Mininet使用初步

# 一、Mininet环境配置

本实验使用的系统是ubuntu 20.04。

首先打开终端，输入指令：sudo apt-get install git，安装git：



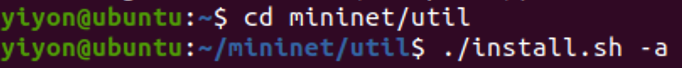
从github上获取Mininet.git：



自定义安装Mininet：

输入指令cd Mininet/util进入指定文件夹；

输入指令./install.sh -a默认安装全部特性：

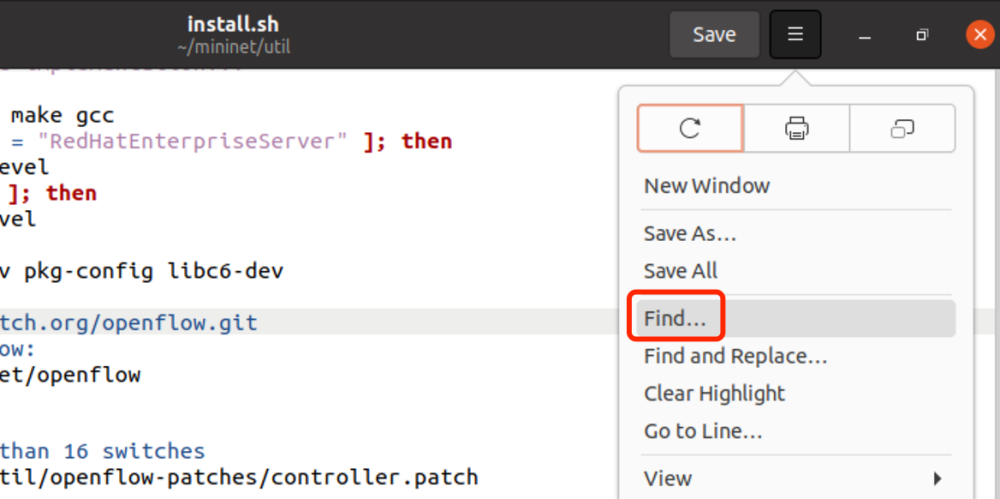


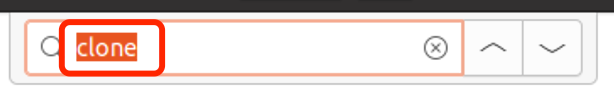
在安装的过程中，会涉及到组件文件夹的下载（如oflops、oftest、openflow和pox等等）由于shell文件里提供的下载源地址都是来自与github，所以容易造成下载缓慢、安装失败的问题，在此提供一个换源的方法，以提高下载速率和安装成功率。

输入指令gedit install.sh打开shell文件并进行编辑：



在打开的shell文档中，点击工具栏中的find，查找关键词“clone”：





查找第一处出现clone的地方，可以看到如下图所示代码：



可见openflow的源地址来自于github，我们将其换到国内源地址，而gitee是一个很好的选择。换源方法如下：

进入gitee官网，注册账号：



注册好账号后，鼠标指向个人页面的右上角“+”处，点击“新建仓库”：



因为是要把已经存在的仓库克隆到自己的仓库里，所以选择右上角的“点击导入”：



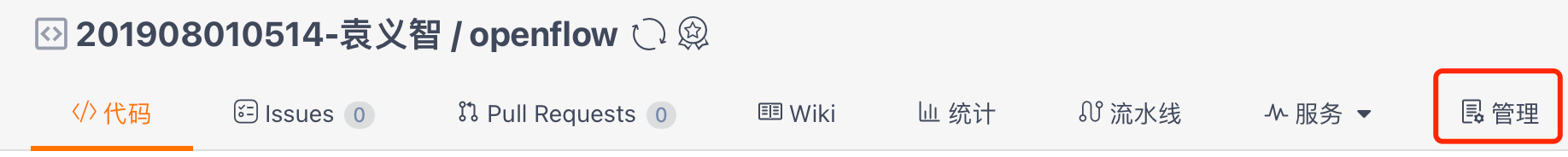
将我们在shell文档里找到的github源地址粘贴到“Git仓库URL”里，网页会自动检测源地址的正确性，并自动为仓库命名：



无需做其他改动，点击最下方的“导入”，等待克隆完毕。

短暂等待后，openflow仓库克隆完毕。但是现在仓库还处于“私人”状态，这样会导致在ubuntu中进行clone操作时会频繁地要求输入账号和密码，所以现在将openflow仓库的权限设置为“公开”：

点击菜单栏里的“管理”选项，进入openflow的仓库管理页面：



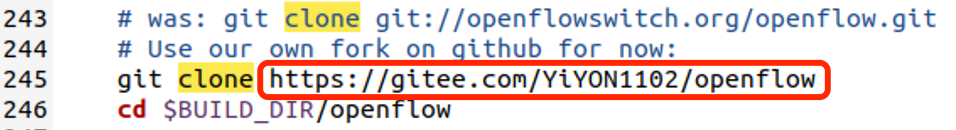
在“仓库设置”的“基本信息”里，填写仓库的“介绍”，选择“是否开源”为“开源”，勾选所有“仓库公开须知”，最后点击最下方的“保存”。



现在，openflow的仓库权限已经被设置为“公开”了：



进入openflow仓库，复制此仓库的地址，并替换shell文档里原有的地址：

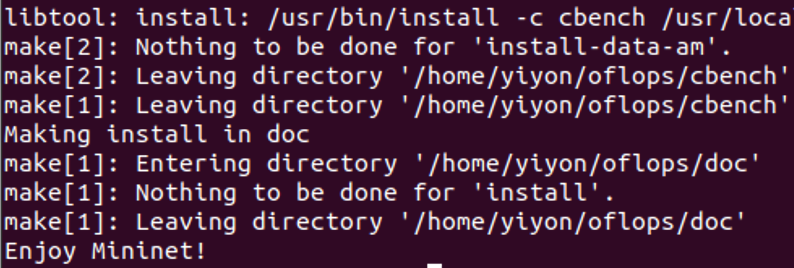


这样在terminal进行git操作时，就会按照新提供的源地址进行下载，能够大大提高下载和安装的速度和成功率。

对剩下的主要配件文件夹如法炮制，经过多次尝试，建议替换的文件夹有：openflow、pox、oftest和oflops。如果在最初下载Mininet文件夹时也遇到下载慢的问题，也可以替换Mininet的源地址。

