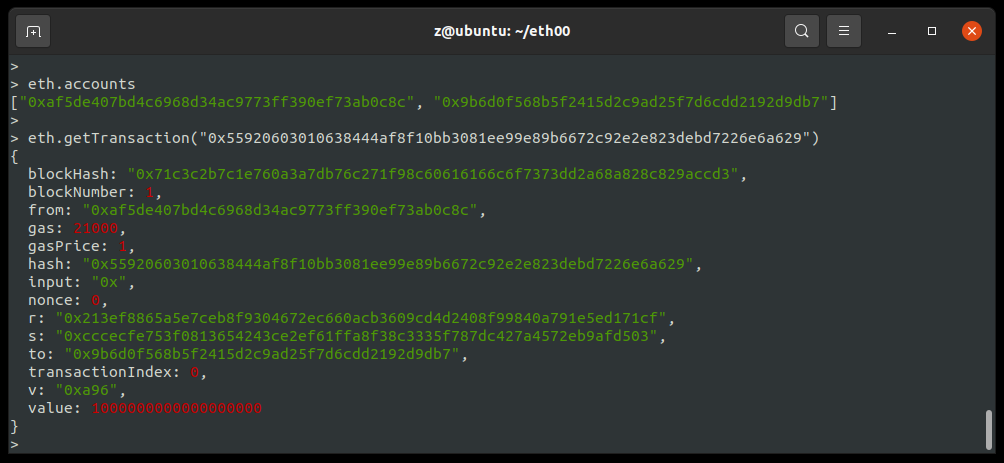


1. 转账

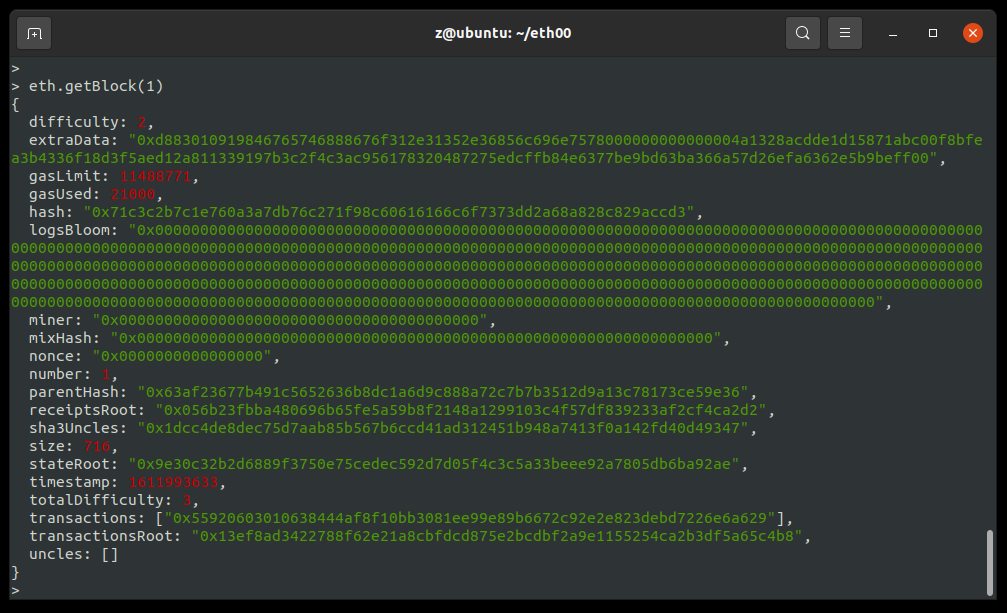
新账户余额为0，无法部署合约（需消耗以太币），故使用转账接口eth.sendTransaction()从默认账户转账1以太币到新用户，由于这是一次交易，可在下方日志信息中看到挖矿记录，其交易哈希为：0x55920603010638444af8f10bb3081ee99e89b6672c92e2e823debd7226e6a629



