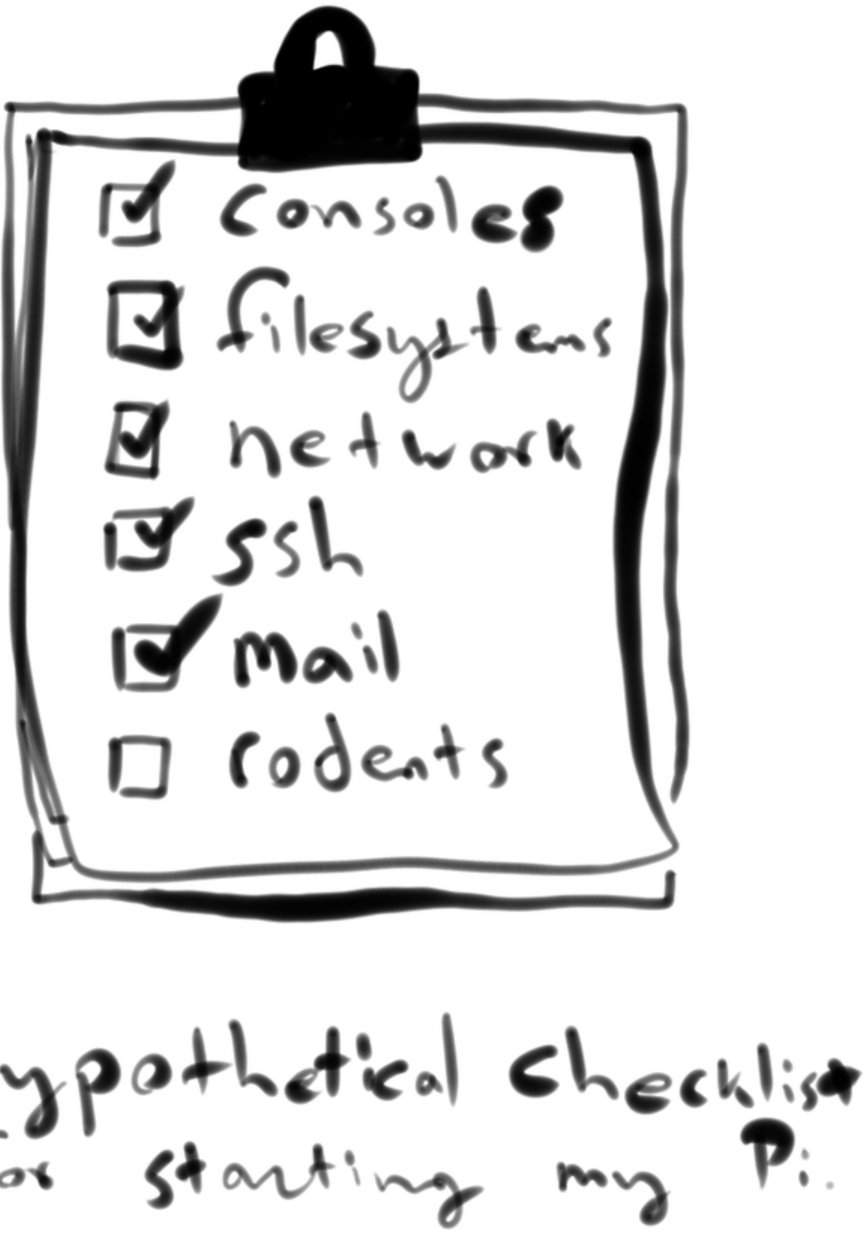
Running Programs Automatically on Your Tiny Computer