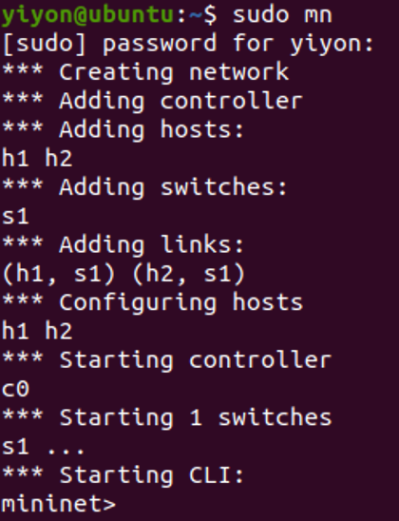


成功安装后，界面如下：

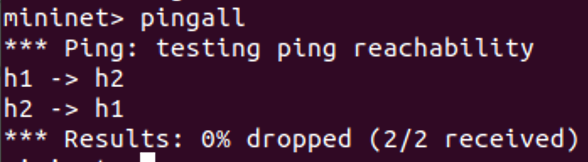


进行Mininet的安装验证：

输入指令sudo mn，进入Mininet并创建一个最小的拓扑，此拓扑只有两台主机h1和h2，以及一台交换机s1:

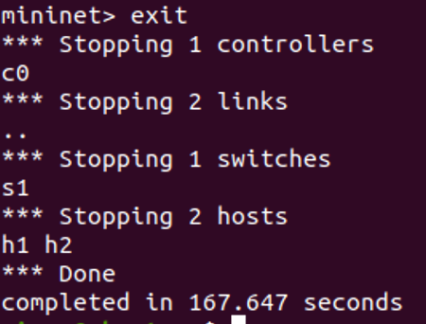


输入指令pingall，测试连通性：



成功ping通，测试成功。

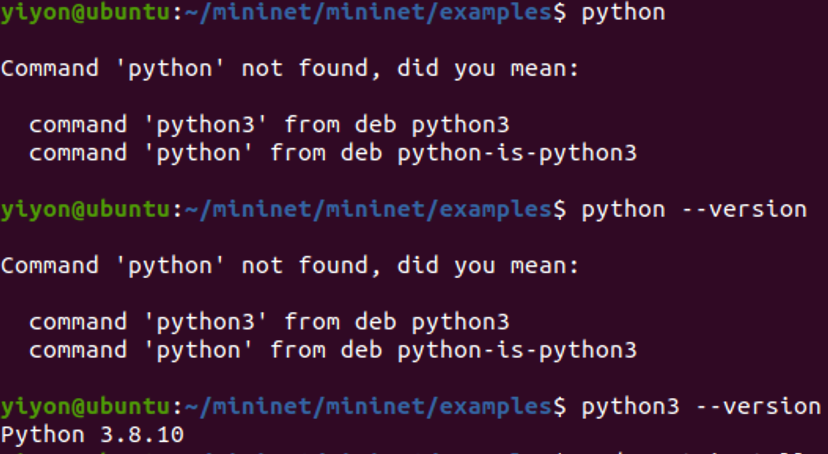
输入指令exit退出Mininet：



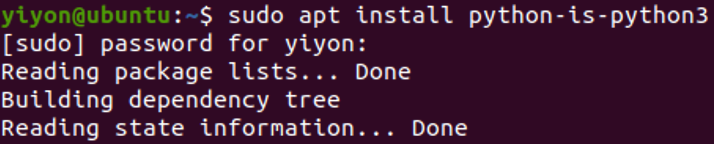
至此，Mininet已经成功安装。

由于我们后面还将用到可视化工具miniedit以及使用Python语言撰写脚本，所以我们还需要将虚拟机的python语言环境配置好。由于Mininet更支持Python2和Python3，而Python2和Python3是不兼容的，为了保持统一性，我们将在本实验中全部使用Python3，以下是配置Python3的步骤与方法：

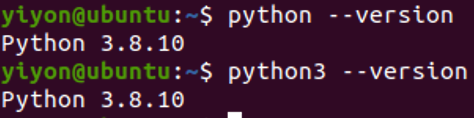
输入指令python，发现无法找到指令，查找Python版本为未安装，查找Python3版本为3.8.10。



输入指令sudo apt install python-is-python3安装软件包，此软件包会将Python指令映射到Python3：



现在指令Python与指令Python3都映射到Python3:



注：

在使用Mininet创建了拓扑后，若要进行下一项实验，应输入指令sudo mn -c删除已经创建的拓扑，避免相同名字的端口被占用。

在运行了进程（失败）之后，会显示地址已被占用，无法进行下一次实验。该错误是由于另一个控制器进程（也就是上一次失败的实验）正在使用该地址和端口，或者是由于之前控制器进程意外终止而没有释放该地址和端口，导致新的控制器进程无法绑定。解决该问题的方法是找到占用该地址和端口的控制器进程，并停止该进程。

输入指令sudo lsof -i :<portid>，该指令会输出所有占用<portid>端口的进程信息。根据该信息确定哪个进程是控制器进程，然后使用kill命令停止该进程，输入指令sudo kill <pid>即可，之后便能继续实验。