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。



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，以及merkle树哈希根。



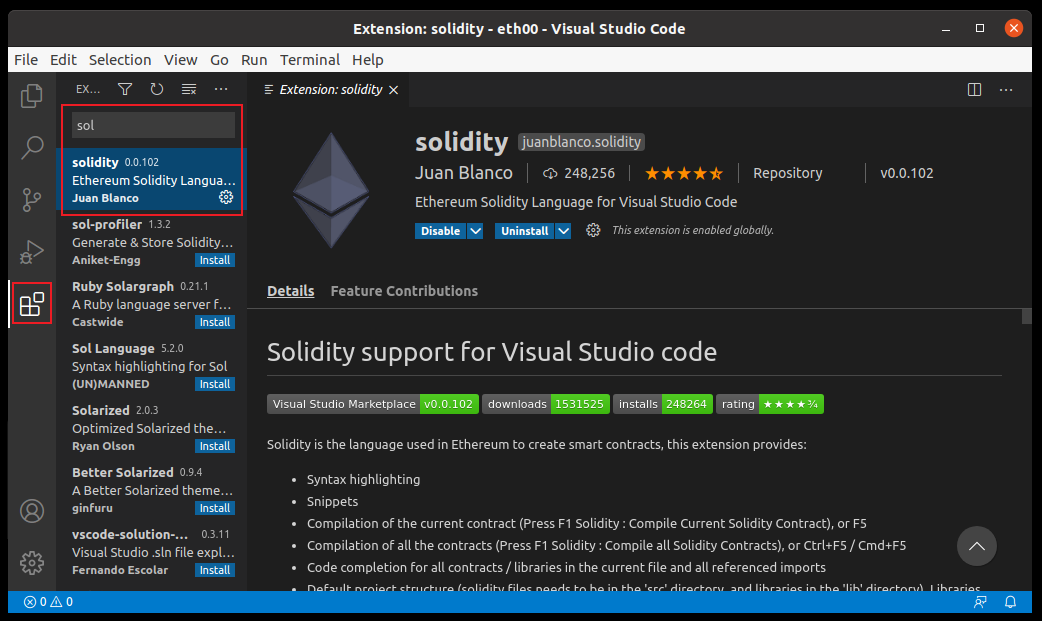
1. 编写智能合约

接下来编写第一个智能合约。

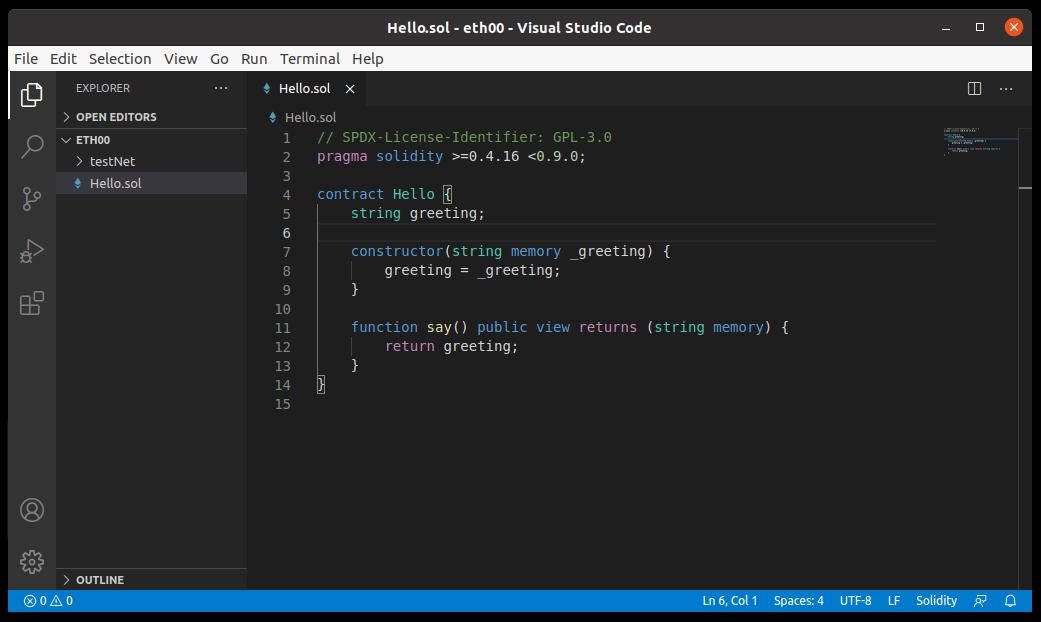
新开一个终端，输入如下命令使用vscode打开项目文件夹：

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

安装solidity扩展：



创建Hello.sol，编写solidity代码如下：



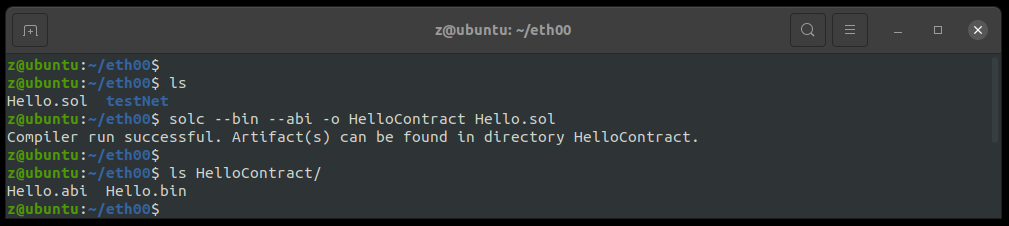
1. 编译智能合约

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。

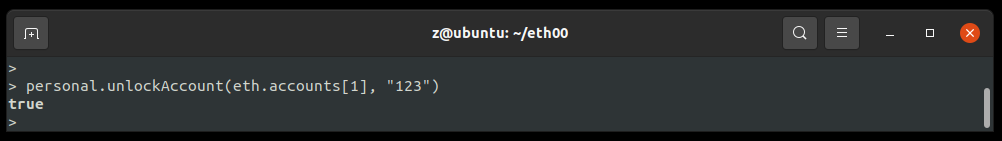
编译后，HelloContract目录下有Hello.abi文件和Hello.bin文件。



