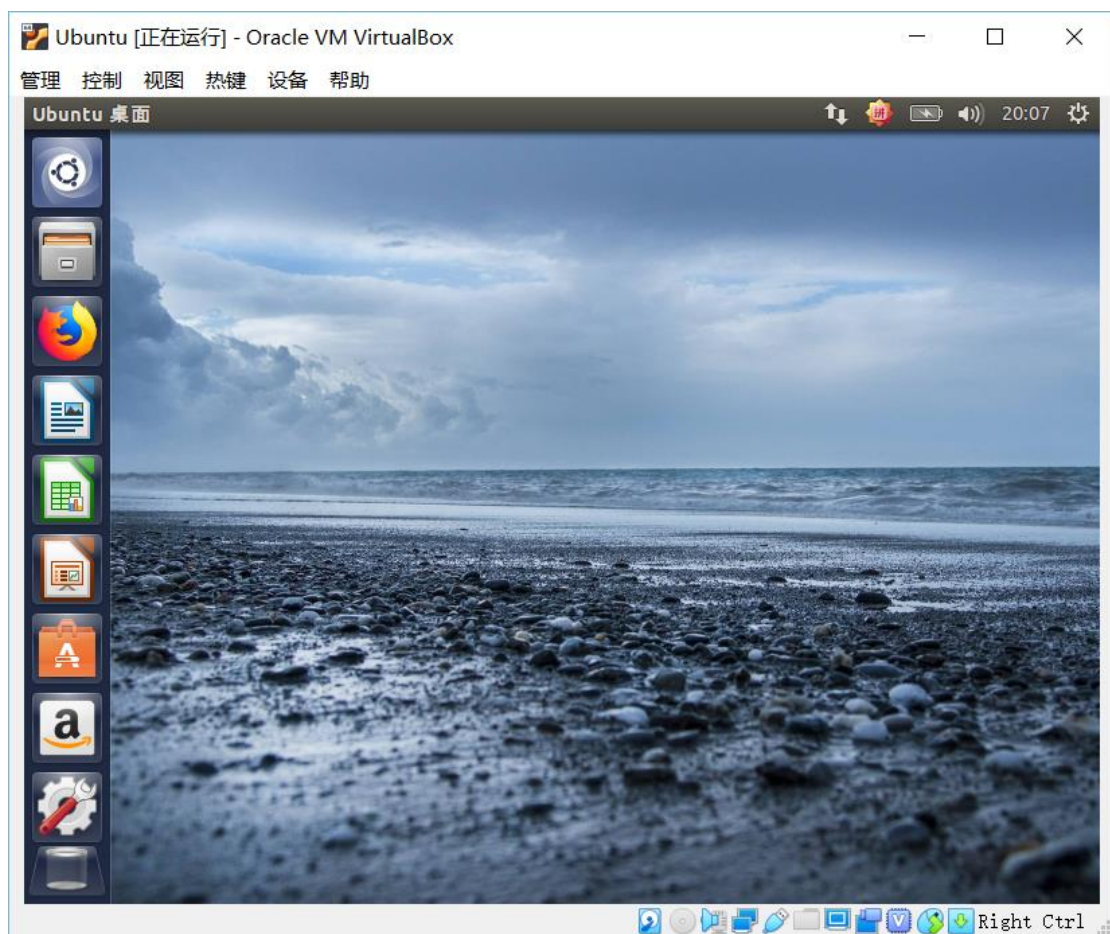


一、安装和配置 Linux 虚拟机

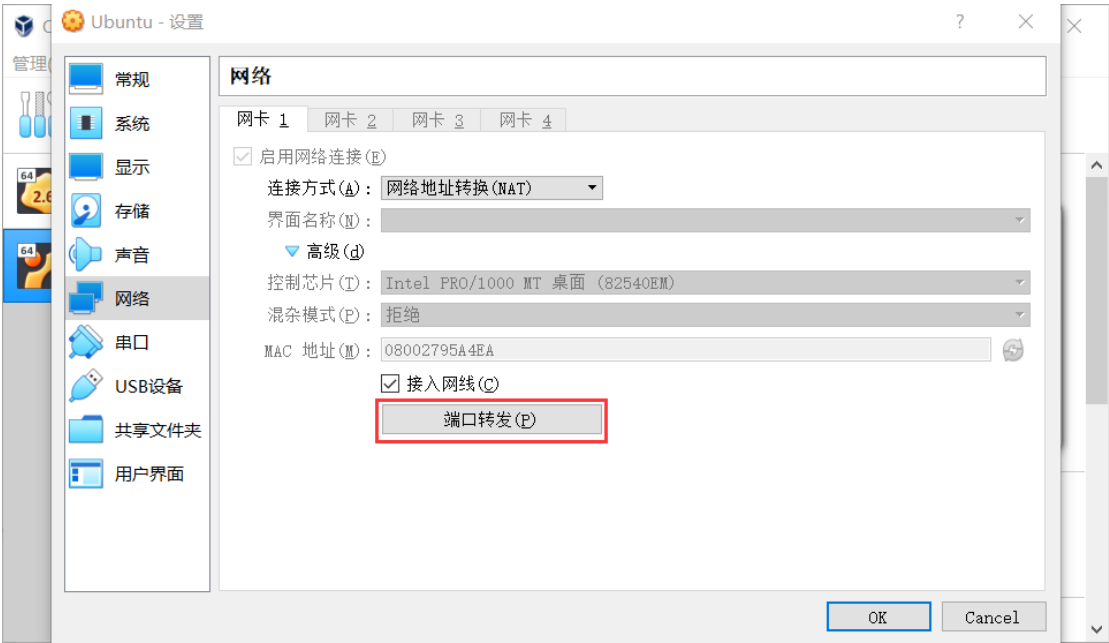
虚拟机安装比较简单，推荐使用 **Ubuntu** 镜像文件 ([下载地址](#))，界面比较 Friendly。

安装完成后的样子：

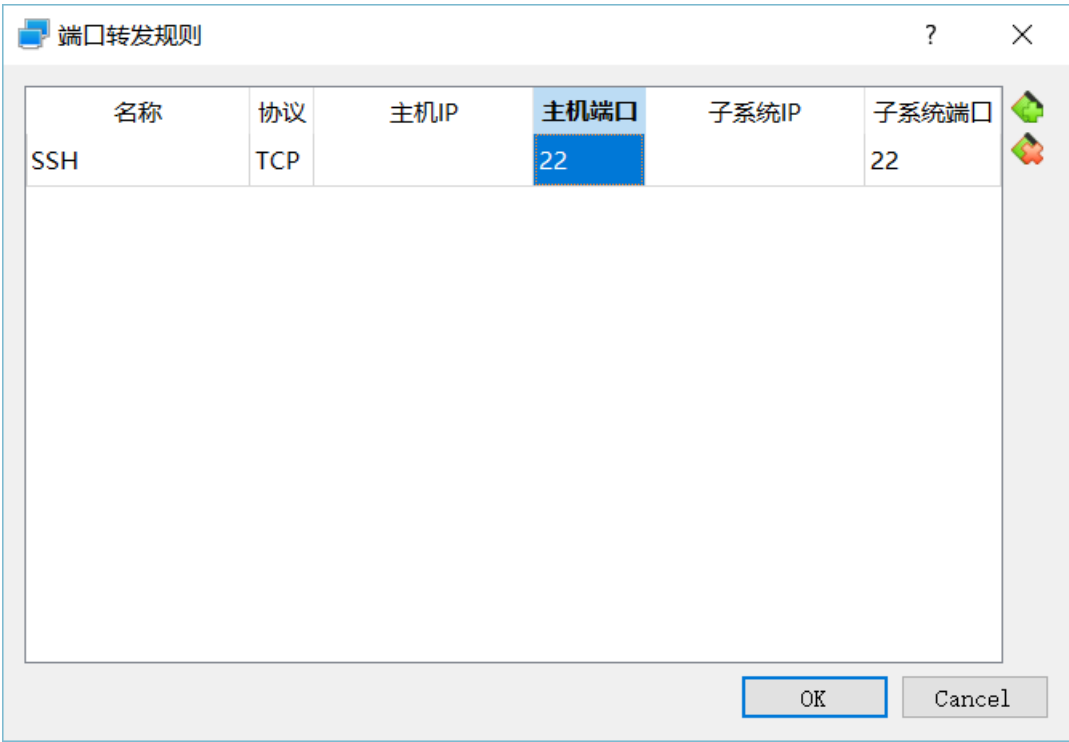


二、设置本地和虚拟机的 SSH 连接

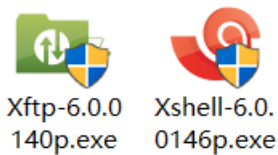
2.1 设置虚拟机的网络：



端口就默认为 22，不用改了



然后启动 XShell，一个终端模拟软件，功能非常强大，和 Xftp 配套使用。



，XShell 用来连接和交互，Xftp 用来传输文件。

接着拟机中打开终端（快捷键为 **Ctrl+Alt+T**），输入“**sudo apt-get update**”