# 二 、Mininet可视化演示及基本操作演练

## 实验目的

本实验将会利用miniedit.py脚本来实现Mininet的可视化操作，以达到在图形界面上直接设计任意拓扑的目的，并可以生成自定义拓扑的Python脚本，以便后续的操作。通过学习本节实验，我们将掌握：

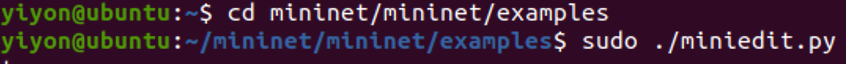
1. miniedit的启动方式
2. 如何自定义可视化拓扑，并配置设备信息

## 实验原理

在Mininet2.2.0里配置了可视化工具miniedit，此脚本文件在目录/home/Mininet/Mininet/examples目录下，执行该脚本即可进入可视化页面进行操作。

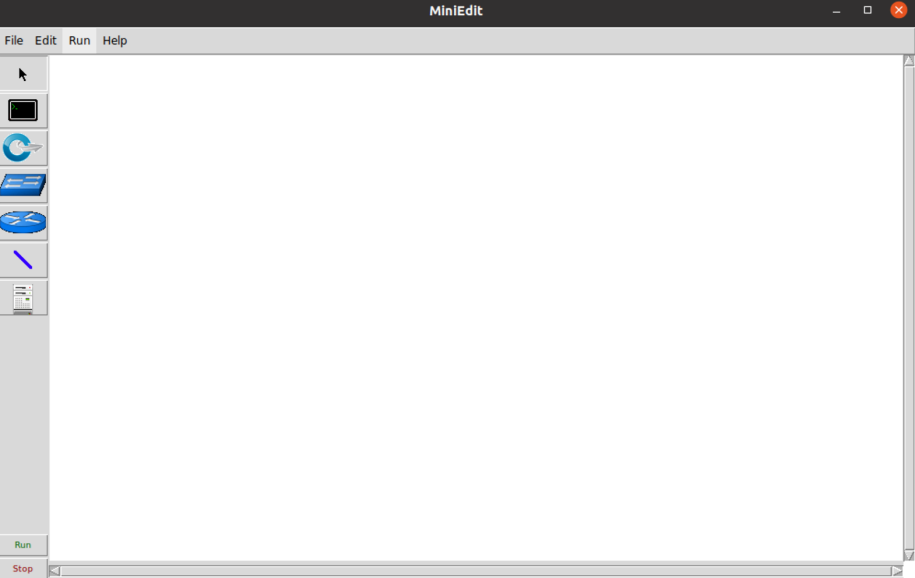
## 实验过程

进入文件夹/Mininet/Mininet/examples，输入指令./miniedit.py执行可视化脚本文件：



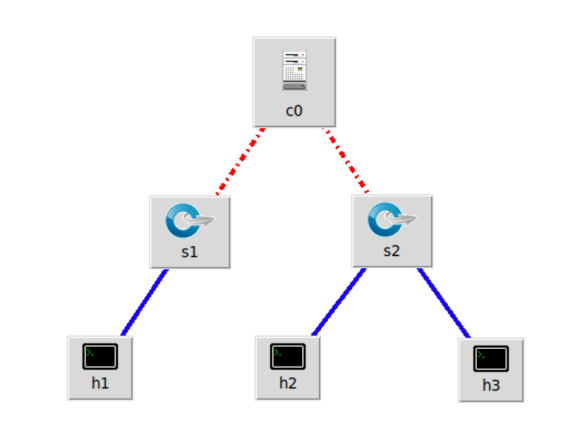
（注意：Mininet需要在sudo命令下执行）

可视化界面如下：

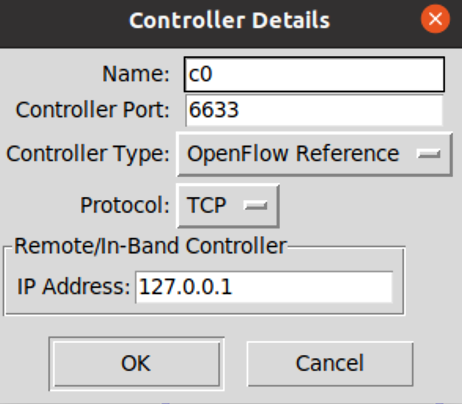


选择左栏的组件，再在右边的空白区域点击，即可部署部件。左栏的组件从上到下依次为：鼠标指针、主机Host、交换机Switch、传统交换机LegacySwitch、网线NetLink和控制器Controller。