1. 解锁账户

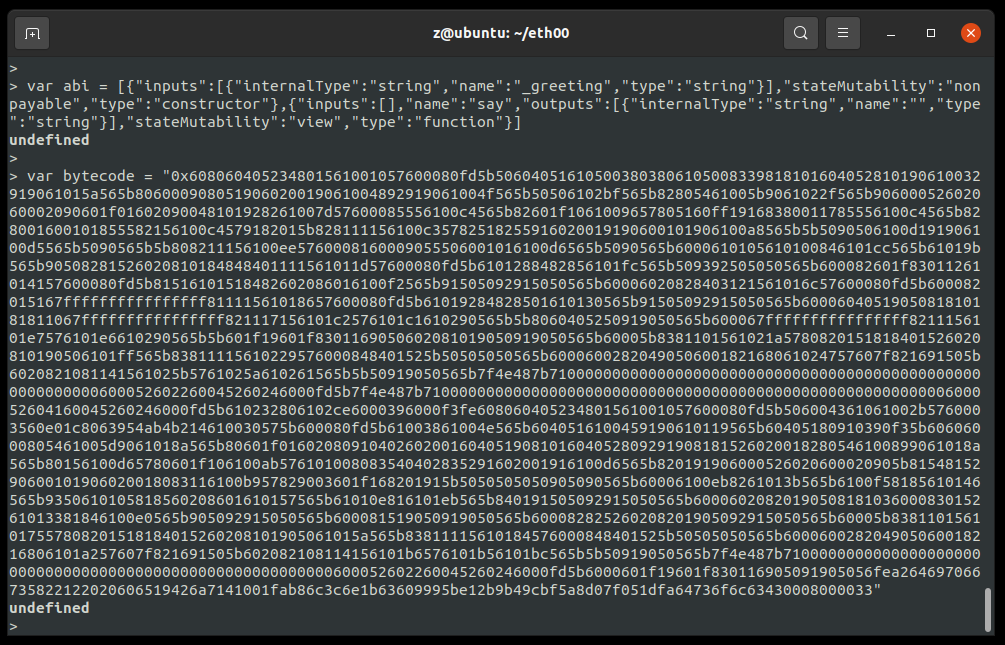
部署合约前需先解锁账户（如同银行转账需输入密码），“123”为前面创建账户设置的密码，输入下面命令后得到“true”即为解锁成功。

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

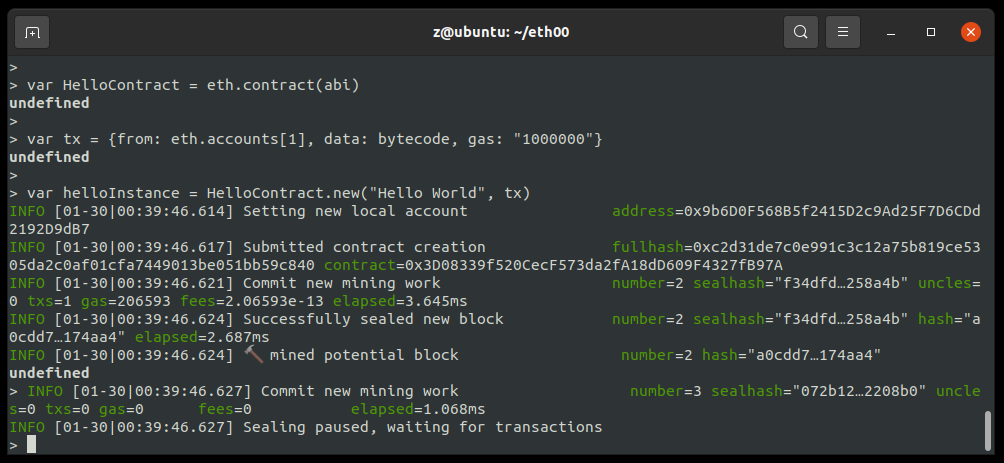


1. 部署智能合约

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**）。



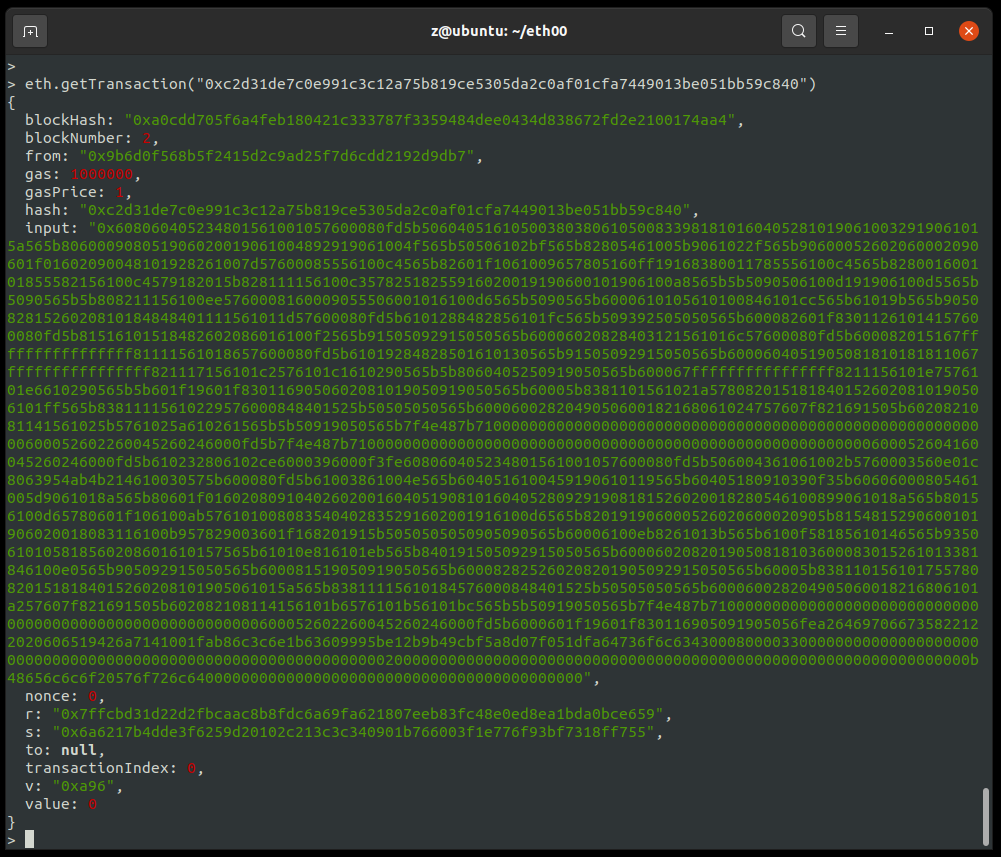
如下图所示，使用eth.contract()创建一个合约实例HelloContract，然后在用HelloContract创建一个HelloContract实例，参数为其构造函数参数和交易参数，此时已部署一个新的合约。部署合约本身也是一笔交易，发送方为eth.accounts[1]），发送数据为合约字节码，消耗一定的gas，故下图日志信息中矿工挖出一个块，块中交易为此次合约部署。**注意等待该交易上链再进行下面的操作，因为只能在合约实例获得地址后才可以调用方法。**