Created by Brennen Bearnes

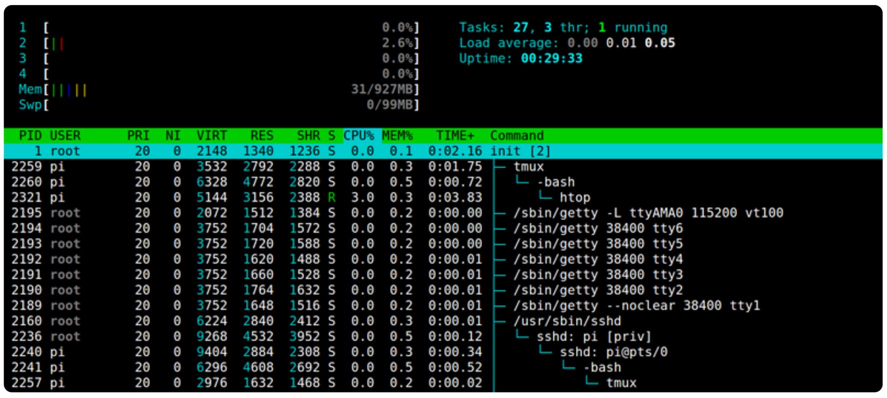


https://learn.adafruit.com/running-programs-automatically-on-your-tiny-computer

Last updated on 2023-08-29 02:58:45 PM EDT

Table of Contents

Overview



```
  1  [                                        0.0%]     Tasks: 27, 3 thr; 1 running
  2  [||                                      2.6%]     Load average: 0.00 0.01 0.05
  3  [                                        0.0%]     Uptime: 00:29:33
  4  [                                        0.0%]
  Mem[|||||                              31/927MB]
  Swp[                                     0/99MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
    1 root       20   0  2148  1340  1236 S  0.0  0.1  0:02.16 init [2]
 2259 pi         20   0  3532  2792  2288 S  0.0  0.3  0:01.75 ├─ tmux
 2260 pi         20   0  6328  4772  2820 S  0.0  0.5  0:00.72 │  └─ -bash
 2321 pi         20   0  5144  3156  2388 R  3.0  0.3  0:03.83 │     └─ htop
 2195 root       20   0  2072  1512  1384 S  0.0  0.2  0:00.00 ├─ /sbin/getty -L ttyAMA0 115200 vt100
 2194 root       20   0  3752  1704  1572 S  0.0  0.2  0:00.00 ├─ /sbin/getty 38400 tty6
 2193 root       20   0  3752  1720  1588 S  0.0  0.2  0:00.00 ├─ /sbin/getty 38400 tty5
 2192 root       20   0  3752  1620  1488 S  0.0  0.2  0:00.01 ├─ /sbin/getty 38400 tty4
 2191 root       20   0  3752  1660  1528 S  0.0  0.2  0:00.01 ├─ /sbin/getty 38400 tty3
 2190 root       20   0  3752  1764  1632 S  0.0  0.2  0:00.01 ├─ /sbin/getty 38400 tty2
 2189 root       20   0  3752  1648  1516 S  0.0  0.2  0:00.01 ├─ /sbin/getty --noclear 38400 tty1
 2160 root       20   0  6224  2840  2412 S  0.0  0.3  0:00.01 ├─ /usr/sbin/sshd
 2236 root       20   0  9268  4532  3952 S  0.0  0.5  0:00.12 │  └─ sshd: pi [priv]
 2240 pi         20   0  9404  2884  2308 S  0.0  0.3  0:00.34 │     └─ sshd: pi@pts/0
 2241 pi         20   0  6296  4608  2692 S  0.0  0.5  0:00.52 │        └─ -bash
 2257 pi         20   0  2976  1632  1468 S  0.0  0.2  0:00.02 │           └─ tmux
```

For projects using a small GNU/Linux system, it's often important that certain code always be running when the computer is on. You can always log in and start a script running by hand once you turn on the power, but this isn't very reliable - after all, the power might flicker, or the system might crash for unrelated reasons. (There's also a good chance you'll be running a system without a monitor or keyboard attached, which can make things that much more complicated - it's much easier to have the right stuff run automatically.)

As an example scenario, I have a Raspberry Pi system in charge of logging data from my mousetrap (), and I'm about to go on a trip to another continent. The electricity where I live goes out now and then, and I'd like to make sure that my mouse-counting code comes back up on a reboot.

a hypothetical checklist for starting my Pi.

On an Arduino or similar device, the code you write usually just runs shortly after the power comes on, courtesy of a bit of software called a bootloader. Linux systems have these too, but what runs by default is the kernel, which in turn runs something known as an init system that's in charge of starting all sorts of other software to provide important services.

We'll look at how to use two different init systems which are both common in the wild.

First, we'll check out SysV-style init on a Raspberry Pi 2. With roots dating back to early Unix systems, sysvinit was widely used on most Linux distributions until recently, and is still used by the version of Raspbian released in May of 2015, based on Debian Wheezy. It's also similar to mechanisms still used by the BSD branch of the Unix family tree ().

