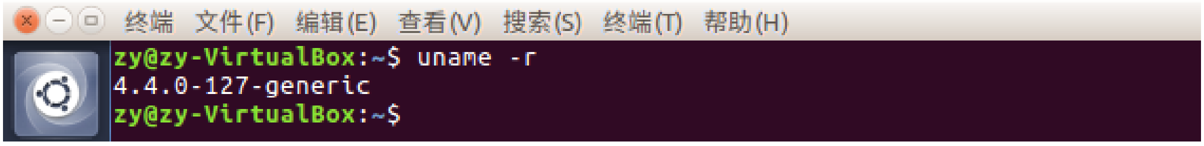
Docker Ubuntu 安装

1.Docker 要求 Ubuntu 系统的内核版本高于 3.10 ，查看本页面的前提条件来验证你的 Ubuntu 版本是否支持 Docker。

通过 uname -r 命令查看你当前的内核版本



2.获取最新版本的 Docker 安装包

wget -qO- https://get.docker.com/ | sh

输入当前用户的密码后，就会下载脚本并且安装 Docker 及依赖包

3.当要以非 root 用户可以直接运行 docker 时，需要执行 sudo usermod -aG docker runoob 命令，然后重新登陆

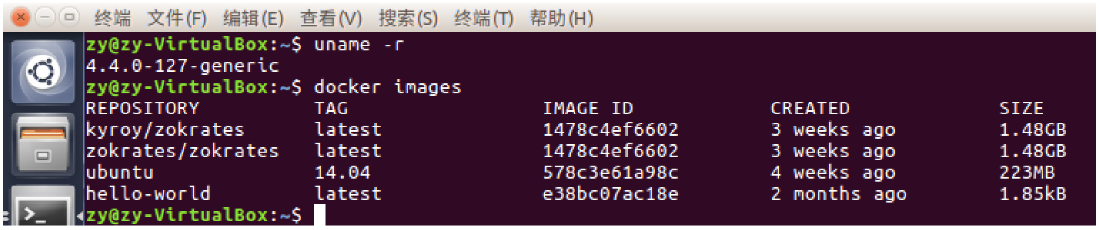
部署 zokrates

推荐使用该方法开始 zokrates

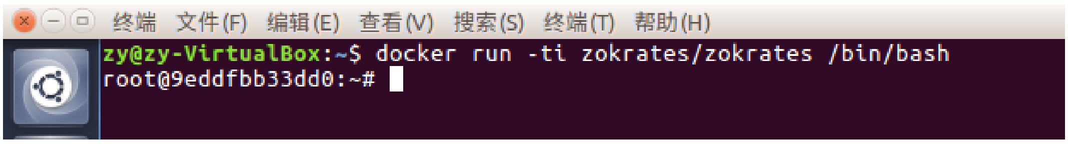
docker run -ti zokrates/zokrates /bin/bash