然后输入当前账户的用户名的密码（就是让你用 **root** 账户登录）。

```
root@cherry-localhost: ~
cherry@cherry-localhost:~$ sudo apt-get install openssh-server
[sudo] password for cherry:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会安装下列额外的软件包：
libck-connector0 ncurses-term openssh-sftp-server ssh-import-id
建议安装的软件包：
rssh molly-guard monkeysphere
下列【新】软件包将被安装：
libck-connector0 ncurses-term openssh-server openssh-sftp-server
ssh-import-id
升级了 0 个软件包，新安装了 5 个软件包，要卸载 0 个软件包，有 64 个软件包未被升级。
需要下载 620 kB 的软件包。
解压缩后会消耗掉 3,483 kB 的额外空间。
您希望继续执行吗？ [Y/n] y
【警告】：下列软件包不能通过验证！
openssh-sftp-server openssh-server
不经验证就安装这些软件包吗？ [y/N] y
获取：1 http://cn.archive.ubuntu.com/ubuntu/ trusty/main libck-connector0 i386 0.4.5-3.1ubuntu2 [10.1 kB]
获取：2 http://cn.archive.ubuntu.com/ubuntu/ trusty/main ncurses-term all 5.9+20140118-1ubuntu1 [243 kB]
```

```
root@cherry-localhost: ~
正在选中未选择的软件包 openssh-sftp-server。
正准备解包 .../openssh-sftp-server_1%3a6.6p1-2ubuntu2.13_i386.deb ...
正在解包 openssh-sftp-server (1:6.6p1-2ubuntu2.13) ...
正在选中未选择的软件包 openssh-server。
正准备解包 .../openssh-server_1%3a6.6p1-2ubuntu2.13_i386.deb ...
正在解包 openssh-server (1:6.6p1-2ubuntu2.13) ...
正在选中未选择的软件包 ssh-import-id。
正准备解包 .../ssh-import-id_3.21-0ubuntu1_all.deb ...
正在解包 ssh-import-id (3.21-0ubuntu1) ...
正在处理用于 man-db (2.6.7.1-1ubuntu1) 的触发器 ...
正在处理用于 ureadahead (0.100.0-16) 的触发器 ...
ureadahead will be reprofiled on next reboot
正在处理用于 ufw (0.34~rc-0ubuntu2) 的触发器 ...
正在设置 libck-connector0:i386 (0.4.5-3.1ubuntu2) ...
正在设置 ncurses-term (5.9+20140118-1ubuntu1) ...
正在设置 openssh-sftp-server (1:6.6p1-2ubuntu2.13) ...
正在设置 openssh-server (1:6.6p1-2ubuntu2.13) ...
Creating SSH2 RSA key; this may take some time ...
Creating SSH2 DSA key; this may take some time ...
Creating SSH2 ECDSA key; this may take some time ...
Creating SSH2 ED25519 key; this may take some time ...
ssh start/running, process 3315
正在设置 ssh-import-id (3.21-0ubuntu1) ...
正在处理用于 libc-bin (2.19-0ubuntu6.14) 的触发器 ...
```

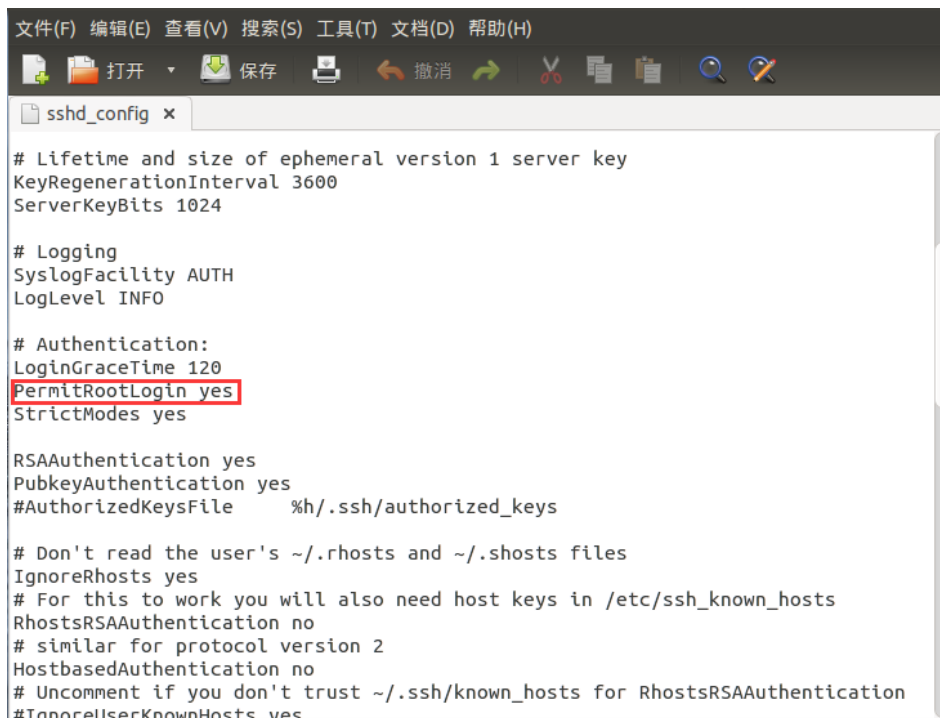
2.2 安装和设置 SSH 服务

在 "终端窗口" 输入 `sudo apt-get install openssh-server`，然后输入 `y`，安装完成。

在 "终端窗口" 输入 `service ssh status` 观察 `ssh` 是否启动。如下状态说明已经启动：或者：打开 "终端窗口"，输入 `sudo ps -e | grep ssh` --> 回车 --> 有 `sshd`，说明 `ssh` 服务已经启动，如果没有启动，输入 `sudo service ssh start` --> 回车 --> `ssh` 服务就会启动

2.3 修改 SSH 设置

打开 "终端窗口"，输入 `sudo gedit /etc/ssh/sshd_config` --> 回车 --> 把配置文件中的找到 `#PasswordAuthentication no` 的注释 `#` 去掉，并且将 `NO` 修改为 `YES` --> 找到 `#PermitRootLogin prohibit-password` 修改为 `PermitRootLogin yes` 保存，修改成功。即修改为如下状态：



```
文件(F) 编辑(E) 查看(V) 搜索(S) 工具(T) 文档(D) 帮助(H)
打开 保存 撤销 剪切 复制 粘贴 查找 替换

sshd_config x

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 1024

# Logging
SyslogFacility AUTH
LogLevel INFO

# Authentication:
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile      %h/.ssh/authorized_keys

# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
#IgnoreUserKnownHosts yes
```

```
sshd_config x
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
#IgnoreUserKnownHosts yes

# To enable empty passwords, change to yes (NOT RECOMMENDED)
PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication no

# Change to no to disable tunnelled clear text passwords
PasswordAuthentication yes

# Kerberos options
#KerberosAuthentication no
#KerberosGetAFSToken no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes

# GSSAPI options
#GSSAPIAuthentication no
```