Next, we'll look at systemd () on a BeagleBone Black running Debian Wheezy with systemd. systemd is "a suite of basic building blocks for a Linux system", recently adopted by most of the major distributions, including Debian Jessie (the current stable release of the project). Eventually, Raspbian will probably run systemd too.

This guide assumes familiarity with the basics of the GNU/Linux command line, navigating the filesystem, and editing files.

As always, if you need an introduction to these topics, please check out the rest of our series on learning Linux with the Raspberry Pi ().

An Example Service: mouse.py

My goal is to have the init system start a script called `mouse.py` every time the Pi boots to multi-user mode (which hopefully will also mean that the network is available).

If you're following along at home, there's a good chance you don't want to mess around with my entire mouse logging project right now, so I'll write a quick Python script for testing purposes (based on this StackOverflow answer by Nathan Jhaveri ()).

You can paste or type the code into `~/mouse.py` using Nano and mark it executable:

```
nano /home/pi/mouse.py
```

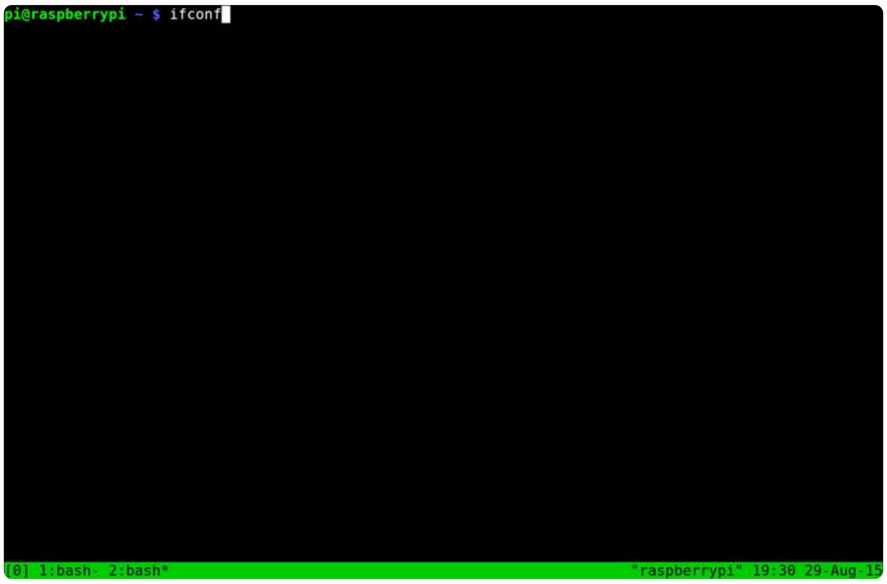
```
#!/usr/bin/python
import SocketServer
from BaseHTTPServer import BaseHTTPRequestHandler

class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/plain")
        self.end_headers()
        self.wfile.write('&lt;:3)~~~')

httpd = SocketServer.TCPServer(("", 80), MyHandler)
httpd.serve_forever()
```

```
chmod +x ~/mouse.py
```

All this does is create a very simple web server that, when you visit the Raspberry Pi's network address, returns an ASCII mouse. To test, I'll check my Pi's IP address, start the script, and use `curl` to make sure it's running. (I'm using tmux to open more than one shell ().)

Your IP address will likely be different, but in my case this looks like so:

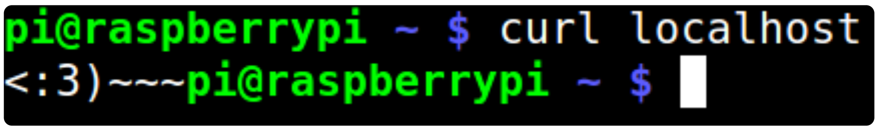
```
ifconfig
sudo ./mouse.py
curl 192.168.0.13
```



```
pi@raspberrypi ~ $ ifconf
```