初次执行该条命令 docker 会自动从 docker 镜像仓库中下载 zokrates，默认是从 Docker Hub 公共镜像源下载。

可以通过 docker images 命令查看已下载镜像



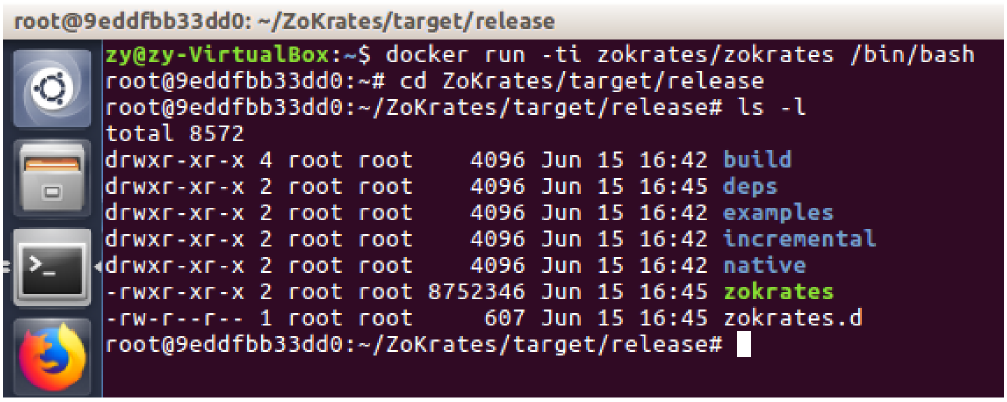
然后进入交互模式



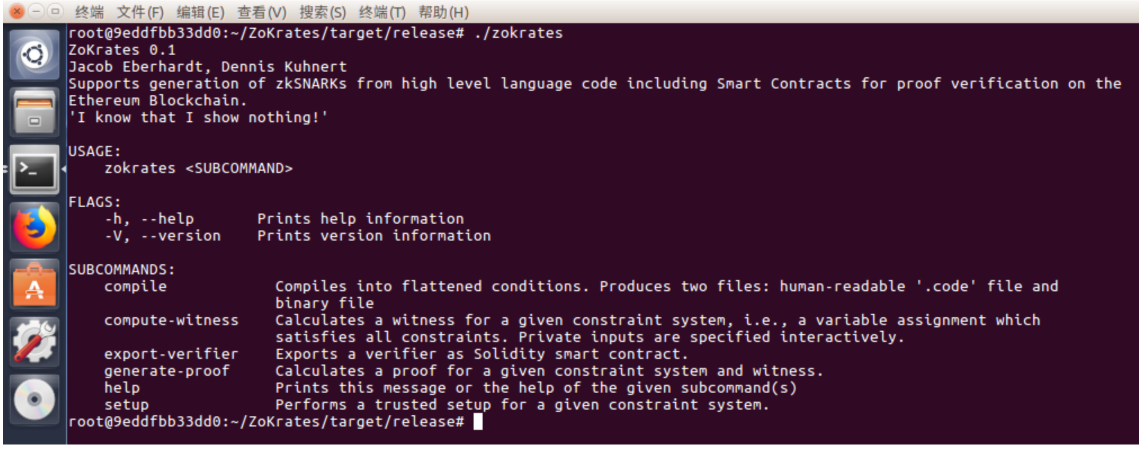
在交互模式下

cd ZoKrates/target/release

该文件夹下有可执行文件 zokrate:



Zokrates 提供命令行界面。 您可以通过运行./zokrate 查看可用子命令的概述:

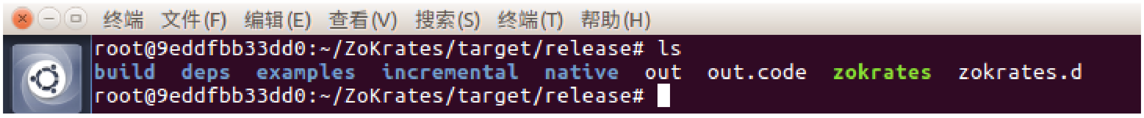


执行一个实例(位于~ZoKrates/examples/add.code)