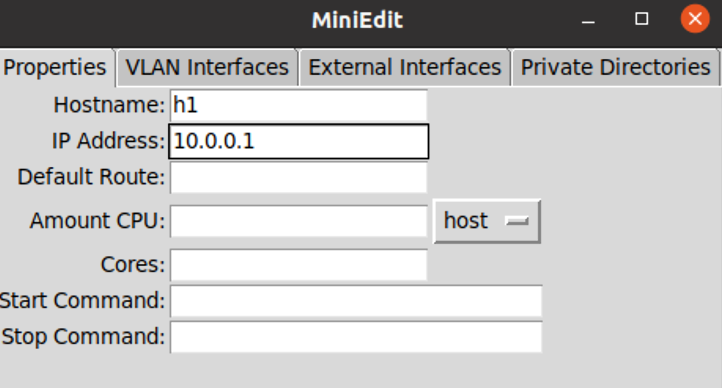
如图创建一个简易拓扑：



将鼠标指针移到组件图标上方，右键选择Properties即可设置组件的属性，如控制器的属性设置如下图：



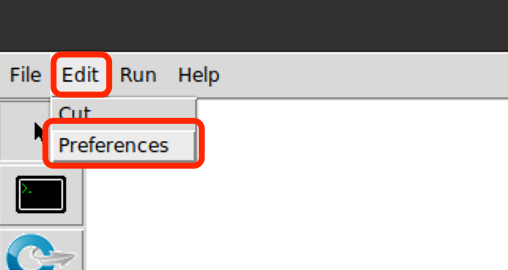
又比如在主机h1的属性里设置IP地址：

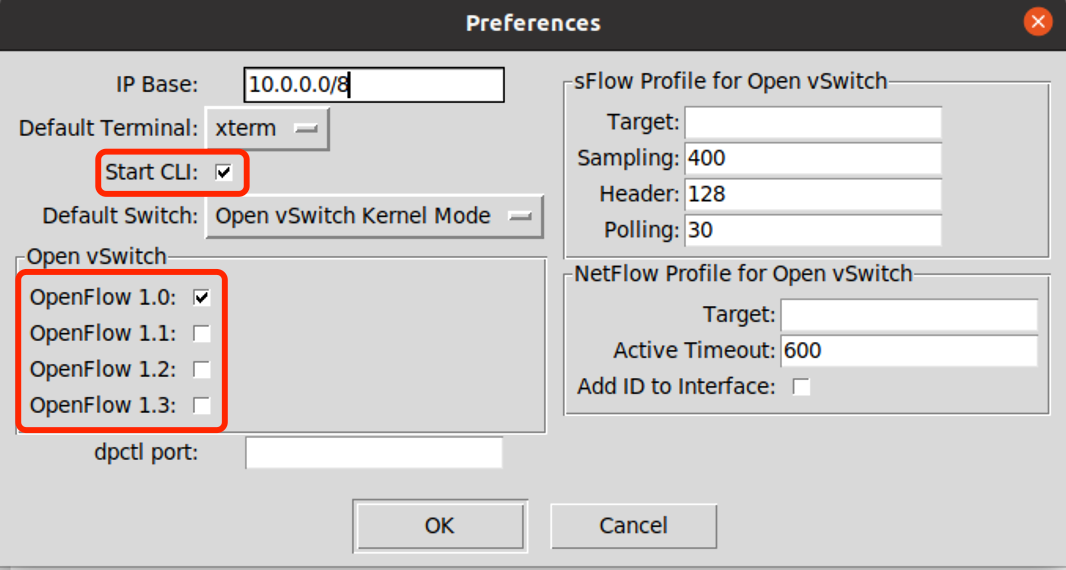


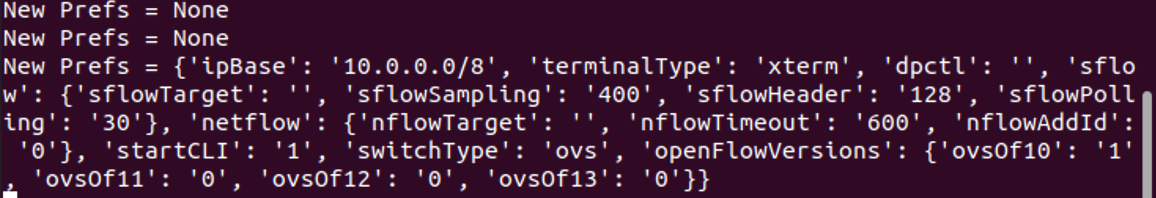
点击OK确认修改，在terminal里会出现如下命令行：



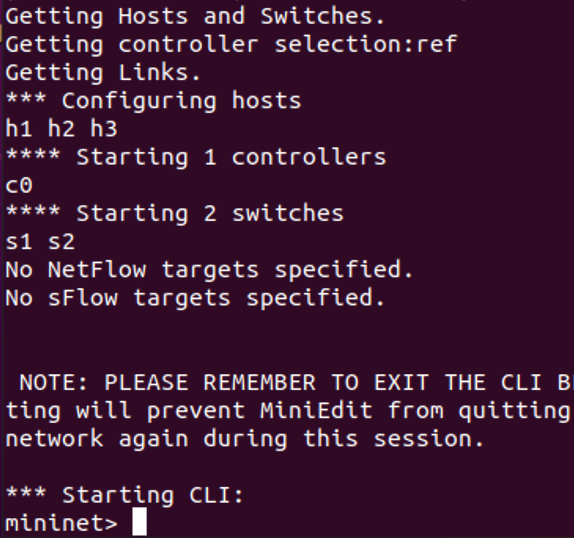
在Edit菜单来里点击Preferences，进入界面后可勾选Start CLI选项，之后就能在命令行界面直接对主机等进行操作，也可以更换交换机支持的OF协议版本（可多选）。



