Manual Networking (and Minetest!)

Contents

1 Skills/Concepts

This module provides a very basic introduction to networking and some of the tools that can be used for network configuration.

The following concepts will be touched upon in this module:

- console use
- getting help for commands
- OSI Model
- IP Addresses
- Network Masks
- Network Switches/Hubs

1.1 Console Use

This module relies heavily on the command-line. You will be using a terminal emulator to execute the following commands.

Command	Description
`apt-get`	A tool for installing software
`apt-cache`	A tool for searching for installable software
`cmake`	A tool for generating build configurations
`ip`	General purpose utility for configuring the network
`make`	A tool for executing compilers and coordinating source builds
`man`	View on-line manual pages
`showmount`	A utility for viewing available NFS shares
`sudo`	Execute a command as another user

1.2 Getting Help for Commands

Technology these days makes Internet searching just as fast as any built in help, but in addition to googling for stack exchange articles, there is a built-in help or on-line manual for most commands under Linux. Built-in help can typically be accessed by passing `--help` or `-h` to the command of interest and it will typically reply back with a short usage block.

In addition to the built-in help, you can refer to manual pages for more usage information for most commands. These pages are generally terribly formatted, frustrating, and are entirely unhelpful if you're not used to reading them.

Various packages usually have some additional documentation in the form of a README or guide that can be found in `/usr/share/doc/`. Sometimes you will need to install an addition `-doc` package for whatever the package is to get these files.

I suggest you use the following approach to getting help:

- Check for a man page
- Check for documentation in `/usr/share/doc/`
- Google

That aside, the general approach to finding help for a given command should be whatever you are most comfortable with or whatever happens to be available. I recommended the above approach because it will help you get used to reading technical documentation so that in the future it will be easier to comprehend whatever documentation is available.

1.3 OSI Model

The Open Systems Interconnection(OSI) model is a conceptual model that aids in understanding the systems involved in networking. It's a seven layer model where data traverses the model one layer at a time.

Application
Presentation
Session
Transport
Network
Data Link
Physical

We'll discuss the layers briefly during the meeting.

Of the various mnemonics for remembering the layers, my favorite is:

> A Python Sent The Network Data Packing

1.4 IP Addresses

Internet Protocol addresses. There are two varieties of IP address in use on the public Internet, IPv4 and IPv6.

1.4.1 IPv4 Address

IPv4 addresses are made up of four bytes (sometimes called octets as each byte is being referred to as a collection of eight bits). IPv4 addresses generally appear in "dotted-decimal" notation. For example: `192.168.2.1`

Because they're constructed of only 4 bytes, there are only a total number of 2^{32} addresses, or 4,294,967,296. This number is further reduced by the allocation of special-use addresses.

1.4.2 IPv6 Address

IPv6 addresses are made up of eight groups of two bytes. IPv6 addresses are generally represented with some abbreviated form of the colon separated byte values. For example: `1234:5678:9ABC:DEF0:0000:0000:0000:0000` or `1234:5678:9ABC:DEF0::`.

Because it's 128 bits of address space, 2^{128} addresses can be represented. Writing this value out in long-form exceeds my vocabulary for numbers.

1.5 Network Masks

Network masks are used in routing tables to define what route traffic must take to reach a location. Any given computer may have multiple network interfaces, these interfaces are generally connected to some sort of network. In order for the operating system to decide which interface is to be used for traffic, it uses the network mask for each interface to determine whether traffic should be routed over that interface. Then, in transit, other systems have their own routing tables that are used to determine where to route the traffic based on the destination IP adddress and its set of configured network masks.

Generally an interface will have a default route over which any traffic not explicitly destined

1.6 Network Switches/Hubs

A network hub is a device that replicates all traffic recieved across all ports. In this context, a port refers to the connection between the ethernet cable and the RJ45 jack. A switch is a device that monitors media access addresses across its ports and only routes traffic between the ports for which traffic is addressed.

A hub can only handle one simultaneous transmission at a time, a switch can handle many.

2 Preparation

We'll be using a CISCO switch in this exercise and plugging in ethernet cables. Plug in the switch, power it on, and boot up the laptops that will be used. Plug your laptop into the switch using one of the provided ethernet cables.

3 Bring Up the Network

In this module, we will manually establish a new IPv4 network.