①./zokrates compile -i ~/ZoKrates/examples/add.code

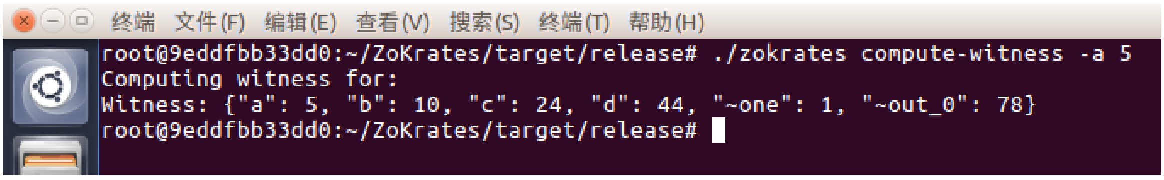
编译一个.code 文件

这个命令会创建一个编译好的 out.code 文件



②计算在 out.code 中找到的已编译程序和程序的参数的见证。 见证是包括计算结果的变量的有效分配。

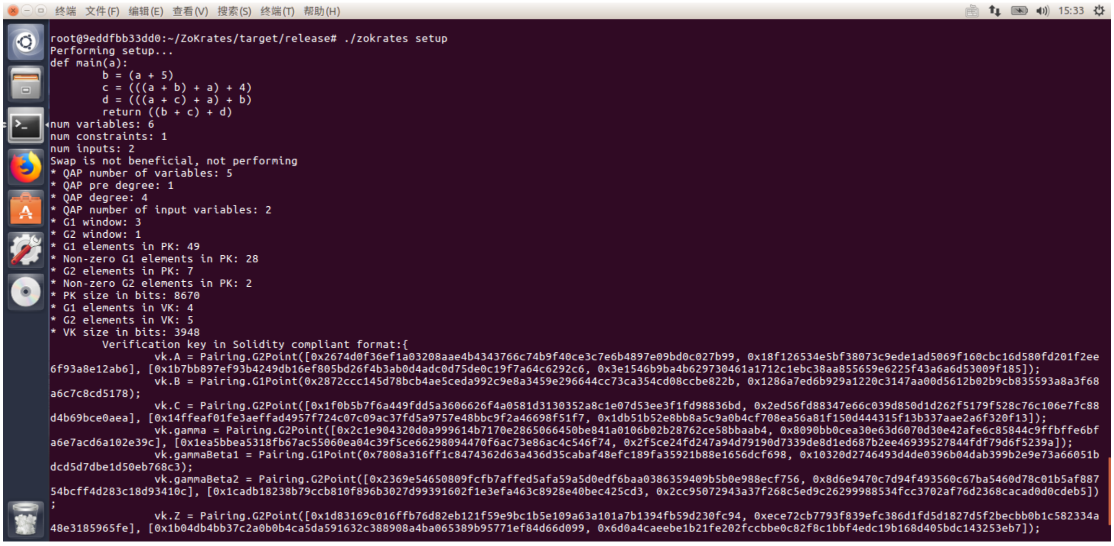
./zokrates compute-witness -a 5

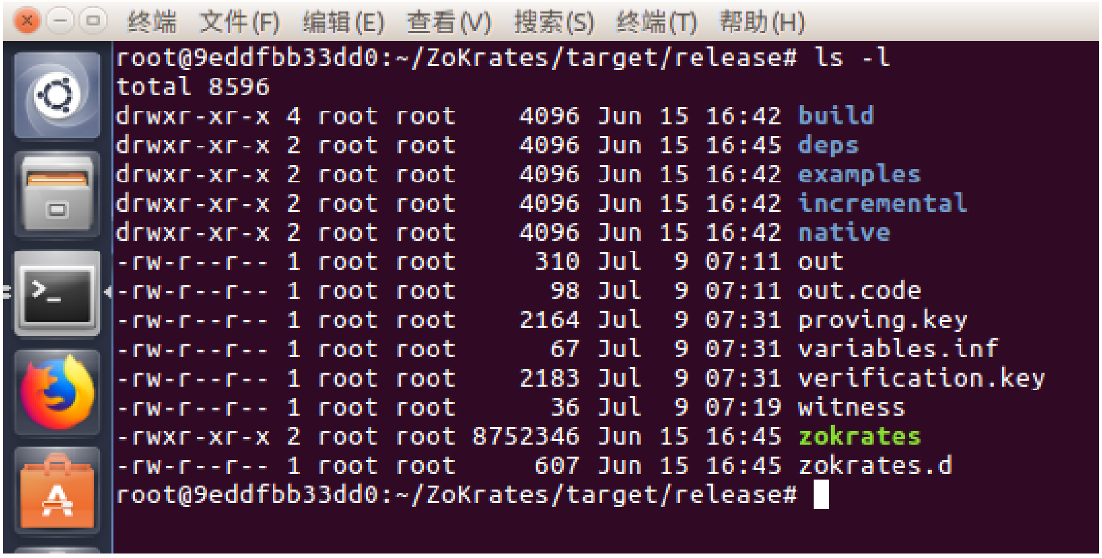