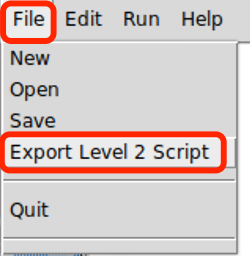




点击菜单栏里的run，启动Mininet，即可运行先前设置的拓扑。在terminal里可以看到运行的拓扑信息：

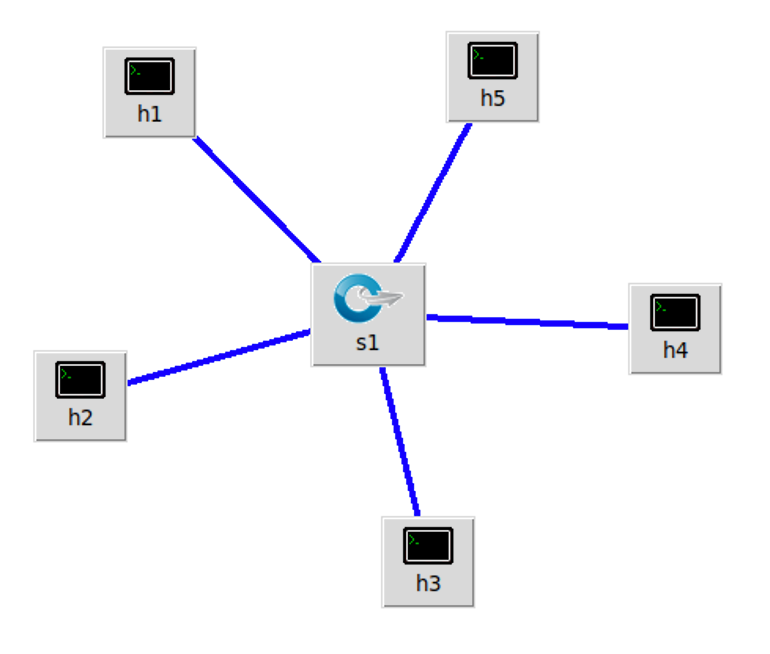


此外，在进行可视化设计后，可以在可视化界面选择将设计好的拓扑保存为python脚本，这样之后可以直接通过脚本来操作拓扑，不需要再重新设计：

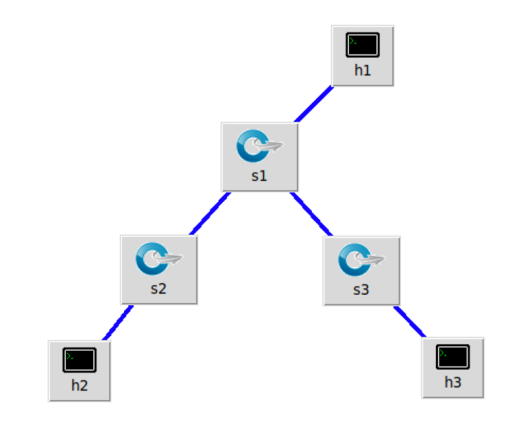


使用可视化界面创建五种常见的拓扑类型：

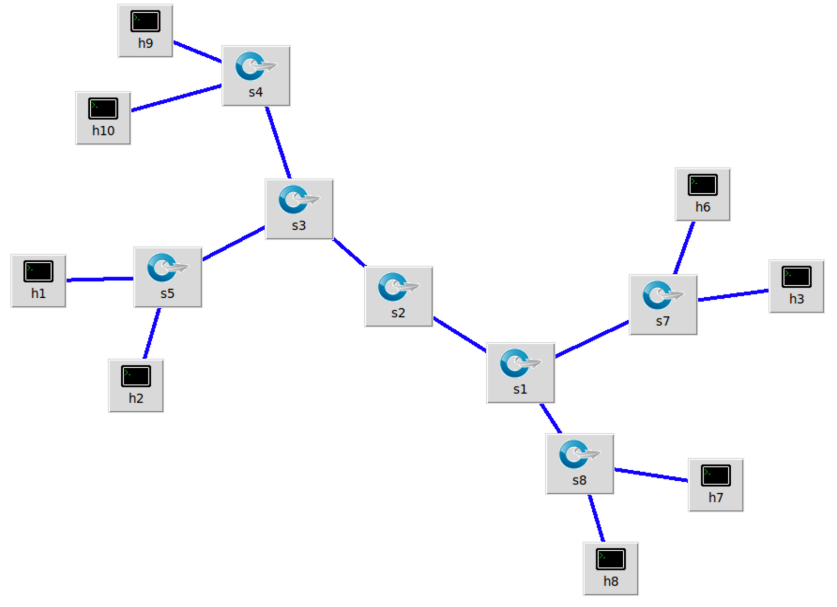
1. single类：一个交换机，主机的个数可以任意设定，呈星型；



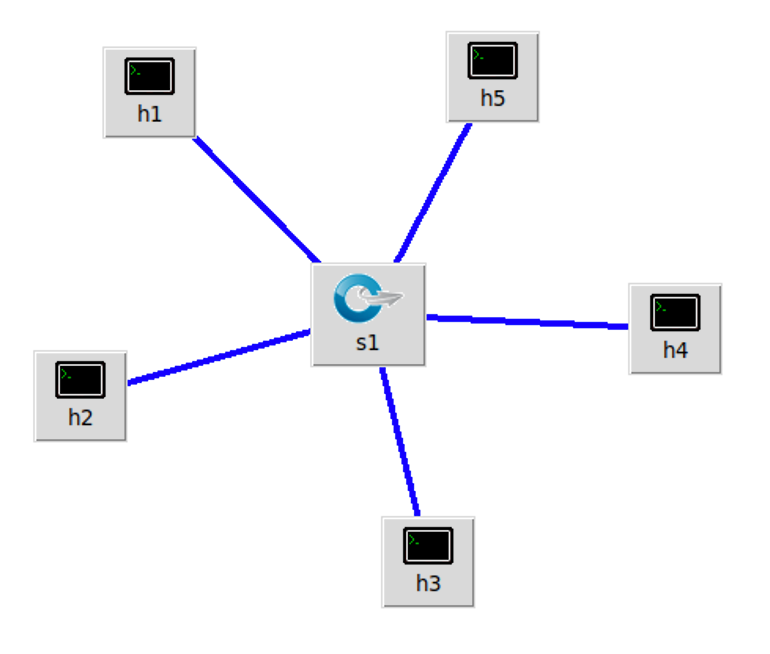
1. linear类：一个交换机连接一个主机，主机和交换机的个数都可以任意设定，呈直线型；



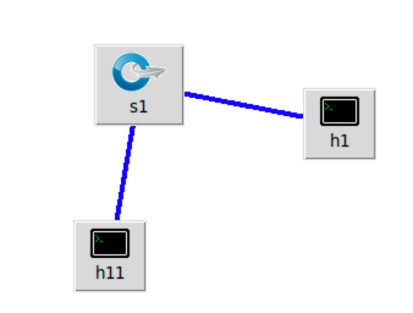
1. tree类：形似二叉树，拓扑中的交换机呈树杈状分散，最底层的交换机连接两个主机；