保存后退出，然后输入 `service ssh restart` 重启服务，这样就设置成功了。

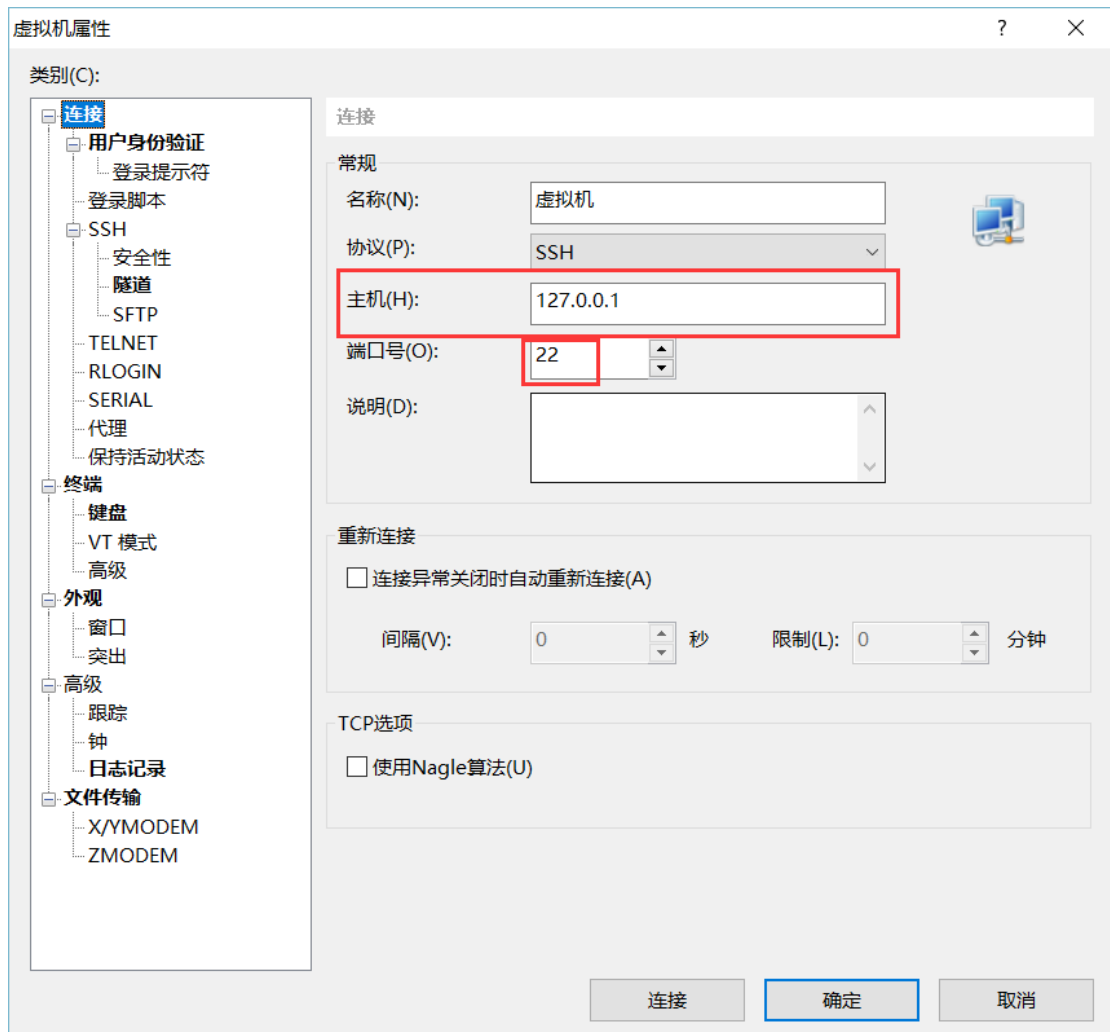
2.4 设置 SSH 连接

打开 XShell，新建会话，在虚拟机中输入 `ifconfig` 查看 IP，然后在主机名中输入 IP 和端口号：

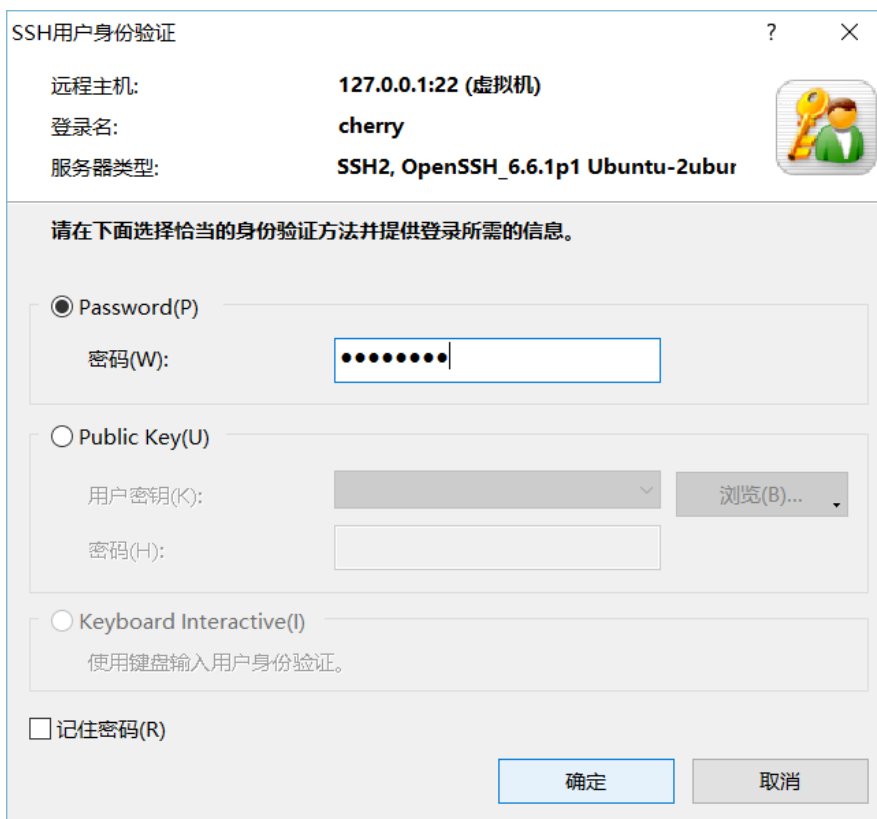
```
root@cherry-localhost: ~
root@cherry-localhost:~# sudo gedit /etc/ssh/sshd_config
(gedit:3665): IBUS-WARNING **: The owner of /home/cherry/.config/ibus/bus is not root!
root@cherry-localhost:~# ifconfig
eth0      Link encap:以太网  硬件地址 08:00:27:95:a4:ea
          inet 地址:10.0.2.15 广播:10.0.2.255 掩码:255.255.255.0
          inet6 地址: fe80::a00:27ff:fe95:a4ea/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:874 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:550 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:680389 (680.3 KB) 发送字节:61312 (61.3 KB)

lo        Link encap:本地环回
          inet 地址:127.0.0.1 掩码:255.0.0.0
          inet6 地址: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536 跃点数:1
          接收数据包:330 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:330 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1
          接收字节:25246 (25.2 KB) 发送字节:25246 (25.2 KB)

root@cherry-localhost:~#
```



点击确定，输入虚拟机的用户名和密码，连接成功。




```

Connecting to 127.0.0.1:22...
Connection established.
To escape to local shell, press 'Ctrl+Alt+]'.

Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 4.4.0-142-generic i686)

 * Documentation:  https://help.ubuntu.com/
New release '16.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

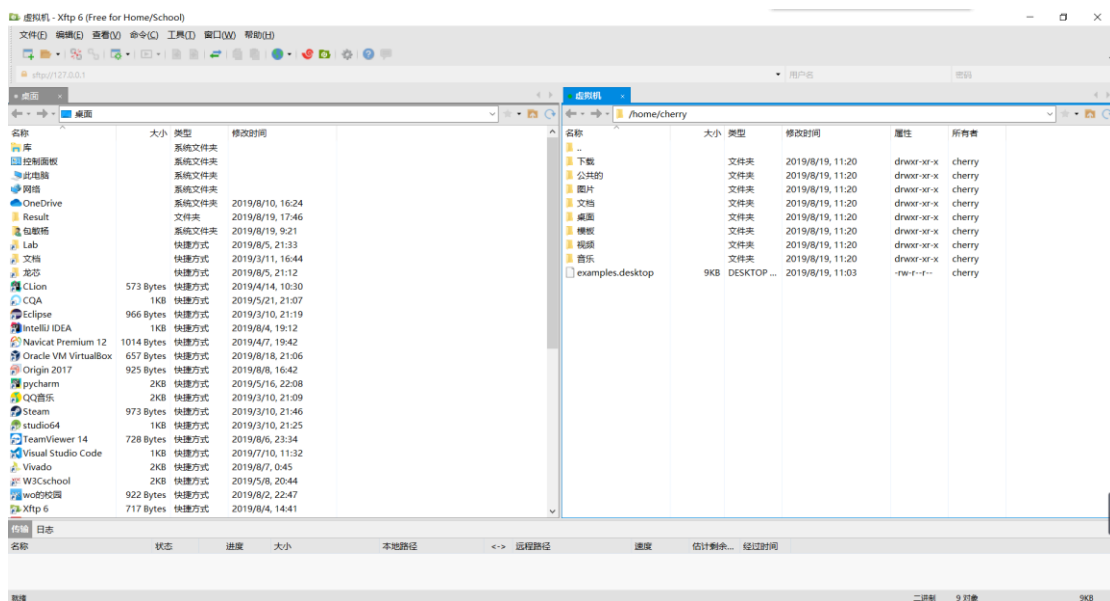
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

cherry@cherry-localhost:~$

```

同理，连接完成的 Xftp 如图所示：



如果发生传送失败，记得在虚拟机中修改文件夹权限

在终端键入：**sudo chmod 777 /opt**

其中 **chmod 777** 表示文件的 **r,w,x**（读、写、执行权限），如果给予全部三个权限，这三个值便全是 **1**，那么 **111** 用十进制表示就是 **7**。