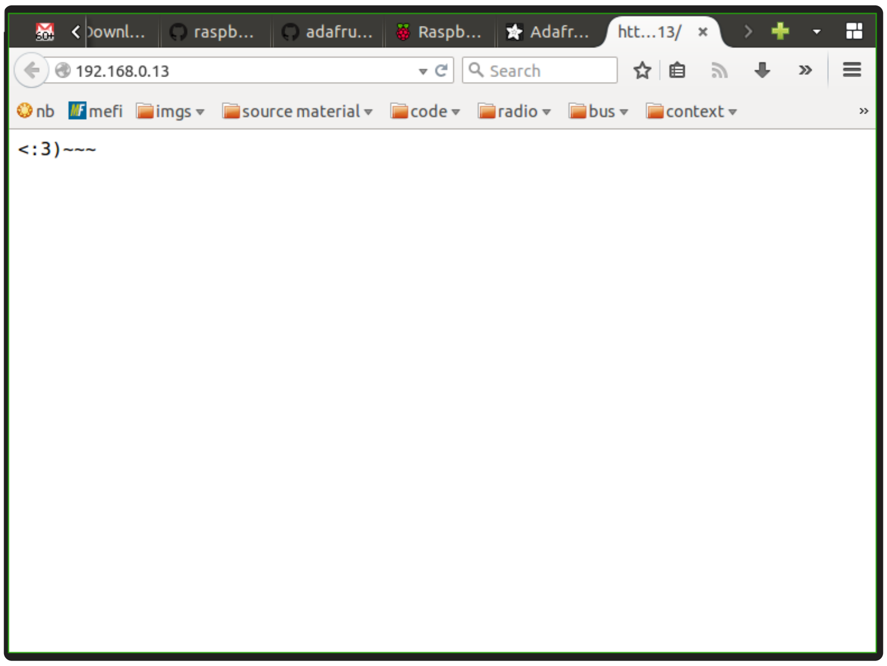
```
[0] 1:bash- 2:bash*                                          "raspberrypi" 19:30 29-Aug-15
```

If you're not on a network, you can just use `localhost` instead:



```
pi@raspberrypi ~ $ curl localhost
<:3)~~~pi@raspberrypi ~ $ 
```

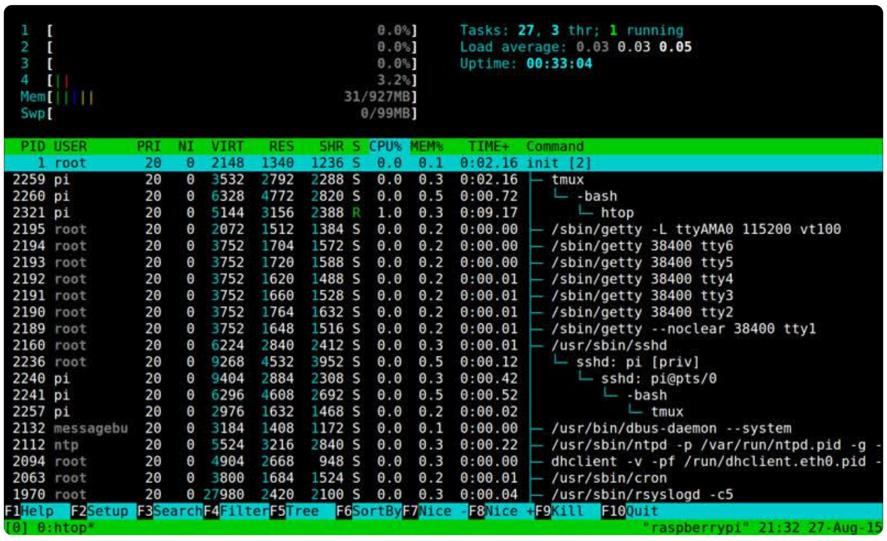
And if you want to test in a web browser, just enter the Pi's address in the URL bar:



```
<:3)~~~
```

Now that we have an example service, let's talk about how to start it.