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。**

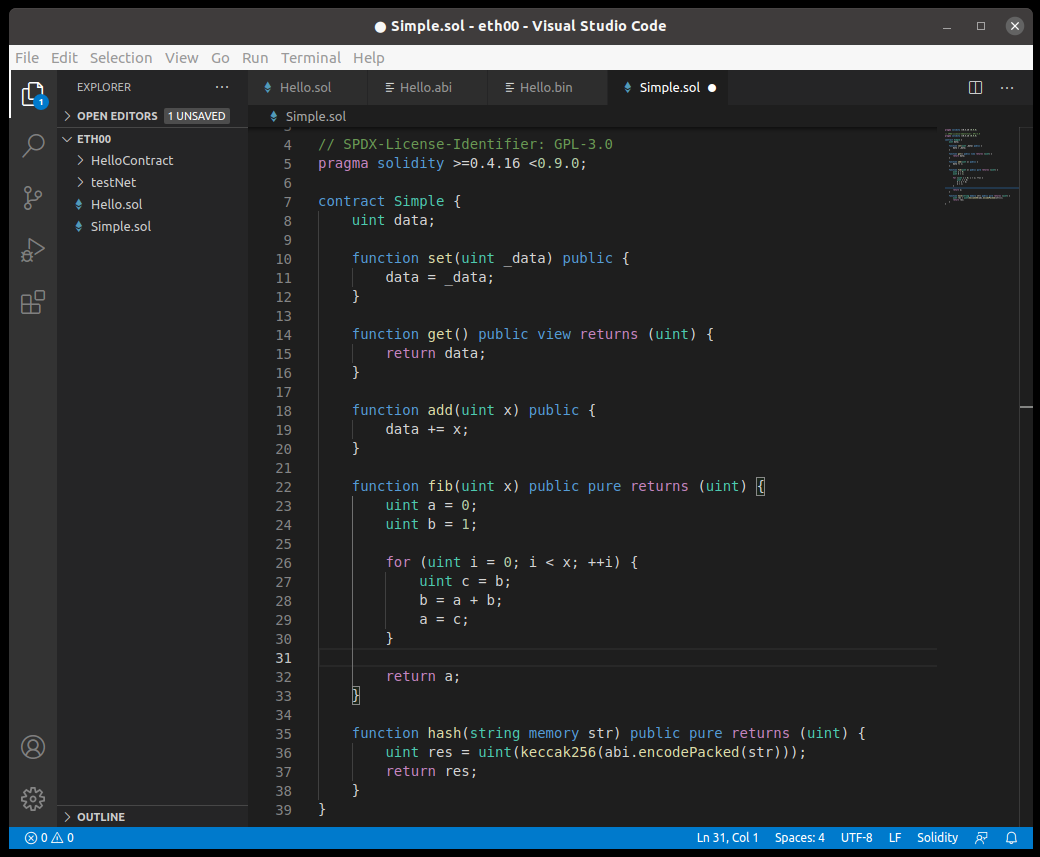


使用eth.getTransaction()查看该部署交易信息：

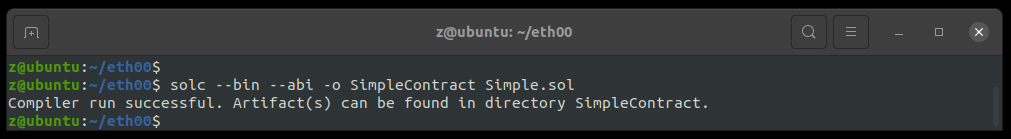


1. 部署第二个简单智能合约