设置完之后应该就可以正常传送文件了。

三、GNU 工具链的安装

使用的是龙芯公司的 `gcc-4.3-ls232.tar.gz`,

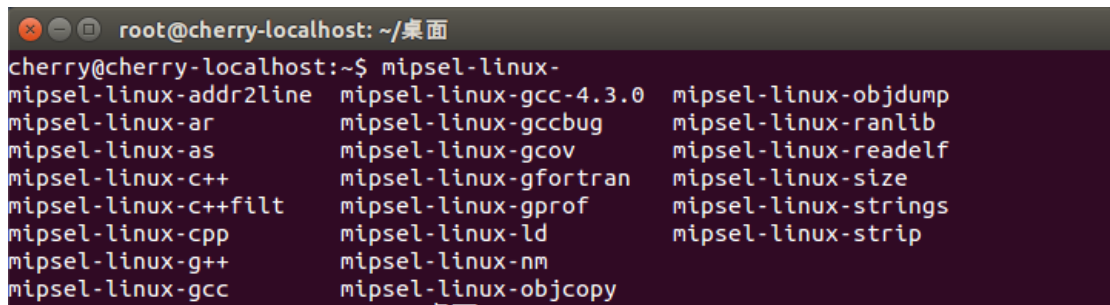
下载地址: (<http://ftp.loongnix.org/toolchain/gcc/release/gcc-4.3-ls232.tar.gz>)

我们开始配置 `gcc`

进入模拟器 `/opt` 文件夹, 在终端输入 `tar -zxvf gcc-4.3-ls232.tar.gz`,
压缩完成。在 `home` 目录下查看隐藏文件 `.bashrc`, 在文件最后一行加上

```
export PATH="$PATH:password/opt/gcc-4.3-ls232/bin"
```

再次在终端输入: `mipsel-linux-`, 并按两次 `TAB`, 如图, 表示系统路径设置正确



```
root@cherry-localhost: ~/桌面
cherry@cherry-localhost:~$ mipsel-linux-
mipsel-linux-addr2line  mipsel-linux-gcc-4.3.0  mipsel-linux-objdump
mipsel-linux-ar         mipsel-linux-gccbug   mipsel-linux-ranlib
mipsel-linux-as         mipsel-linux-gcov    mipsel-linux-readelf
mipsel-linux-c++        mipsel-linux-gfortran mipsel-linux-size
mipsel-linux-c++filt    mipsel-linux-gprof   mipsel-linux-strings
mipsel-linux-cpp        mipsel-linux-ld      mipsel-linux-strip
mipsel-linux-g++        mipsel-linux-nm
mipsel-linux-gcc        mipsel-linux-objcopy
```

再次键入: `mipsel-linux-gcc -v`, 如图: 表示 `GCC` 工具链安装完成

```
cherry@cherry-localhost: ~  
cherry@cherry-localhost:~$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/opt/gcc-4.3-ls232/bin  
cherry@cherry-localhost:~$ mipsel-linux-gcc -v  
Using built-in specs.  
Target: mipsel-linux  
Configured with: ../gcc-4.3.0/configure --prefix=/opt/gcc-4.3-ls232 --host=i486-pc-linux-gnu --build=i486-pc-linux-gnu --target=mipsel-linux --host=i486-pc-linux-gnu --with-sysroot=/opt/gcc-4.3-ls232/sysroot --with-abi=32 --disable-nls --enable-shared --disable-multilib --enable-__cxa_atexit --enable-c99 --enable-long-long --enable-threads=posix --enable-languages=c,c++,fortran  
Thread model: posix  
gcc version 4.3.0 (GCC)  
cherry@cherry-localhost:~$
```

至此，一些基本的准备工作就完成了，但是在此过程中会因为不熟悉 Linux 系统而导致一系列问题，在此总结一下。

四、在此过程中遇到的问题及解释

4.1 关于解压缩命令

tar

- c: 建立压缩档案
- x: 解压
- t: 查看内容
- r: 向压缩归档文件末尾追加文件
- u: 更新原压缩包中的文件

这五个是独立的命令，压缩解压都要用到其中一个，可以和别的命令连用但只能用其中一个。下面的参数是根据需要在压缩或解压档案时可选的。

- z: 有 gzip 属性的
- j: 有 bz2 属性的
- Z: 有 compress 属性的
- v: 显示所有过程
- O: 将文件解开到标准输出

下面的参数-f 是必须的

-f: 使用档案名字，切记，这个参数是最后一个参数，后面只能接档案名。

```
# tar -cf all.tar *.jpg
```

这条命令是将所有.jpg 的文件打成一个名为 all.tar 的包。-c 是表示产生新的包，-f 指定包的文件名。

```
# tar -rf all.tar *.gif
```

这条命令是将所有.gif 的文件增加到 all.tar 的包里面去。-r 是表示增加文件的意思。

```
# tar -uf all.tar logo.gif
```

这条命令是更新原来 tar 包 all.tar 中 logo.gif 文件，-u 是表示更新文件的意思。

```
# tar -tf all.tar
```

这条命令是列出 all.tar 包中所有文件，-t 是列出文件的意思

```
# tar -xf all.tar
```

这条命令是解出 all.tar 包中所有文件，-t 是解开的意思

压缩

```
tar -cvf jpg.tar *.jpg //将目录里所有 jpg 文件打包成 tar.jpg
```

```
tar -czf jpg.tar.gz *.jpg //将目录里所有 jpg 文件打包成 jpg.tar 后，并且将其用 gzip 压缩，生成一个 gzip 压缩过的包，命名为 jpg.tar.gz
```

```
tar -cjf jpg.tar.bz2 *.jpg //将目录里所有 jpg 文件打包成 jpg.tar 后，并且将其用 bzip2 压缩，生成一个 bzip2 压缩过的包，命名为 jpg.tar.bz2
```

```
tar -cZf jpg.tar.Z *.jpg //将目录里所有 jpg 文件打包成 jpg.tar 后，并且将其用 compress 压缩，生成一个 umcompress 压缩过的包，命名为 jpg.tar.Z
```

```
rar a jpg.rar *.jpg //rar 格式的压缩，需要先下载 rar for linux
```

```
zip jpg.zip *.jpg //zip 格式的压缩，需要先下载 zip for linux
```

解压

`tar -xvf file.tar` //解压 tar 包

`tar -xzvf file.tar.gz` //解压 tar.gz

`tar -xjvf file.tar.bz2` //解压 tar.bz2

`tar -xZvf file.tar.Z` //解压 tar.Z

`unrar e file.rar` //解压 rar

`unzip file.zip` //解压 zip

总结

- 1、*.tar 用 `tar -xvf` 解压
- 2、*.gz 用 `gzip -d` 或者 `gunzip` 解压
- 3、*.tar.gz 和 *.tgz 用 `tar -xzf` 解压
- 4、*.bz2 用 `bzip2 -d` 或者用 `bunzip2` 解压
- 5、*.tar.bz2 用 `tar -xjf` 解压
- 6、*.Z 用 `uncompress` 解压
- 7、*.tar.Z 用 `tar -xZf` 解压
- 8、*.rar 用 `unrar e` 解压
- 9、*.zip 用 `unzip` 解压

参考：（<https://www.cnblogs.com/rickzhai/p/6432240.html>）

要注意压缩包是 **tar** 类型还是 **gz** 类型

4.2 关于环境变量

类似于 **Windows** 下的环境变量，本质含义就是让系统运行一个程序时，不仅仅要在当前目录下面寻找，还要到 **PATH** 指定的路径下寻找，此操作实际简化了程序执行的复杂度，提高了效率。

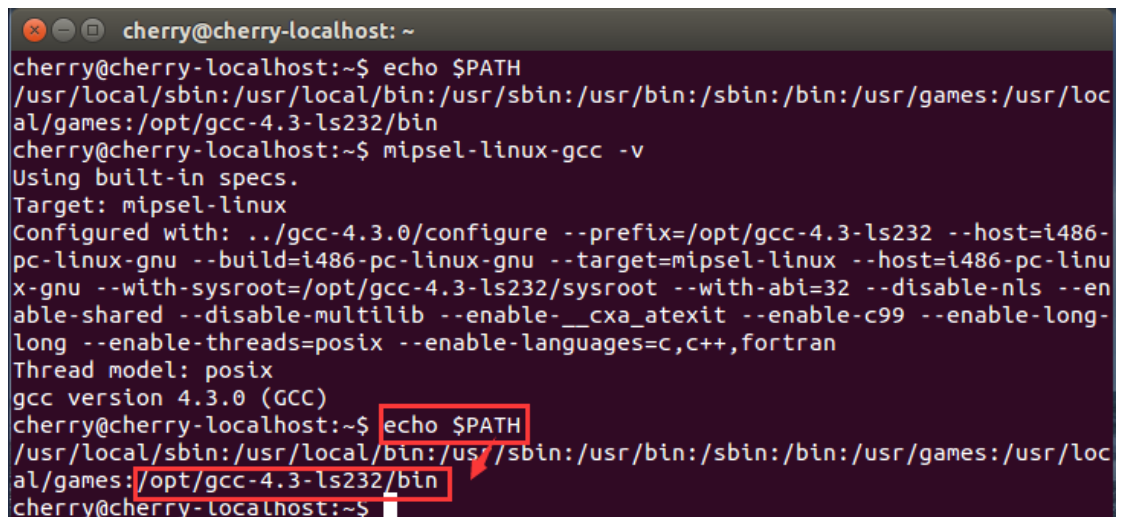
以上面的 **GCC** 路径为例，我们写入的 **PATH** 路径为：

```
export PATH="$PATH:/opt/gcc-4.3-ls232/bin"
```