SysV init: Runlevels



```
  1  [                                        0.0%]     Tasks: 27, 3 thr; 1 running
  2  [                                        0.0%]     Load average: 0.03 0.03 0.05
  3  [                                        0.0%]     Uptime: 00:33:04
  4  [||                                      3.2%]
  Mem[|||||                              31/927MB]
  Swp[                                     0/99MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
    1 root       20   0  2148  1340  1236 S  0.0  0.1  0:02.16 init [2]
 2259 pi         20   0  3532  2792  2288 S  0.0  0.3  0:02.16 ├─ tmux
 2260 pi         20   0  6328  4772  2820 S  0.0  0.5  0:00.72 │  └─ -bash
 2321 pi         20   0  5144  3156  2388 R  1.0  0.3  0:09.17 │     └─ htop
 2195 root       20   0  2072  1512  1384 S  0.0  0.2  0:00.00 ├─ /sbin/getty -L ttyAMA0 115200 vt100
 2194 root       20   0  3752  1704  1572 S  0.0  0.2  0:00.00 ├─ /sbin/getty 38400 tty6
 2193 root       20   0  3752  1720  1588 S  0.0  0.2  0:00.00 ├─ /sbin/getty 38400 tty5
 2192 root       20   0  3752  1620  1488 S  0.0  0.2  0:00.01 ├─ /sbin/getty 38400 tty4
 2191 root       20   0  3752  1660  1528 S  0.0  0.2  0:00.01 ├─ /sbin/getty 38400 tty3
 2190 root       20   0  3752  1764  1632 S  0.0  0.2  0:00.01 ├─ /sbin/getty 38400 tty2
 2189 root       20   0  3752  1648  1516 S  0.0  0.2  0:00.01 ├─ /sbin/getty --noclear 38400 tty1
 2160 root       20   0  6224  2840  2412 S  0.0  0.3  0:00.01 ├─ /usr/sbin/sshd
 2236 root       20   0  9268  4532  3952 S  0.0  0.5  0:00.12 │  └─ sshd: pi [priv]
 2240 pi         20   0  9404  2884  2308 S  0.0  0.3  0:00.42 │     └─ sshd: pi@pts/0
 2241 pi         20   0  6296  4608  2692 S  0.0  0.5  0:00.52 │        └─ -bash
 2257 pi         20   0  2976  1632  1468 S  0.0  0.2  0:00.02 │           └─ tmux
 2132 messagebu  20   0  3184  1408  1172 S  0.0  0.1  0:00.00 ├─ /usr/bin/dbus-daemon --system
 2112 ntp        20   0  5524  3216  2840 S  0.0  0.3  0:00.22 ├─ /usr/sbin/ntpd -p /var/run/ntpd.pid -g -
 2094 root       20   0  4904  2668   948 S  0.0  0.3  0:00.00 ├─ dhclient -v -pf /run/dhclient.eth0.pid -
 2063 root       20   0  3800  1684  1524 S  0.0  0.2  0:00.01 ├─ /usr/sbin/cron
 1970 root       20   0 27980  2420  2100 S  0.0  0.3  0:00.04 ├─ /usr/sbin/rsyslogd -c5
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
[0] 0:htop*                                               "raspberrypi" 21:32 27-Aug-15
```

Runlevels and /etc/rc*.d/

The core idea of sysvinit is something called runlevels (), which are essentially just a way of organizing a collection of init scripts which have to run when the system starts or shuts down. Each runlevel corresponds to a directory in `/etc/`, which in turn contains symlinks to scripts in `/etc/init.d/`.

For historical reasons, there are a lot of runlevels, but a bunch of them aren't really used for anything special on Debian systems. You can read the full details of this in the Debian manual (), but the short version is that the system normally boots to runlevel 2, which is multi-user mode. Adapted from the manual, here's a partial table of levels:

runlevel	directory	usage
N		System boot.
0	/etc/rc0.d/	Halt the system.
1	/etc/rc1.d/	Single user mode (if you switch from multi-user mode).

2	/etc/rc2.d/	Multi-user mode.
3 - 5	/etc/rc3.d through /etc/rc5.d/	Identical to runlevel 2 (unless you do something funny with your system).
6	/etc/rc6.d/	Reboot the system.

Let's get a list of what's in runlevel 2:

```
cd /etc/rc2.d/
ls -l
```

```
pi@raspberrypi ~ $
```

`ls -l` gives a long listing of files, which will helpfully show you if a file is really a link to another file. Again, all of the actual init scripts turn out to live in `/etc/init.d`.

If a file in `/etc/rc*.d/` starts with `K`, it will be invoked by the init system with `[name of script] stop` (K is for kill), and if it starts with `S`, it will be invoked with `[name of script] start`.