编写智能合约代码如下：

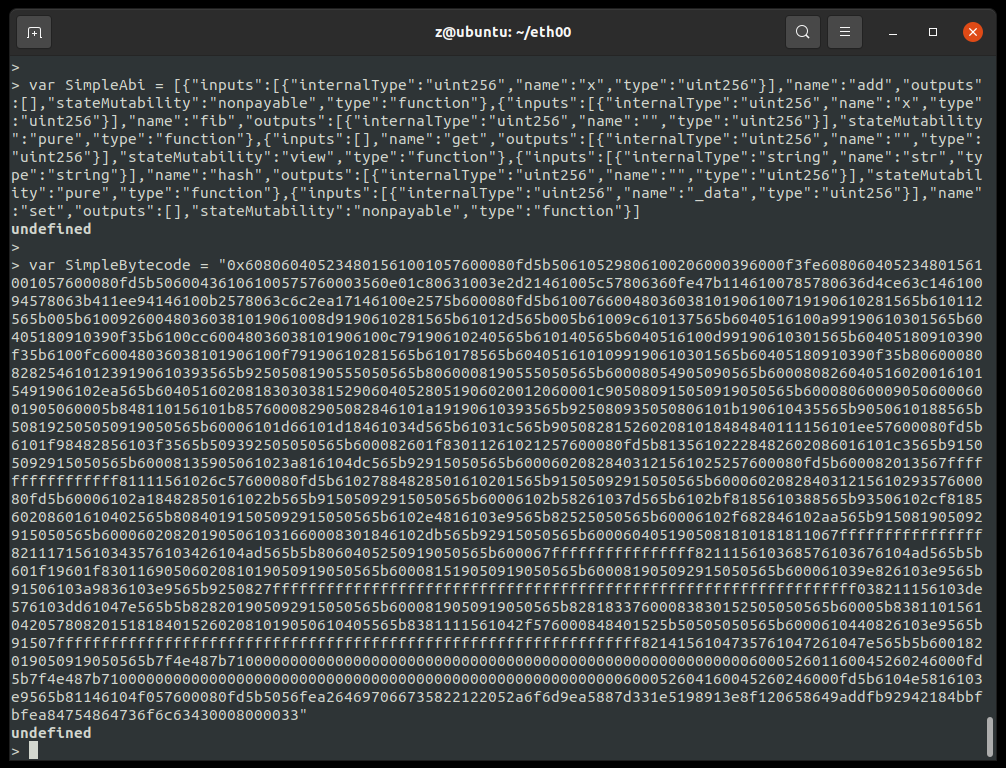


编译智能合约：

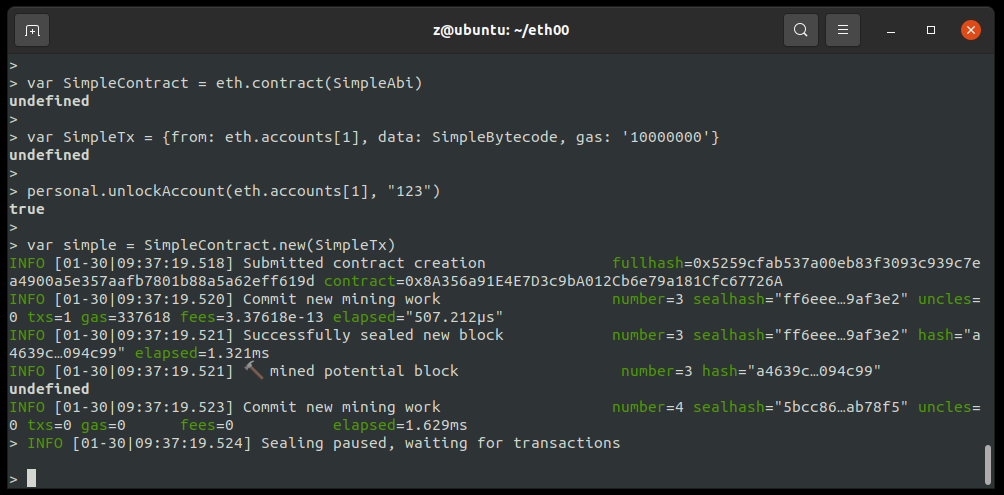


在geth中创建相应变量，将abi和字节码复制进去：

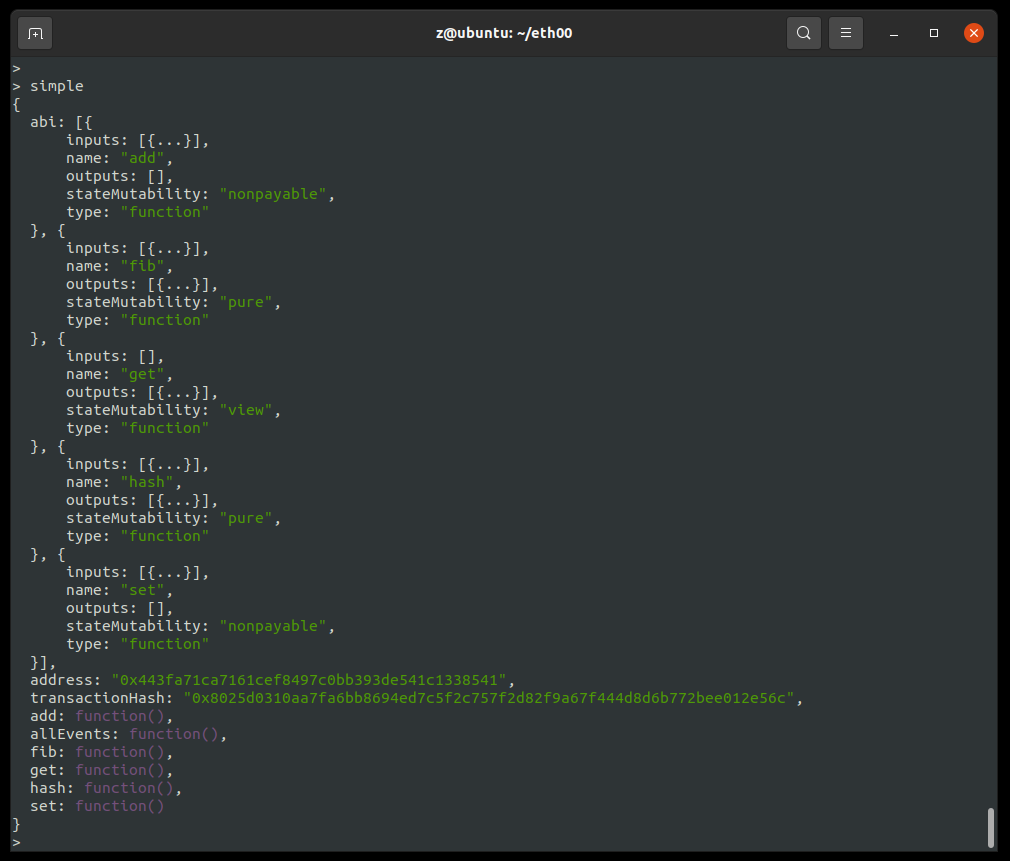
By