什么意思？可以把 **PATH** 看成一个字符串，用 “/”、“:”、“\$” 等符号进行划分，系统根据划分的字符串寻找目标地址。

因为 **PATH** 是一个字符串，用 “\$PATH” 表示引用这个字符串，就是表示 **PATH** 所指的内容；“:” 是 **Linux** 中分隔符，相当于 **Windows** 中的 “;”，因此该语句表示在原有的 **PATH** 路径下再加上 **/opt/gcc-4.3-ls232/bin** 路径，我们在终端中检验一下：

输入 **echo \$PATH**，结果如图：（**echo** 是回显命令，是常见的脚本命令）

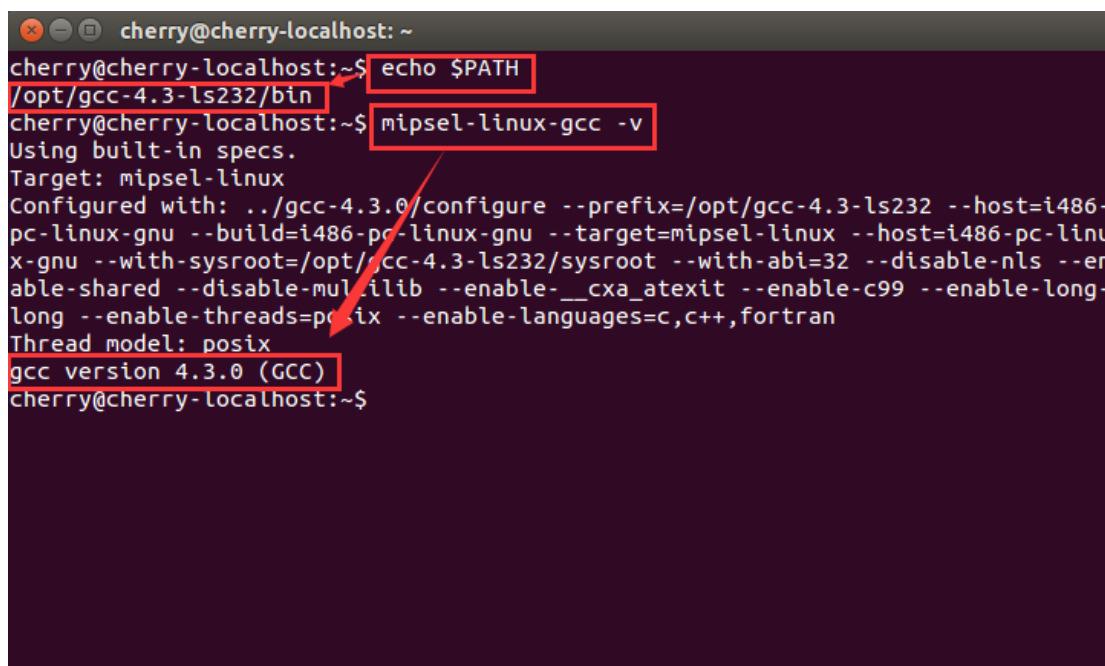


```
cherry@cherry-localhost: ~
cherry@cherry-localhost:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/opt/gcc-4.3-ls232/bin
cherry@cherry-localhost:~$ mipsel-linux-gcc -v
Using built-in specs.
Target: mipsel-linux
Configured with: ../gcc-4.3.0/configure --prefix=/opt/gcc-4.3-ls232 --host=i486-pc-linux-gnu --build=i486-pc-linux-gnu --target=mipsel-linux --host=i486-pc-linux-gnu --with-sysroot=/opt/gcc-4.3-ls232/sysroot --with-abi=32 --disable-nls --enable-shared --disable-multilib --enable-__cxa_atexit --enable-c99 --enable-long-long --enable-threads=posix --enable-languages=c,c++,fortran
Thread model: posix
gcc version 4.3.0 (GCC)
cherry@cherry-localhost:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/opt/gcc-4.3-ls232/bin
cherry@cherry-localhost:~$
```

实际上，若要仅仅配置 **GCC**，不需要加上之前的 **PATH** 路径，因此只要这样写即可：

```
export PATH="/opt/gcc-4.3-ls232/bin"
```

我们在终端中再次输入 **echo \$PATH**，结果证明是正确的，**gcc** 安装无误：



```
cherry@cherry-localhost: ~  
cherry@cherry-localhost:~$ echo $PATH  
/opt/gcc-4.3-ls232/bin  
cherry@cherry-localhost:~$ mipsel-linux-gcc -v  
Using built-in specs.  
Target: mipsel-linux  
Configured with: ../gcc-4.3.0/configure --prefix=/opt/gcc-4.3-ls232 --host=i486-  
pc-linux-gnu --build=i486-pc-linux-gnu --target=mipsel-linux --host=i486-pc-linu  
x-gnu --with-sysroot=/opt/gcc-4.3-ls232/sysroot --with-abi=32 --disable-nls --en  
able-shared --disable-multilib --enable-__cxa_atexit --enable-c99 --enable-long-  
long --enable-threads=posix --enable-languages=c,c++,fortran  
Thread model: posix  
gcc version 4.3.0 (GCC)  
cherry@cherry-localhost:~$
```

PATH 环境变量的用处还有很多，比如有人喜欢通过命令行来执行程序，然后就不用进入文件夹执行了，其实利用 **PATH** 路径也是非常简单的，我在 **Windows** 环境下简单演示一遍：

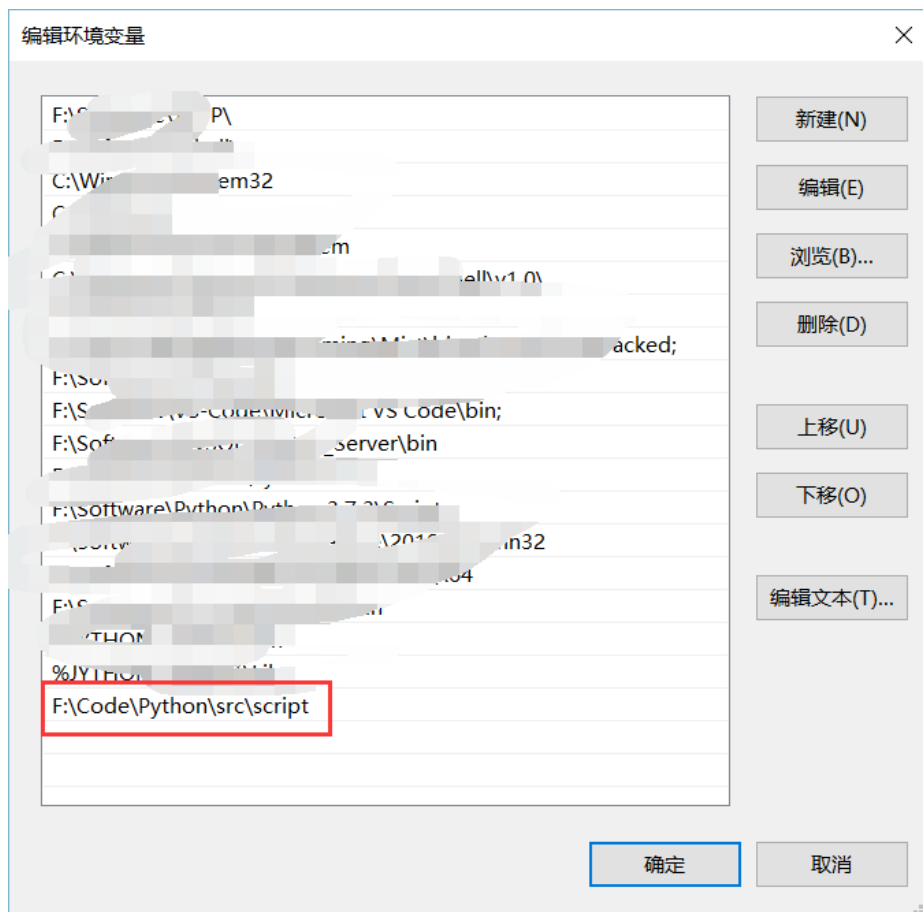
若我想运行 **CSDN_Visit.py**（一个 **python** 脚本文件），在命令行里输入 **CSDN_Visit.py**



```
管理员: 命令提示符  
Microsoft Windows [版本 10.0.17134.950]  
(c) 2018 Microsoft Corporation。保留所有权利。  
C:\WINDOWS\system32>CSDN_Visit.py  
'CSDN_Visit.py' 不是内部或外部命令，也不是可运行的程序  
或批处理文件。
```