1. reversed类：此拓扑类型与single类拓扑一致；



1. minimal类：此类拓扑是最小的、也是默认的拓扑，一般在一个交换机连接两个主机。

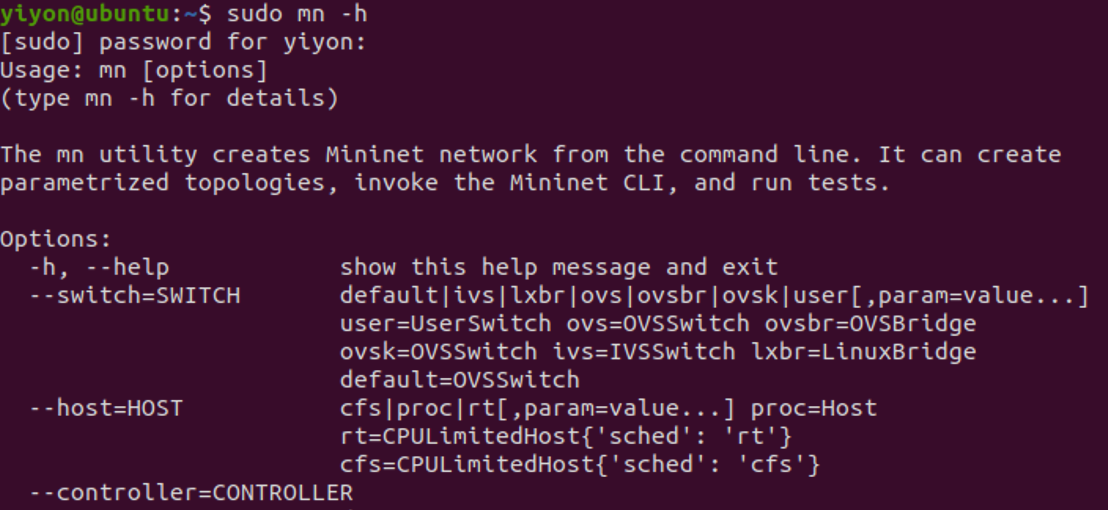


# 三、Mininet命令行操作实验

Mininet自定义拓扑两种实现方式：命令行创建、Python脚本编写。

Mininet是一个可以模拟虚拟网络的工具，支持多种拓扑结构。对于一些特定的需求，我们可以使用自定义拓扑的方式来创建自己所需要的拓扑结构。

输入指令sudo mn -h查看启动Mininet的相关参数：



比较常用的几个参数有switch、controller、topo和custom。这里进行简要介绍：

|  |  |
| --- | --- |
| 参数名称 | 功能 |
| switch | 用来指明将要使用什么样的交换机的类型 |
| controller | 用来指明控制器 |
| topo | 用来指明将要使用的拓扑类型 |
| custom | 用来进行用户自定义拓扑，读取python文件中的拓扑 |

其中，常用的拓扑类型一般有五种：single、linear、tree、reversed和minimal。

除了help指令，一些常用的查看信息的指令还有：

|  |  |
| --- | --- |
| 参数名称 | 功能 |
| nodes | 列出所有结点 |
| net | 列出所有网络链路 |
| dump | 列出结点、链路相关信息 |
| h1 ifconfig -a | 显示h1的网络接口信息 |
| s1 ps -a | 显示s1的所有进程 |

用于测试网络的连通性的指令：

|  |  |
| --- | --- |
| 参数名称 | 功能 |
| h1 ping h2 | h1 ping h2 |
| h1 ping -c 5 h2 | h1 ping h2，并有-c限定次数 |
| pingall | 所有主机间互ping |

其他命令：

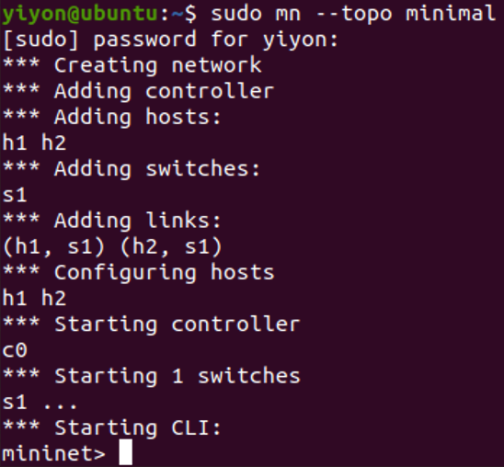
|  |  |
| --- | --- |
| 参数名称 | 功能 |
| exit | 退出Mininet |
| sudo mn -c | 清除Mininet缓存 |