Init scripts usually accept (most of) the following command-line parameters:

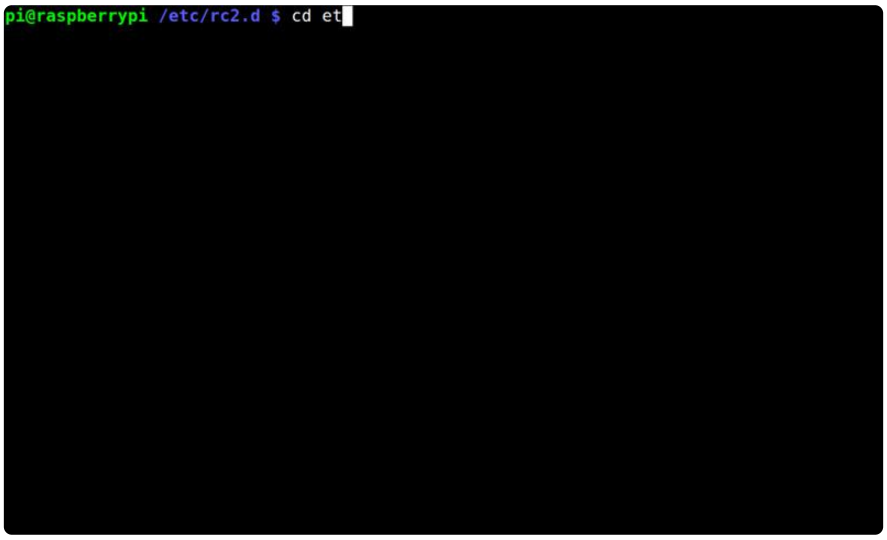
- `start` - start the service
- `stop` - stop the service
- `status` - check whether the service is running
- `reload` - have the service reload configuration files
- `restart` - stop and start the service

So, for example, you could say `/etc/init.d/ssh restart` to stop and start the SSH service. You often see commands like this in tutorials and HOWTOs.

The SSH Init Script

As an example, load up `/etc/init.d/ssh` in a text editor and give it a look.

```
cd /etc/init.d
nano ssh
```



```
pi@raspberrypi /etc/rc2.d $ cd et
```

This is actually quite a few lines of code for what seems like a pretty simple task - it just needs to start or stop a program, right? It turns out, though, that there can be a lot of considerations involved in doing that. For example, these rather squirrelly-looking lines...

```
test -x /usr/sbin/sshd || exit 0
( /usr/sbin/sshd -\? 2&gt;&amp;1 | grep -q OpenSSH ) 2&gt;/dev/null || exit 0
```

...make sure that `sshd` exists, is executable, and at least claims to be the SSH daemon provided by the OpenSSH project. The rest of the script defines functions for all sorts of housekeeping and sanity checks before it gets to the part where it handles the start/stop/etc. commands in a case statement:

```
case "$1" in
  start)
        check_privsep_dir
        check_for_no_start
        check_dev_null
        log_daemon_msg "Starting OpenBSD Secure Shell server" "sshd" || true
        if start-stop-daemon --start --quiet --oknodo --pidfile /var/run/sshd.pid --
exec /usr/sbin/sshd -- $SSHD_OPTS; then
            log_end_msg 0 || true
        else
            log_end_msg 1 || true
        fi
        ;;
  stop)
        log_daemon_msg "Stopping OpenBSD Secure Shell server" "sshd" || true
        if start-stop-daemon --stop --quiet --oknodo --pidfile /var/run/sshd.pid;
then
            log_end_msg 0 || true
        else
            log_end_msg 1 || true
        fi
        ;;

   # a whole bunch of other stuff happens here

esac
```

For one-off projects on the Pi, you almost certainly don't need to exercise this level of caution, but you will want to handle the common commands.

It would be a pain to write a lot of this stuff out from scratch. Fortunately, we have `/etc/init.d/skeleton` to work with. This just provides the "bones" of an init script, and should work with a few minor changes. Next, we'll look at adapting it to run our example `mouse.py`.