显然没有用，这时我把该文件放到 F 盘 F:\Code\Python\src\script 目录下：

在系统变量中加入 F:\Code\Python\src\script



然后再次启动 cmd，输入 CSDN_Visit.py 我们看到脚本已经成功执行

```
管理员: 命令提示符 - CSDN_Visit.py
Microsoft Windows [版本 10.0.17134.950]
(c) 2018 Microsoft Corporation. 保留所有权利。

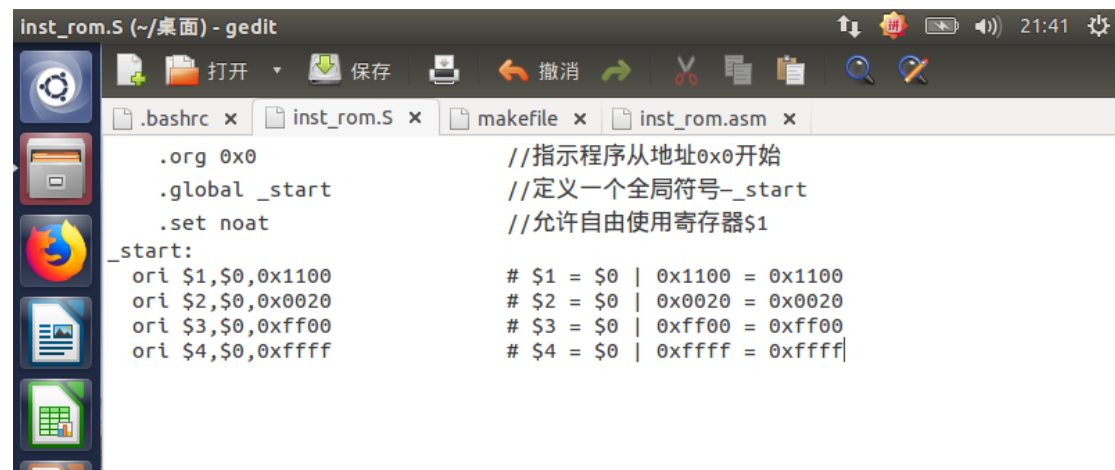
C:\WINDOWS\system32>CSDN_Visit.py
<!DOCTYPE html>
<html lang="zh-CN">
<head>
  <meta charset="UTF-8">
  <link rel="canonical" href="https://blog.csdn.net/gq_42650988"/>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <meta name="renderer" content="webkit"/>
  <meta name="force-rendering" content="webkit"/>
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>
  <meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0, user-scalable=no">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">
  <meta name="report" content="{ 'pid': 'blog' }">
  <meta name="referrer" content="always">
  <meta http-equiv="Cache-Control" content="no-siteapp" />
  <link rel="alternate" media="handheld" href="#" />
  <meta name="shenma-site-verification" content="5a59773ab8077d4a62bf469ab966a63b_1497598843">
  <meta name="csdn-baidu-search" content="{ 'autorun': true, 'install': true, 'keyword': '【忆往昔峥嵘岁月稠】计算机小白' }">
  <link href="https://csdnimg.cn/public/favicon.ico" rel="SHORTCUT ICON">
  <title>【忆往昔峥嵘岁月稠】计算机小白 - CSDN博客</title>

  <link rel="stylesheet" href="https://csdnimg.cn/release/phenix/template/css/list-9d332b1c58.min.css">
</head>
```


五、编译汇编指令

5.1 编译

我们先编写一组汇编指令，命名为 `inst_rom.S`



```
inst_rom.S (~桌面) - gedit
.org 0x0                                //指示程序从地址0x0开始
.global _start                          //定义一个全局符号_start
.set noat                               //允许自由使用寄存器$1
_start:
ori $1,$0,0x1100                        # $1 = $0 | 0x1100 = 0x1100
ori $2,$0,0x0020                        # $2 = $0 | 0x0020 = 0x0020
ori $3,$0,0xffff00                     # $3 = $0 | 0xffff00 = 0xffff00
ori $4,$0,0xffff                       # $4 = $0 | 0xffff = 0xffff
```

然后我们在终端中输入指令：

```
mipsel-linux-as -mips32 inst_rom.S -o inst_rom.o
```

表示用 `as` 工具将 `inst_rom.S` 文件编译成 `inst_rom.o` 文件

再输入：

```
mipsel-linux-ld -T ram.ld inst_rom.o -o inst_rom.om
```

表示将 `inst_rom.o` 文件链接成 `inst_rom.om` 文件

再输入：

```
mipsel-linux-objcopy -O binary inst_rom.om inst_rom.bin
```

根据 `.om` 文件得到了 `bin` 文件

再输入：

```
mipsel-linux-objdump -D inst_rom.om > inst_rom.asm
```

对汇编指令进行反汇编，得到与机器指令对应的二进制字

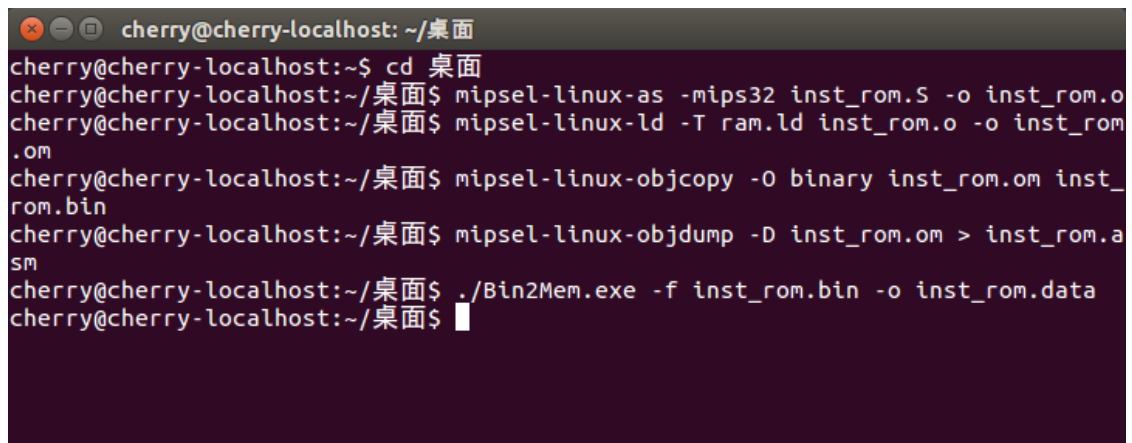
最后将 **Bin2MEM.exe** 拷贝到与 **inst_rom** 系列文件相同的目录下，执行

```
./Bin2Mem.exe -f inst_rom.bin -o inst_rom.data
```