Once the machine is powered on, and plugged into the CISCO switch, open a terminal and carry out the following commands.

1. View the current network configuration with the ip command:

   ip addr show

The results will vary, but generally will look like the following:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
```

```
       valid_lft forever preferred_lft forever
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 60:a4:4c:ac:89:99 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.131/24 brd 192.168.10.255 scope global enp3s0
       valid_lft forever preferred_lft forever
    inet6 fe80::62a4:4cff:feac:8999/64 scope link
       valid_lft forever preferred_lft forever
```

2. Be lazy, and use the abbreviated form of the command:

   ip a

3. Bring up your wired ethernet device (if its state is listed as DOWN):

   sudo ip link set enp3s0 up

4. Assign your ethernet device an ip address, in this example we use `192.168.2.1` with a 24-bit mask:

   sudo ip addr add dev enp3s0 192.168.99.2/24

In this context, the 24-bit mask works out to being the 192.168.99 part of the address. So anything destined for the `192.168.99.0` network will be routed through the device specified.

4. Try to ping the raspbery pi server at 192.168.99.1:

   ping -c 2 192.168.99.1

The result will look similar to the following:

```
PING 192.168.99.1 (192.168.99.1) 56(84) bytes of data.
64 bytes from 192.168.99.1: icmp_seq=1 ttl=64 time=0.168 ms
64 bytes from 192.168.99.1: icmp_seq=2 ttl=64 time=0.185 ms
--- 192.168.99.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.164/0.172/0.180/0.008 ms
```

5. Confirm that your ping used the right interface with the `tracepath` command:
6. See what it looks like to use a different path (make sure you are connected to the wireless for this!):

   tracepath 8.8.8.8

3.1 Conclusion

You brought up the network interface with a static route. Then, you confirmed connectivity with other nodes on the network with the ping command. Although very simple tools, these are surprisingly useful tools for troubleshooting network problems.

Next, you confirmed that the route you thought was being used was actually being used for the new interface, and you were able to see the difference when a different route than expected was used.

4 Get the Files

For this module, we will use the connected network to mount a network share from a remote system and use it to download and build a game.

1. View the available shares on the remote system.

   showmount -e 192.168.99.1

Note: This may require the nfs-common package, if so:

```
sudo apt-get install nfs-common
```

Expected output:

```
Export list for 192.168.99.1:
/srv/minetest *
```

This indicates that there is a directory `/srv/minetest` that anyone may mount.

2. Mount the directory locally:

   sudo mount 192.168.99.1:/srv/minetest /mnt

3. Check out the files

   git clone /mnt/minetest/minetest

Epected output:

```
Cloning into 'minetest'...
done.
```

4. The probject supports various games, so check out the common minetest game as well:

```
cd minetest/games
git clone /mnt/minetest/minetest_game
```

Expected output:

```
Cloning into 'minetest_game'...
done.
```

4.1 Conclusion

You used the `showmount` command to see what was available on the remote system, then you used the mount command to make it available to your local system. Now, browsing that `/mnt` directory provides all of those remote files as though they existed locally. Then, you were able to make use of those files with the `git` command.

5 Build the Project

Now that you have the source, it's time to put it to use. This may require installing some compiler tools, such as CMake. If any of the commands fail because they are not installed we will address them as they fail, however, the general approach is to just apt-get install the package containing the missing command. For example, if `cmake` is missing:

```
sudo apt-get install cmake
```

1. Use cmake to create the build configuration:

   cmake . -DRUN_IN_PLACE=TRUE

Successful output will look like the following:

```
-- The C compiler identification is GNU 5.2.1
-- The CXX compiler identification is GNU 5.2.1
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- *** Will build version 0.4.15-dev ***
-- Found Irrlicht: /usr/lib/x86_64-linux-gnu/libIrrlicht.so
-- Using GMP provided by system.
-- Found GMP: /usr/lib/x86_64-linux-gnu/libgmp.so
-- Using bundled JSONCPP library.
-- Could NOT find LuaJit (missing:  LUA_LIBRARY LUA_INCLUDE_DIR)
-- LuaJIT not found, using bundled Lua.
-- Found CURL: /usr/lib/x86_64-linux-gnu/libcurl.so
-- cURL support enabled.
-- GetText disabled.
-- Found OpenAL: /usr/lib/x86_64-linux-gnu/libopenal.so
-- Found VORBIS: /usr/include
-- Sound enabled.
-- Found Freetype: /usr/lib/x86_64-linux-gnu/libfreetype.so
-- Freetype enabled.
-- Found Curses: /usr/lib/x86_64-linux-gnu/libcurses.so
-- ncurses console enabled.
-- Could NOT find PostgreSQL (missing:  PostgreSQL_LIBRARY
PostgreSQL_INCLUDE_DIR PostgreSQL_TYPE_INCLUDE_DIR)
-- PostgreSQL not found!
-- LevelDB not found!
-- Redis not found!
-- Found SQLite3: /usr/lib/x86_64-linux-gnu/libsqlite3.so
-- SpatialIndex not found!
-- Looking for XOpenDisplay in
/usr/lib/x86_64-linux-gnu/libX11.so;/usr/lib/x86_64-linux-gnu/libXext.so
-- Looking for XOpenDisplay in
/usr/lib/x86_64-linux-gnu/libX11.so;/usr/lib/x86_64-linux-gnu/libXext.so -
found
-- Looking for gethostbyname
-- Looking for gethostbyname - found
-- Looking for connect
-- Looking for connect - found
-- Looking for remove
-- Looking for remove - found
-- Looking for shmat
-- Looking for shmat - found
-- Looking for IceConnectionNumber in ICE
-- Looking for IceConnectionNumber in ICE - found
-- Found X11: /usr/lib/x86_64-linux-gnu/libX11.so
-- Found OpenGL: /usr/lib/x86_64-linux-gnu/libGL.so
-- Found JPEG: /usr/lib/x86_64-linux-gnu/libjpeg.so
-- Found BZip2: /usr/lib/x86_64-linux-gnu/libbz2.so (found version "1.0.6")
-- Looking for BZ2_bzCompressInit in /usr/lib/x86_64-linux-gnu/libbz2.so
-- Looking for BZ2_bzCompressInit in /usr/lib/x86_64-linux-gnu/libbz2.so -
found
```

```
-- Found ZLIB: /usr/lib/x86_64-linux-gnu/libz.so (found version "1.2.8")
-- Found PNG: /usr/lib/x86_64-linux-gnu/libpng.so (found version "1.2.51")
-- Looking for clock_gettime in rt
-- Looking for clock_gettime in rt - found
-- Looking for include file endian.h
-- Looking for include file endian.h - found
-- Could NOT find Doxygen (missing:  DOXYGEN_EXECUTABLE)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/minetest
```

Note that even if some things are not found, it can still be compiled.

2. Use make to build the project:

   make

This step will take the longest. It's creating the executable files that will actually run the game server and the client.

The final line of output should look like the following:

```
[100%] Built target minetest
```

During the build process, some utilities may be missing. We'll address these on a case by case basis and install them as necessary, but generally the development packages for whatever the failed dependency is will need to be installed. For example:

If you see:

```
-- Could NOT find Irrlicht (missing:  IRRLICHT_LIBRARY IRRLICHT_INCLUDE_DIR)
```

You might search for it as follows:

```
apt-cache search irrlicht dev
```

And the expected output would be:

```
libirrlicht-dev - High performance realtime 3D engine development library
```

So you would then need to install that package:

```
sudo apt-get install libirrlicht-dev
```

Or, knowing the dependencies, you could install them in one shot:

```
sudo apt-get install build-essential libirrlicht-dev cmake libbz2-dev \
libpng-dev libjpeg-dev libxxf86vm-dev libgl1-mesa-dev libsqlite3-dev \
libogg-dev libvorbis-dev libopenal-dev libcurl4-gnutls-dev \
libfreetype6-dev zlib1g-dev libgmp-dev libjsoncpp-dev
```

5.1 Conclusion

We took shortcuts here because I am familiar with the project and I knew what was involved in building it. The general approach to building something from source is to look for instructions on how to build that project. In the case of Minetest, there is a project `README.txt` that outlines the steps we followed.

6 Run the Game

Run the game and connect to the pi server.

1. From the directory where the game was built, you can launch the `minetest` executable from the `bin` sub-directory.

For example:

```
./bin/minetest
```

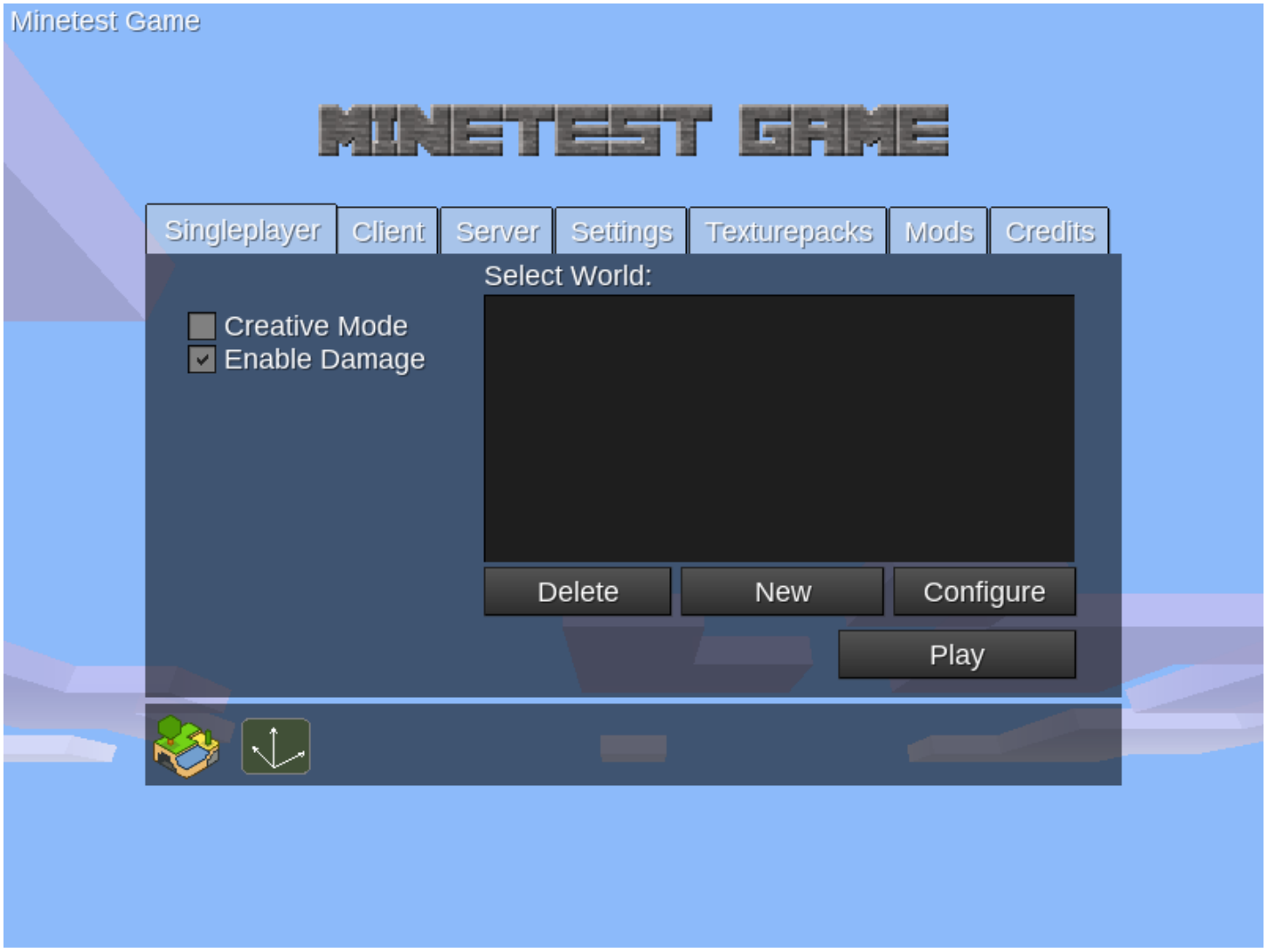

Figure 1: Minetest Window

2. Click on the *Client* tab, enter the IP address of the pi, enter a user name, pick a password, and click connect.

Alternately Choose the *Server* tab and create your own world.

6.1 Conclusion

Generally there is an additional step of installing the compiled code, but many projects support running the executables in place, as is the case with Minetest. But beware of performing a `make install` if a project suggests it unless you have taken the necessary steps according to that projects instructions to specify an installation path. If you blindly install projects to your installed system, you could end up with collisions between software you have installed from source and software installed from the distribution's package manager.

Figure 2: Minetest Client Tab