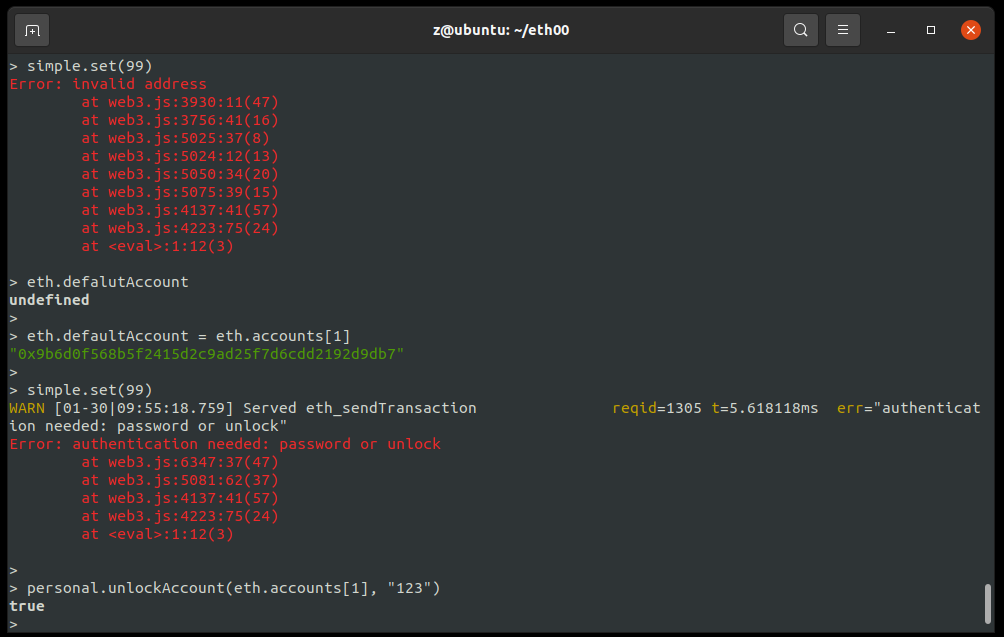
创建合约实例SimpleContract和SimpleContract实例simple:



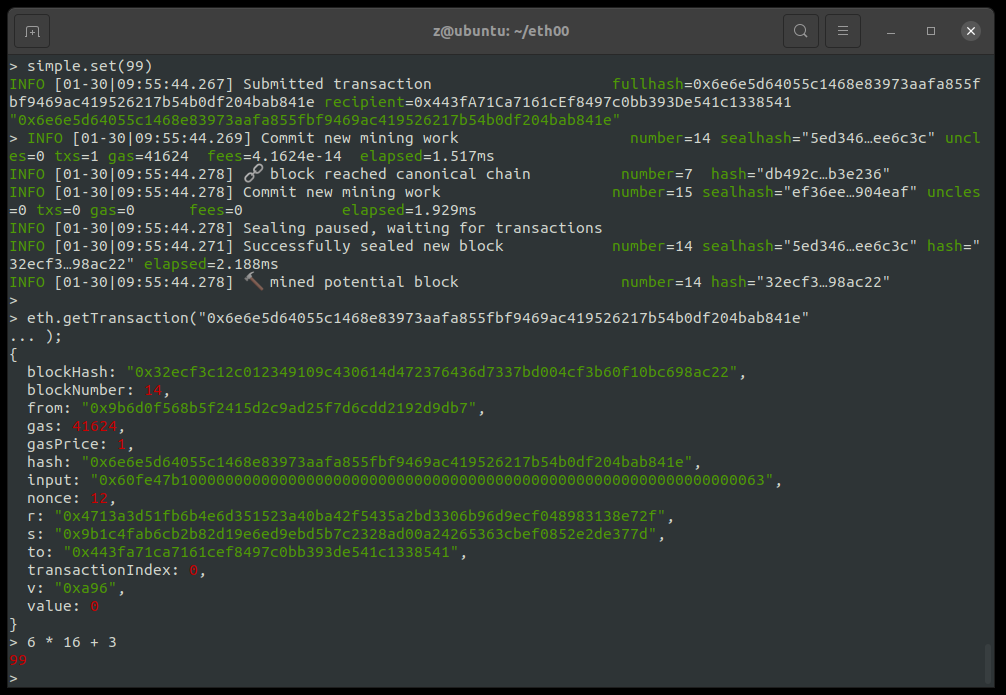
打印simple，如果address为undefined且下面无方法，可将上面的实例命令再执行一遍，重新创建一个simple实例。如果合约上链可以看到最后面的接口方法：