第一次执行应该需要修改 **Bin2Mem.exe** 文件的权限，输入：

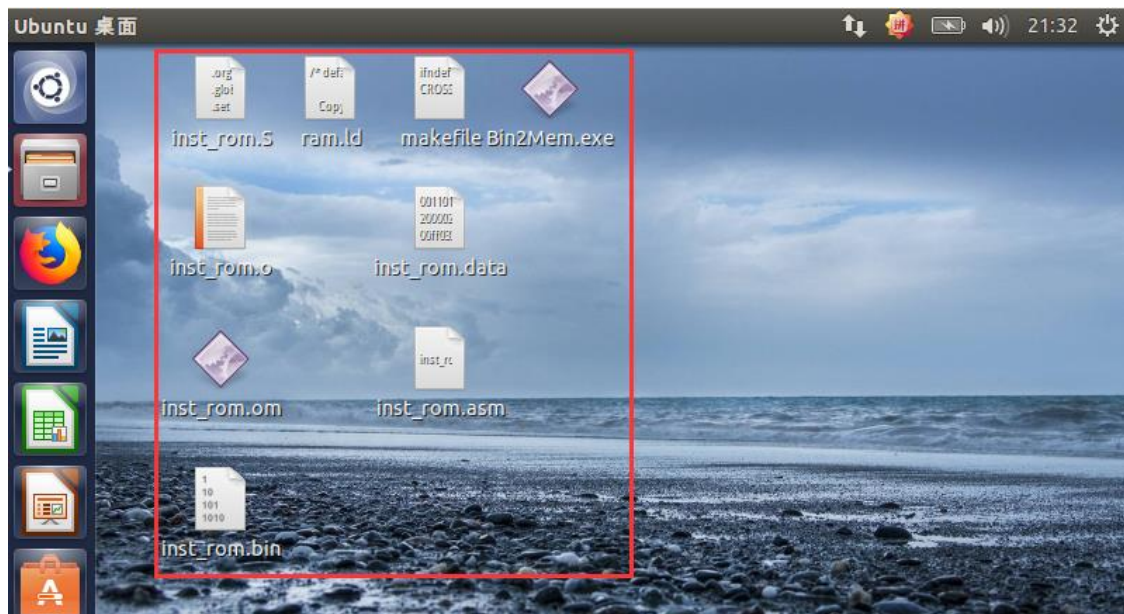
```
Chmod 777 Bin2Mem.exe
```

最终转化为与 **Vivado** 程序中读入的文件类型 (**.data**)，执行结果如图：



```
cherry@cherry-localhost: ~/桌面
cherry@cherry-localhost:~$ cd 桌面
cherry@cherry-localhost:~/桌面$ mipsel-linux-as -mips32 inst_rom.S -o inst_rom.o
cherry@cherry-localhost:~/桌面$ mipsel-linux-ld -T ram.ld inst_rom.o -o inst_rom.om
cherry@cherry-localhost:~/桌面$ mipsel-linux-objcopy -O binary inst_rom.om inst_rom.bin
cherry@cherry-localhost:~/桌面$ mipsel-linux-objdump -D inst_rom.om > inst_rom.asm
cherry@cherry-localhost:~/桌面$ ./Bin2Mem.exe -f inst_rom.bin -o inst_rom.data
cherry@cherry-localhost:~/桌面$
```

生成的文件：



我们打开 **inst_rom.data** 文件查看：

```
cherry@cherry-localhost: ~/桌面
cherry@cherry-localhost:~$ cd 桌面
cherry@cherry-localhost:~/桌面$ mipsel-linux-as -mips32 inst_rom.S -o inst_rom.o
cherry@cherry-localhost:~/桌面$ mipsel-linux-ld -T ram.ld inst_rom.o -o inst_rom.om
cherry@cherry-localhost:~/桌面$ mipsel-linux-objcopy -O binary inst_rom.om inst_rom.bin
cherry@cherry-localhost:~/桌面$ mipsel-linux-objdump -D inst_rom.om > inst_rom.asm
cherry@cherry-localhost:~/桌面$ ./Bin2Mem.exe -f inst_rom.bin -o inst_rom.data
cherry@cherry-localhost:~/桌面$ cat inst_rom.data
00110134
20000234
00ff0334
ffff0434
1e000000
00000000
00000000
00000000
00000000
00000000
00000000
cherry@cherry-localhost:~/桌面$
```

这便是我们一开始那几条汇编指令的机器指令。

5.2 大小端地址

大端（存储）模式：是指一个数据的低位字节序的内容放在高地址处，高位字节序存的内容放在低地址处。

小端（存储）模式：是指一个数据的低位字节序内容存放在低地址处，高位字节序的内容存放在高地址处。（可以总结为“小小小”即低位、低地址、小端）

在上述的例子中，我们发现原来的机器指令 **34011100** 变成了 **00110134**，高位数据放在了地址的高位，是小端存储，这样看上去不是很方便，我们可以修改编译指令改变最终编译的结果，输入 **mipsel-linux-as -help** 查看帮助

```
cherry@cherry-localhost: ~/桌面
cherry@cherry-localhost:~/桌面$ mipsel-linux-as -help
Usage: mipsel-linux-as [option...] [asmfile...]
Options:
  -a[sub-option...]      turn on listings
                          Sub-options [default hls]:
                          c      omit false conditionals
                          d      omit debugging directives
                          g      include general info
                          h      include high-level source
                          l      include assembly
                          m      include macro expansions
                          n      omit forms processing
                          s      include symbols
                          =FILE list to FILE (must be last sub-option)
  --alternate             initially turn on alternate macro syntax
  -D                     produce assembler debugging messages
  --debug-prefix-map OLD=NEW Map OLD to NEW in debug information
  --defsym SYM=VAL       define symbol SYM to given value
  --em=[mipslelf | mipsbelf | mipslelf]
                          emulate output (default mipslelf)
  --execstack            require executable stack for this object
  --noexecstack          don't require executable stack for this object
  -f                     skip whitespace and comment preprocessing
  -g --gen-debug         generate debugging information
```

```
cherry@cherry-localhost: ~/桌面
@FILE read options from FILE
MIPS options:
-EB generate big endian output
-EL generate little endian output
-g, -g2 do not remove unneeded NOPs or swap branches
-G NUM allow referencing objects up to NUM bytes
        implicitly with the gp register [default 8]
-mips1 generate MIPS ISA I instructions
-mips2 generate MIPS ISA II instructions
-mips3 generate MIPS ISA III instructions
-mips4 generate MIPS ISA IV instructions
-mips5 generate MIPS ISA V instructions
-mips32 generate MIPS32 ISA instructions
-mips32r2 generate MIPS32 release 2 ISA instructions
-mips64 generate MIPS64 ISA instructions
-mips64r2 generate MIPS64 release 2 ISA instructions
-march=CPU/-mtune=CPU generate code/schedule for CPU, where CPU is one of:
                        mips1, mips2, mips3, mips4, mips5, mips32,
                        mips32r2, mips64, mips64r2, r3000, r2000, r3900,
                        r6000, r4000, r4010, vr4100, vr4111, vr4120,
                        vr4130, vr4181, vr4300, r4400, r4600, orion,
                        r4650, loongson2e, loongson2f, r8000, r10000,
                        r12000, vr5000, vr5400, vr5500, rm5200, rm5230,
                        rm5231, rm5261, rm5721, rm7000, rm9000, 4kc, 4km,
```

我们修改指令:

```
cherry@cherry-localhost: ~/桌面
cherry@cherry-localhost:~/桌面$ mipsel-linux-as -mips32 -EB inst_rom.S -o inst_rom.o
cherry@cherry-localhost:~/桌面$ mipsel-linux-ld -EB -T ram.ld inst_rom.o -o inst_rom.om
cherry@cherry-localhost:~/桌面$ mipsel-linux-objcopy -O binary inst_rom.om inst_rom.bin
cherry@cherry-localhost:~/桌面$ mipsel-linux-objdump -D inst_rom.om > inst_rom.asm
cherry@cherry-localhost:~/桌面$ ./Bin2Mem.exe -f inst_rom.bin -o inst_rom.data
cherry@cherry-localhost:~/桌面$
```

查看 inst_rom.data 文件:

```
cherry@cherry-localhost: ~/桌面
cherry@cherry-localhost:~/桌面$ mipsel-linux-as -mips32 -EB inst_rom.S -o inst_rom.o
cherry@cherry-localhost:~/桌面$ mipsel-linux-ld -EB -T ram.ld inst_rom.o -o inst_rom.om
cherry@cherry-localhost:~/桌面$ mipsel-linux-objcopy -O binary inst_rom.om inst_rom.bin
cherry@cherry-localhost:~/桌面$ mipsel-linux-objdump -D inst_rom.om > inst_rom.asm
cherry@cherry-localhost:~/桌面$ ./Bin2Mem.exe -f inst_rom.bin -o inst_rom.data
cherry@cherry-localhost:~/桌面$ cat inst_rom.data
34011100
34020020
3403ff00
3404ffff
0000001e
00000000
00000000
00000000
00000000
00000000
00000000
cherry@cherry-localhost:~/桌面$
```

已经修改为大端模式了。

5.3 Make 工具

每次都输入四个（加上.asm 一共五个）指令显得很麻烦，我们可以编写一个脚本来自动执行命令。

Make 的工作原理

当我们只输入 **make** 命令的工作流程是：

1. **make** 会在当前目录下找名字叫 “**Makefile**” 或 “**makefile**” 的文件；
2. 如果找到，它会找文件中的第一个目标文件 (**target**)，在上面的例子中，他会找到 “**inst_rom.data**” 这个文件，并把这个文件作为最终的目标文件；
3. 如果 **inst_rom.data** 文件不存在，或是 **.data** 所依赖的后面的 **.om** 文件的文件修改时间要比 **.data** 这个文件新，那么 **make** 会执行下面定义的命令来生成 **.data** 文件；
4. 如果 **.data** 所依赖的 **.om** 文件也存在，那么 **make** 会在当前文件中找目标为 **.om** 文件的依赖性，如果找到再根据命令生成 **.om** 文件（这是一个递归的过程）；

如果在找寻的过程中，出现了被依赖的文件找不到的错误，那么 **make** 就会直接退出，并报错。

如果在一条依赖链中，比如：**A** 依赖 **B**，**B** 依赖 **C**，**C** 依赖 **D**。那么当 **D** 更新后，**make** 发现 **D** 比 **C** 新则会重新构建 **C**，以此类推，最终 **A** 也会被更新。

简单的说，**makefile** 带来的好处就是——自动化编译，只要一个 **make** 命令，所有工程和文件自动编译，类似于 **Shell** 的 **.sh** 和 **cmd** 的 **.bat**