这行命令创建一个 witness 的文件

③为 out.code 中的已编译程序生成可信设置，在 proving.key 和 verificating.key 上创建一个证明密钥和一个验证密钥。

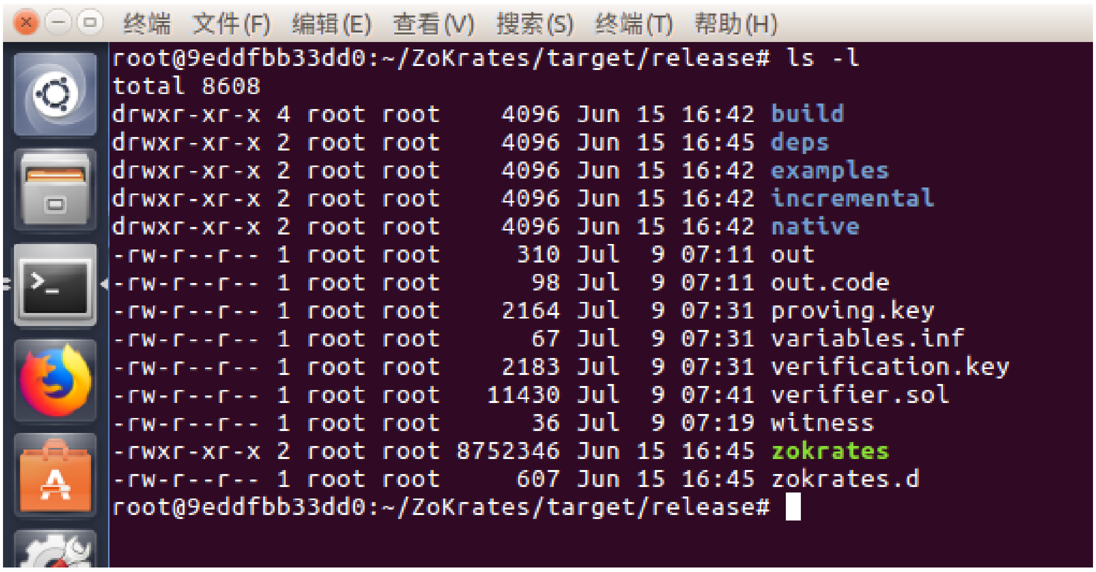
./zokrates setup





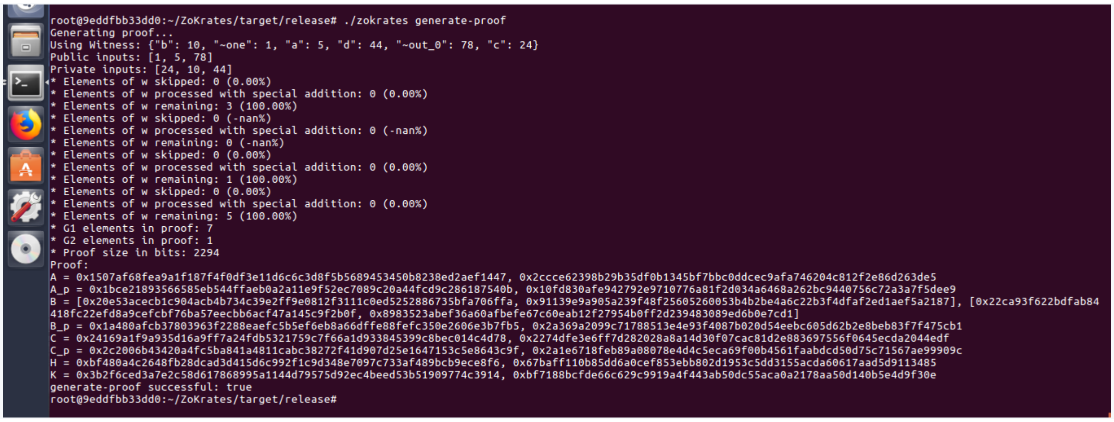
④使用./verificating.key 上的验证密钥，生成一个 Solidity 合约，以便在验证已编译程序 out.code 的计算证明。在./verifier.sol 创建验证者合约。