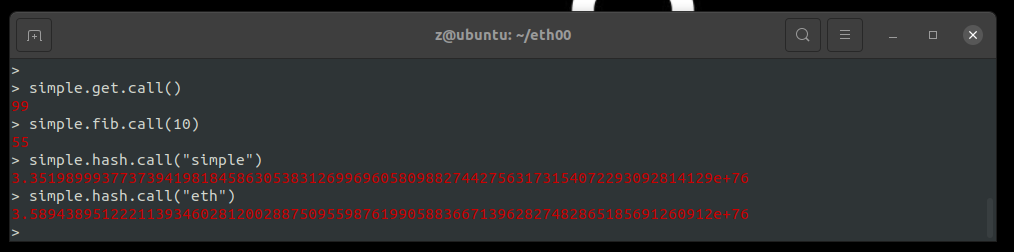
执行simple.set()方法，需设置默认账户（执行合约的账户）或解锁账户（解锁一段时间后会重新锁定，需再次解锁）:



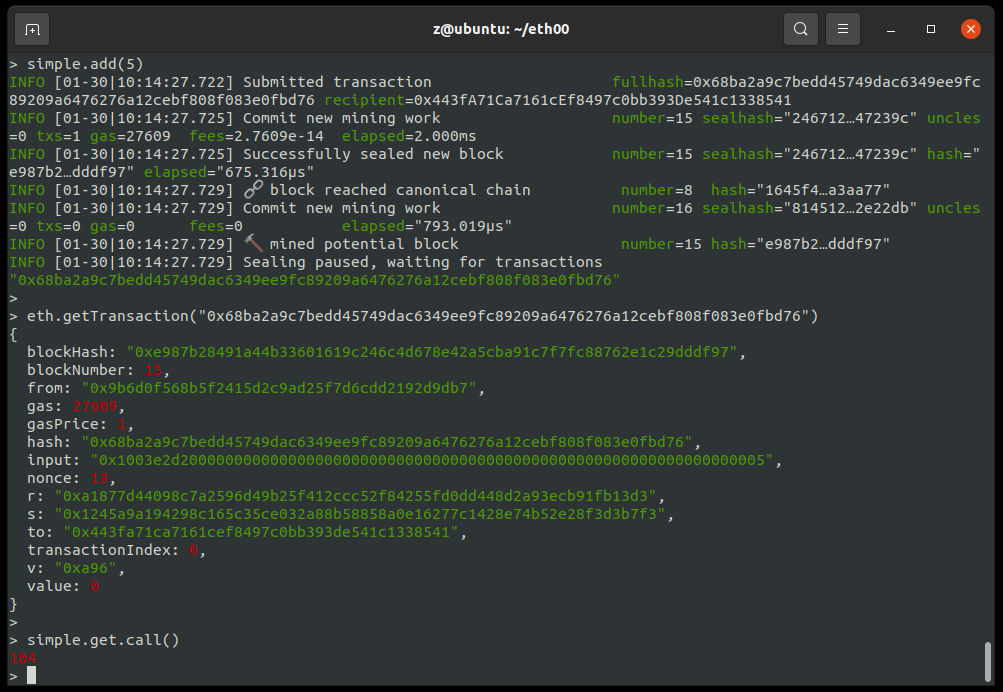
重新执行，结果如下图所示，该方法产生一笔交易，并被矿工放入块中成功上链，通过其哈希可查看相应信息，注意其中的input值，前面的0x60fe47b1是set的函数签名，后面的0x63即为set的参数值。这里我们可以知道区块链像日志一样将相应的操作都保存在链上，合约中的数据皆可溯源之。

