sshuttle - a VPN for the lazy

`sshuttle` is a Python based script that allows you to tunnel connections through SSH in a far more efficient way then traditional ssh proxying.

By far the greatest perk of sshuttle is that it requires no installation on the server side. **As long as you have an SSH server (with python installed) you're good to go.**

Because it inserts iptables entries, the client running sshuttle must be `root` however no special privileges are needed on the server.

How to use it

Install

```
$ sudo pip install sshuttle
# you may need to install
# python-pip
```

Tunnel to all networks (0.0.0.0/0)

```
$ sudo sshuttle -r user@server 0/0
```

Tunnel to a particular subnet

```
$ sudo sshuttle -r admin@router:31415 192.168.1.0/24
```

(You can set the port of the server in -r)

Other options you may want to use

`-e --ssh-cmd`
Customize the ssh command (by default, just ssh).
e.g. to use a specific ssh keypair `-e 'ssh -i ~/.ssh/aws/apse2.pem'`

`--dns`
Force all dns requests through the tunnel

`-D --daemon`
Run it as a daemon. Useful if you want to make a systemd unit

`-v -vv -vvv`
Set different levels of verbosity, at max, you see it on the console each time a new tcp session is redirected

How it works

The Docs explain it far better than I could. The key point is this

> sshuttle assembles the TCP stream locally, multiplexes it statefully over an ssh session, and disassembles it back into packets at the other end. So it never ends up doing TCP-over-TCP. It’s just data-over-TCP, which is safe.

Normal VPNs create an interface and forward traffic along as packets. The way sshuttle works is network agnostic. The SSH side works kinda like a serial link, rather than sending the traffic as complete packets, it sends it as raw data which only becomes packets at the other end.

On the host side of things, sshuttle listens on a local port, and creates iptables NAT rules that redirect traffic into this port. This port is the endpoint to "suck" traffic and through the use of magic (described in the docs) "teleport" the traffic to the server.

It creates the following iptables chain in the `nat` table.

```
$ iptables -t nat -N sshuttle-12300
```

At the top of the `PREROUTING` and `OUTPUT` chains, we jump to the sshuttle chain. The sshuttle chain has two quite simple rules.

- Do nothing for tcp traffic destined to localhost
  `-j RETURN --dest 127.0.0.1/32 -p tcp`
- Redirect tcp traffic into the locally listening sshuttle port.
  `-j REDIRECT --dest 0.0.0.0/0 -p tcp --to-ports 12300 -m ttl ! --ttl 42`

In the sshuttle docs, there is an option to specify the binding of this port. You can make it listen on a particular interface, or listen on all, or the default, listen on `127.0.0.1`.

In a pinch you could have sshuttle running on a box, open the port to a particular interface, and then on a router, apply these rules to redirect tcp traffic into sshuttle port on the box.

This way the router would not have to waste its limited processing power running a python script.