./zokrates export-verifier



⑤使用./proving.key 中的证明密钥，生成用于计算已编译程序 out.code 的证明，从而验证 witness。

./zokrates generate-proof



⑥通过验证者合约，可以检查此证明。 例如，使用 web3，调用将如下所示

sudo docker cp 9eddfbb33dd0:/root/ZoKrates/target/release/verifier.sol /opt/soft/

//将 verifier.sol 从 docker 容器中移出

//如何进入上次启动的容器

//docker ps -a

//docker start 9eddfbb33dd0

//docker exec -ti 9eddfbb33dd0 /bin/bash

//不然重新 docker run 的话会重新开一个容器

利用 truffle 及 ganache-cli 部署该合约

truffle console

>>Verifier.at(<verifier contract address>).verifyTx(A, A_p, B, B_p, C, C_p, H, K, [...publicInputs, ...outputs])

//[5,78]

ZoKrates 调用的 libsnark 接口

//在 ZoKrates/lib 中的 wraplibsnark.cpp wraplibsnark.hpp 中再次封装

//下面是主要调用的函数

#include "libsnark/algebra/curves/alt_bn128/alt_bn128_pp.hpp"

// alt_bn128_pp 是一种椭圆曲线

#include "libsnark/zk_proof_systems/ppzksnark/r1cs_ppzksnark/r1cs_ppzksnark.hpp"

//包含 zksnark 主要函数

① r1cs_ppzksnark_constraint_system<alt_bn128_pp> createConstraintSystem(const uint8_t* A, const uint8_t* B, const uint8_t* C, int constraints, int variables, int inputs)

//创建约束系统

// r1cs_constraint_system<Fr<alt_bn128_pp> > cs;

//调用类 r1cs_constraint_system 默认构造函数

②r1cs_ppzksnark_keypair<alt_bn128_pp> generateKeypair(const r1cs_ppzksnark_constraint_system<alt_bn128_pp> &cs)

//调用 r1cs_ppzksnark_generator<alt_bn128_pp>(cs);

//生成 verification_key 和 proving_key

//下面是两个最主要的函数

③bool _setup(const uint8_t* A, const uint8_t* B, const uint8_t* C, int constraints, int variables, int inputs, const char* pk_path, const char* vk_path)

// alt_bn128_pp::init_public_params(); 初始化椭圆曲线参数

// r1cs_constraint_system<Fr<alt_bn128_pp>> cs; 初始化约束系统

// cs = createConstraintSystem(A, B ,C , constraints, variables, inputs); 为 cs 赋值

//r1cs_ppzksnark_keypair<alt_bn128_pp> keypair= r1cs_ppzksnark_generator<alt_bn128_pp>(cs); 生成 verification_key 和 proving_key

//serializeProvingKeyToFile(keypair.pk, pk_path);

//serializeVerificationKeyToFile(keypair.vk, vk_path); 将 pk、vk 输出到文件中

④ bool generate_proof(const char* pk_path, const uint8_t* public_inputs, int public_inputs_length, const uint8_t* private_inputs, int private_inputs_length)

// r1cs_ppzksnark_proof<alt_bn128_pp> proof = r1cs_ppzksnark_prover<alt_bn128_pp>(pk, primary_input, auxiliary_input); 产生证明