si

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。

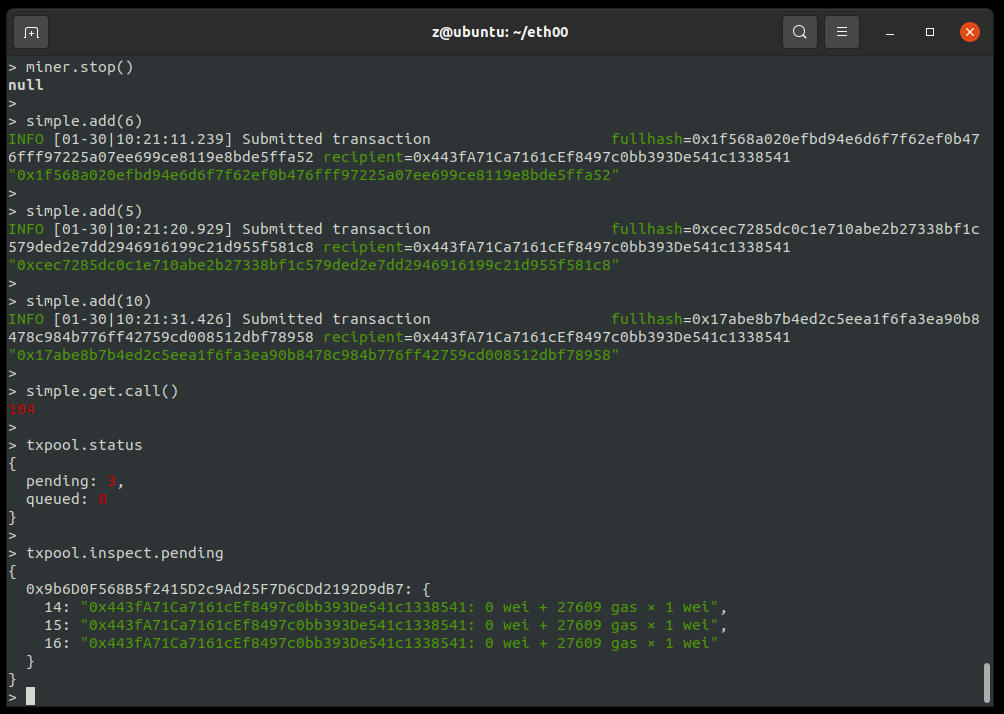


调用add()，产生交易并上链，可看到不同的input参数，结果为99 + 5 = 104。

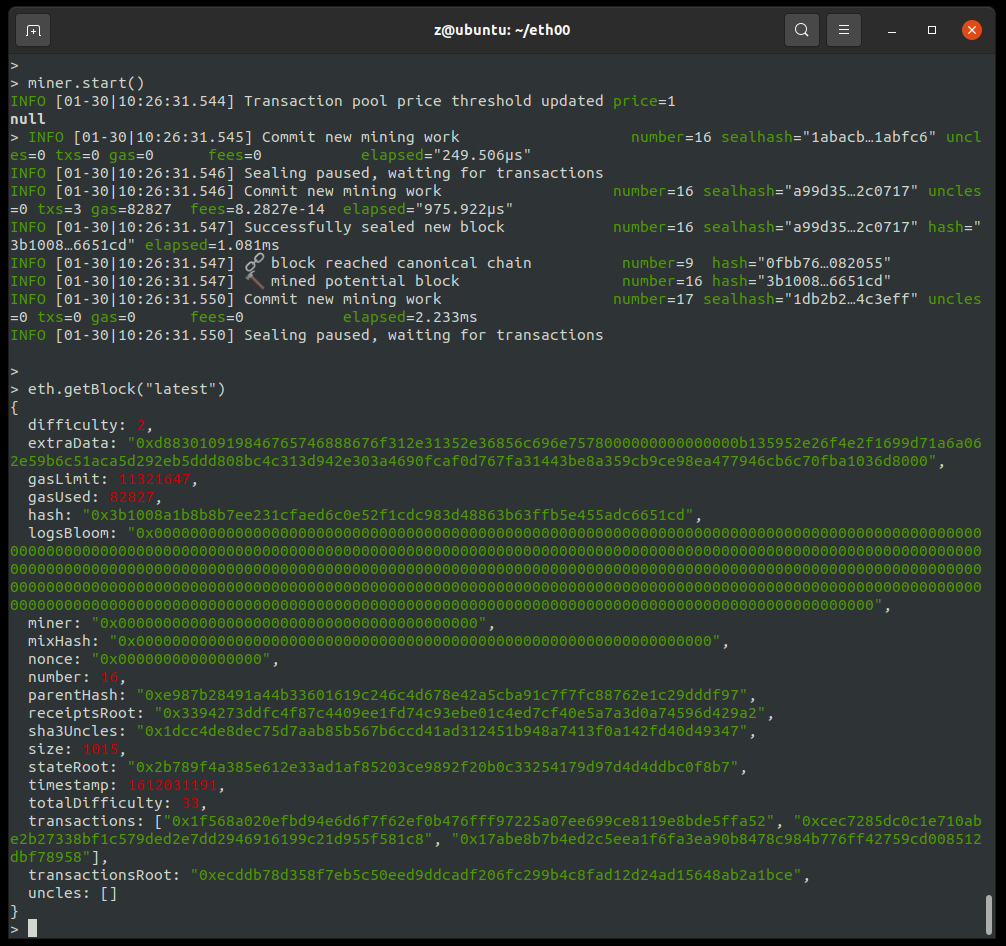


1. 挖矿和交易池

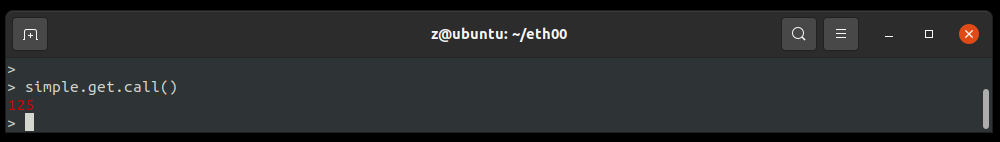
使用miner.stop()停止挖矿，此时创建数个交易将保留在交易池中，合约中的数据不会改变，只有矿工去挖矿才会执行这些交易以改变合约中的数据。



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。



simple.get()查看，交易上链，改变了合约数据：



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：