我们将运用以上部分参数和指令进行实验。

## 实验过程

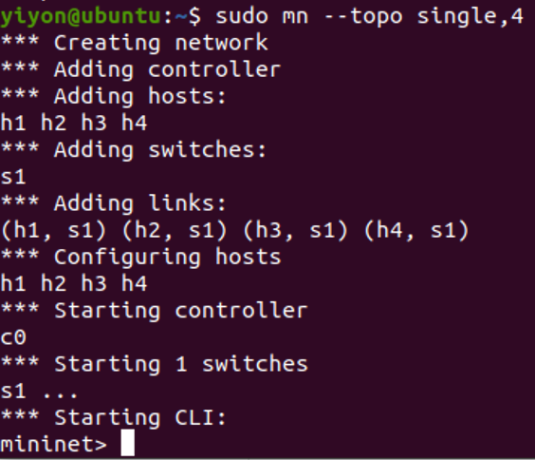
### 实验2.1 命令行创建拓扑

输入指令sudo mn --topo minimal，创建默认的最小拓扑：

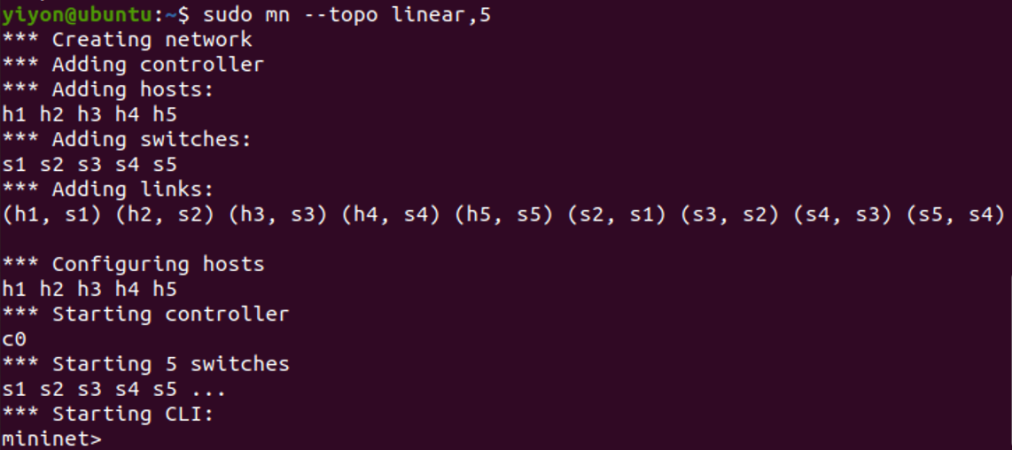


此拓扑只有一个交换机，并连接两个主机。

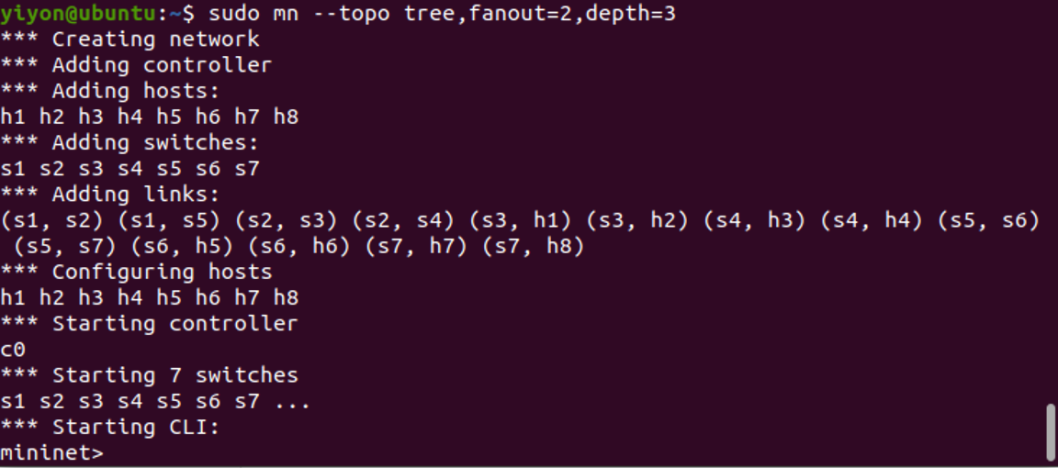
输入指令sudo mn --topo single,4，由于此类拓扑只有一个交换机，所以如果参数是4，代表此拓扑中有4台主机：



输入指令sudo mn --topo linear,5创建线性拓扑，由于线性拓扑的每一台交换机只连接一台主机，所以如果参数是5，代表此拓扑中有5台交换机和5台主机：



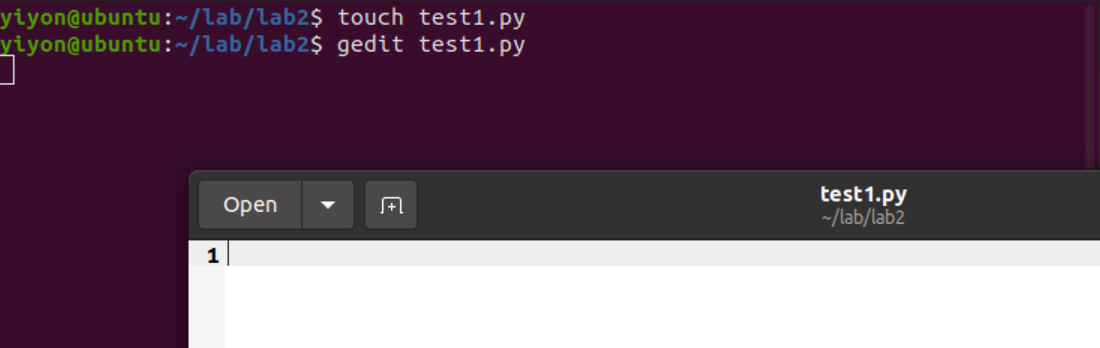
输入指令sudo mn --topo tree,fanout=2,depth=3创建树形拓扑，fanout=2代表扇出为2，depth=3代表深度为3：



### 实验2.2 python脚本文件创建拓扑

输入指令touch test1.py创建python脚本文件；

输入指令gedit test1.py编辑python脚本文件：



编写代码文件，构造一个线性拓扑，每一行的注释如下：

from mininet.net import Mininet

from mininet.topo import LinearTopo

# 导入Mininet和LinearTopo模块

Linear4 = LinearTopo(k=4) # 四个交换机，分别连一个主机

net = Mininet(topo=Linear4) # 创建包含四个交换机的线性拓扑

net.start() # 启动Mininet网络

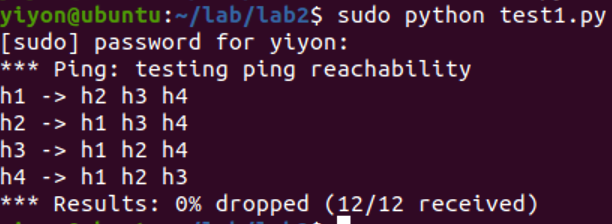
net.pingAll() # 对所有主机进行ping测试

net.stop() # 关闭Mininet网络

保存后，输入指令chmod +x test1.py为其增加修改权限，将它修改为可执行文件:



输入指令sudo python test1.py运行该脚本：



结果显示四台主机之间互通性良好，没有丢包。

我们继续构造一个single类的拓扑，每台交换机下连接三台主机，代码如下：

from mininet.net import Mininet

from mininet.topo import SingleSwitchTopo

# 导入Mininet和LinearTopo模块

Single3 = SingleSwitchTopo(k=3) # 1台交换机下连接3台主机

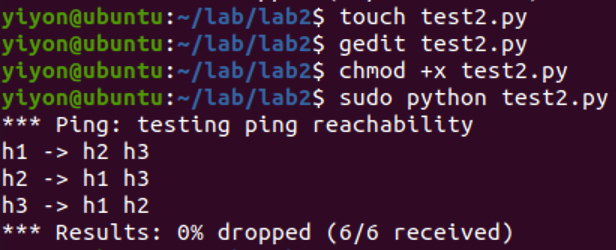
net = Mininet(topo=Single3) # 创建包含3台交换机的single类拓扑

net.start() # 启动Mininet网络

net.pingAll() # 对所有主机进行ping测试

net.stop() # 关闭Mininet网络

赋权后，输入指令sudo python test2.py运行测试：



同理进行树形拓扑的构造：

from mininet.net import Mininet

from mininet.topolib import TreeTopo

# 导入Mininet和TreeTopo模块

Tree22 = TreeTopo(depth=2,fanout=2) # 深度为2，扇出为2

net = Mininet(topo=Tree22) # 创建深2扇2的树形拓扑

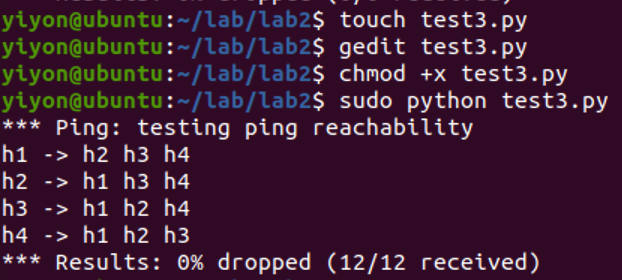
net.start() # 启动Mininet网络

net.pingAll() # 对所有主机进行ping测试

net.stop() # 关闭Mininet网络

注意，深度为2代表“树枝”（也就是交换机）的层数，扇出为2表示“结点”（也就是每个分叉处的交换机）要连接两台主机。

测试如下：



接下来我们尝试创建非常见类型的拓扑，例如：创建一个含有1台交换机、2台主机的网络拓扑，并给主机赋上IP地址：

from mininet.net import Mininet

# 导入Mininet模块

net = Mininet()

# 创建网络节点

c0 = net.addController()

h0 = net.addHost('h0')

s0 = net.addSwitch('s0')

h1 = net.addHost('h1')

# 创建两节点之间的连接

net.addLink(h0, s0)

net.addLink(h1, s0)

# 在接口处配置ip地址

h0.setIP('192.168.1.1', 24)

h1.setIP('192.168.1.2', 24)

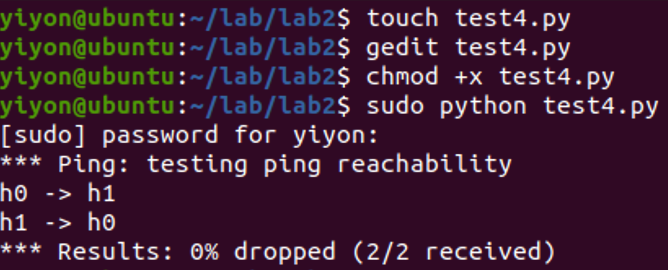
# 启动、ping测试、停止

net.start()

net.pingAll()

net.stop()

运行测试结果如下：



除了用python编写脚本创建拓扑以外，还可以通过脚本限制拓扑的功能，比如使用addHost()函数设置主机的cpu（百分数的形式）；使用addLink()函数设置带宽bw、延迟delay、最大队列的大小max\_queue\_size和损耗率loss等。

具体实现如下：

from mininet.net import Mininet

from mininet.node import CPULimitedHost

from mininet.link import TCLink

net = Mininet(host=CPULimitedHost, link=TCLink)

# 创建网络节点

c0 = net.addController()

s0 = net.addSwitch('s0')

h0 = net.addHost('h0')

h1 = net.addHost('h1', cpu=0.5)

h2 = net.addHost('h1', cpu=0.5)

# 主机h1和h2的CPU被限制为50%，即最大只能占用机器一半的CPU资源

net.addLink(s0, h0, bw=10, delay='5ms',

max\_queue\_size=1000, loss=10, use\_htb=True)

# 为交换机s0和主机h0添加一条链路，并设置参数

net.addLink(s0, h1)

net.addLink(s0, h2)

net.start()

net.pingAll()

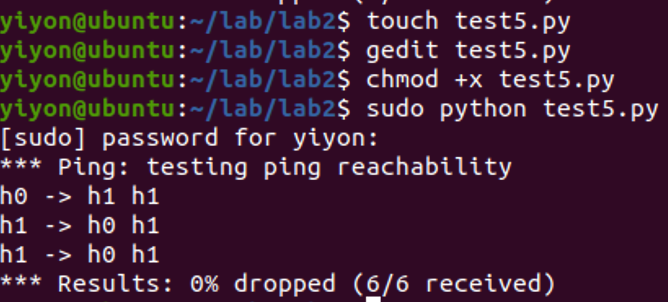
net.stop()

其中，进行限制的参数说明如下：

bw=10：表示链路的带宽为10 Mbps；delay='5ms'：表示链路的时延为5毫秒；m\_q\_s=1000：表示链路缓冲队列的最大大小为1000个数据包；loss=10：表示链路的数据包丢失率为10%。use\_htb=True：表示对链路应用Hierarchical Token Bucket调度算法，将链路带宽分配为一组子队列。HTB算法是一个带宽管理工具，可以通过配置不同数据流量的规则，将带宽分配和管理在网络设备的输出端口上。

通过调整这些参数，可以控制链路的性能，从而影响网络数据传输的效率和可靠性。

测试结果如下：



1. 实验参考文件