我们编写如下代码：

```
ifndef CROSS_COMPILE
    CROSS_COMPILE = mipsel-linux-
endif

CC = $(CROSS_COMPILE)as
LD = $(CROSS_COMPILE)ld

OBJCOPY = $(CROSS_COMPILE)objcopy
OBJDUMP = $(CROSS_COMPILE)objdump

OBJECTS = inst_rom.o
export CROSS_COMPILE

all:inst_rom.data inst_rom.om inst_rom.o inst_rom.bin inst_rom.asm

%.o:%.S
    $(CC) -mips32 -EB $< -o $@

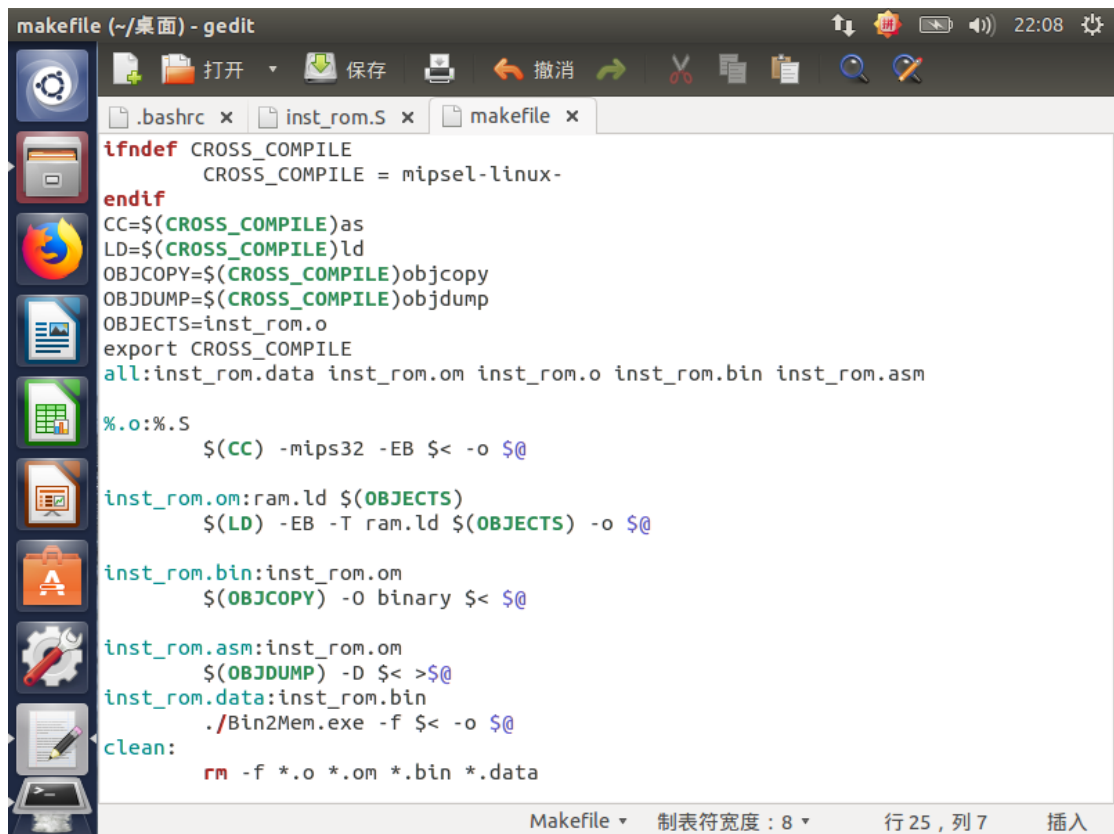
inst_rom.om:ram.ld $(OBJECTS)
    $(LD) -EB -T ram.ld $(OBJECTS) -o $@

inst_rom.bin:inst_rom.om
    $(OBJCOPY) -O binary $< $@

inst_rom.asm:inst_rom.om
    $(OBJDUMP) -D $< >$@

inst_rom.data:inst_rom.bin
    ./Bin2Mem.exe -f $< -o $@

clean:
    rm -f *.o *.om *.bin *.data
```



```
makefile (~/.桌面) - gedit
ifndef CROSS_COMPILE
    CROSS_COMPILE = mipsel-linux-
endif
CC=$(CROSS_COMPILE)as
LD=$(CROSS_COMPILE)ld
OBJCOPY=$(CROSS_COMPILE)objcopy
OBJDUMP=$(CROSS_COMPILE)objdump
OBJECTS=inst_rom.o
export CROSS_COMPILE
all:inst_rom.data inst_rom.om inst_rom.o inst_rom.bin inst_rom.asm

%.o:%.S
    $(CC) -mips32 -EB $< -o $@

inst_rom.om:ram.ld $(OBJECTS)
    $(LD) -EB -T ram.ld $(OBJECTS) -o $@

inst_rom.bin:inst_rom.om
    $(OBJCOPY) -O binary $< $@

inst_rom.asm:inst_rom.om
    $(OBJDUMP) -D $< >$@
inst_rom.data:inst_rom.bin
    ./Bin2Mem.exe -f $< -o $@

clean:
    rm -f *.o *.om *.bin *.data
```

其中 `$<` 表示第一个依赖文件的名称

`$@` 表示目标的完整名称

至此，我们用 `gcc` 编译汇编指令生成机器码的过程就做完了。

六、附录

6.1 ram.ld 代码

```
MEMORY
{
    ram(RW)    : ORIGIN = 0x00000000, LENGTH = 0x00001000
}

SECTIONS
{
    /*
    For some reason the linker script can't see the _reset_vector symbol
    (even if we declare it global), so we explicitly set it. */
    .text :
    {
        *(.text)
    } > ram

    .data :
    {
        *(.data)
    } > ram

    .bss :
    {
        *(.bss)
    } > ram

    .stack ALIGN(0x10) (NOLOAD):
    {
        *(.stack)
        _ram_end = .;
    } > ram
}

ENTRY (_start)
```

6.2 Bin2Mem.c 代码

```
#include <stdlib.h>
#include <stdio.h>

char *option_invalid = NULL;
char *option_file_in = NULL;
char *option_file_out = NULL;

FILE *file_in_descriptor = NULL;
FILE *file_out_descriptor = NULL;

void exception_handler(int code) {
    switch (code) {
        case 0:
            break;
        case 10001:
            printf("Error (10001): No option recognized.\n");
            printf("Please specify at least one valid option.\n");
            break;
        case 10002:
            printf("Error (10002): Invalid option: %s\n", option_invalid);
            break;
        case 10003:
            printf("Error (10003): No input Binary file specified.\n");
            break;
        case 10004:
            printf("Error (10004): Cannot open file: %s\n",
option_file_in);
            break;
        case 10005:
            printf("Error (10005): Cannot create file: %s\n",
option_file_out);
            break;
        default:
            break;
    }

    if (file_in_descriptor != NULL) {
        fclose(file_in_descriptor);
    }
    if (file_out_descriptor != NULL) {
        fclose(file_out_descriptor);
    }
}
```

```

    exit(0);
}

int main(int argc, char **argv) {

    int i=0,j=0;
    unsigned char temp1,temp2,temp3,temp4;
    unsigned int option_flag = 0;

    while (argc > 0) {
        if (**argv == '-') {
            (*argv) ++;
            switch (**argv) {
                case 'f':
                    option_flag |= 0x4;
                    argv ++;
                    option_file_in = *argv;
                    argc --;
                    break;
                case 'o':
                    option_flag |= 0x8;
                    argv ++;
                    option_file_out = *argv;
                    argc --;
                    break;
                default:
                    option_flag |= 0x1;
                    (*argv) --;
                    option_invalid = *argv;
                    break;
            }
        }
        argv ++;
        argc --;
    }

    file_in_descriptor = fopen(option_file_in, "rb");
    if (file_in_descriptor == NULL) {
        exception_handler(10004);
    }

    file_out_descriptor = fopen(option_file_out, "w");
    if (file_out_descriptor == NULL) {

```

```

        exception_handler(10005);
    }

    while (!feof(file_in_descriptor)) {

        fscanf(file_in_descriptor, "%c", &temp1);
        fscanf(file_in_descriptor, "%c", &temp2);
        fscanf(file_in_descriptor, "%c", &temp3);
        fscanf(file_in_descriptor, "%c", &temp4);

        if(!feof(file_in_descriptor))
        {
            fprintf(file_out_descriptor, "%02x", temp1);
            fprintf(file_out_descriptor, "%02x", temp2);
            fprintf(file_out_descriptor, "%02x", temp3);
            fprintf(file_out_descriptor, "%02x", temp4);
            fprintf(file_out_descriptor, "\n");
        }

    }

    exception_handler(0);
    return 0;
}

```