SysV Init: Writing an Init Script for mouse.py

Copy and Modify the Example Init Script

Start by copying `/etc/init.d/skeleton` to `/etc/init.d/mouselogger`, and marking it executable:

```
cd /etc/init.d
sudo cp skeleton mouselogger
sudo chmod +x mouselogger
```

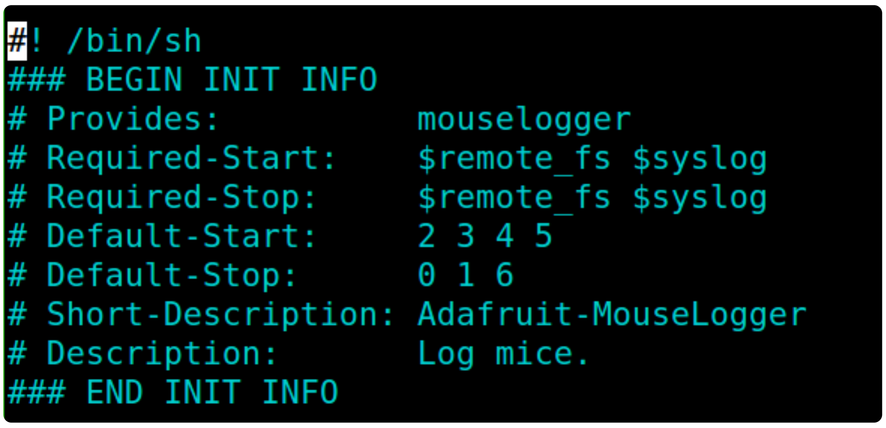
We need the `sudo` here because (like most things in `/etc`) files in `/etc/init.d` are owned by root. (You can imagine what might happen on a system with lots of users if just anyone could muck around in the startup files.)

Next, open this up in Nano:

```
nano mouselogger
```

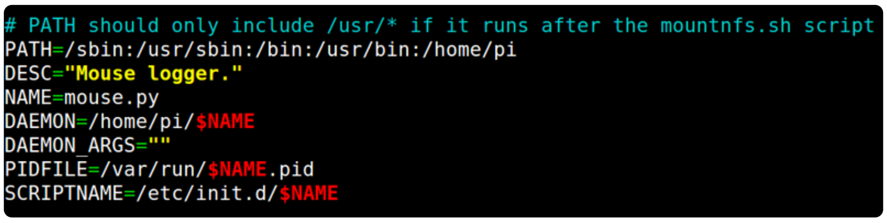
```
pi@raspberrypi /etc/init.d $ ud
```

I had to change three things. First, the metadata between `BEGIN INIT INFO` and `END INIT INFO` to specifically reference a service called "mouselogger" and provide a description:



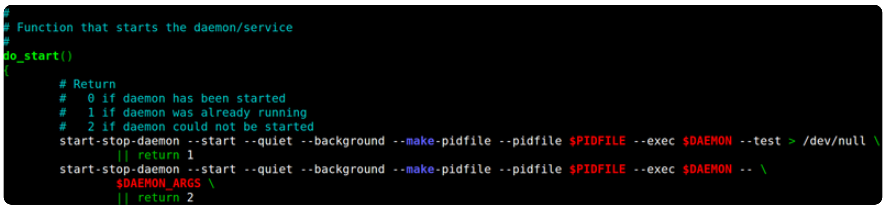
```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          mouselogger
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Adafruit-MouseLogger
# Description:       Log mice.
### END INIT INFO
```

Second, the environment variables that the rest of the script uses to do its business. I added `/home/pi` to the path, changed the description to "Mouse logger.", and changed the name to `mouse.py`, as well as emptying the `DAEMON_ARGS` variable:



```
# PATH should only include /usr/* if it runs after the mountnfs.sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/home/pi
DESC="Mouse logger."
NAME=mouse.py
DAEMON=/home/pi/$NAME
DAEMON_ARGS=""
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME
```

And lastly, I added `--background` and `--make-pidfile` to the `start-stop-daemon` options in the `do_start()` function:



```
#
# Function that starts the daemon/service
#
do_start()
{
        # Return
        #   0 if daemon has been started
        #   1 if daemon was already running
        #   2 if daemon could not be started
        start-stop-daemon --start --quiet --background --make-pidfile --pidfile $PIDFILE --exec $DAEMON --test > /dev/null \
                || return 1
        start-stop-daemon --start --quiet --background --make-pidfile --pidfile $PIDFILE --exec $DAEMON -- \
                $DAEMON_ARGS \
                || return 2
```

This ensures that the script is sent to the background instead of running in the foreground after it starts, and creates a file in `/var/run` that contains the process ID of the script.

Here's a copy of the entire file ().

Test the Script and Install with update-rc.d

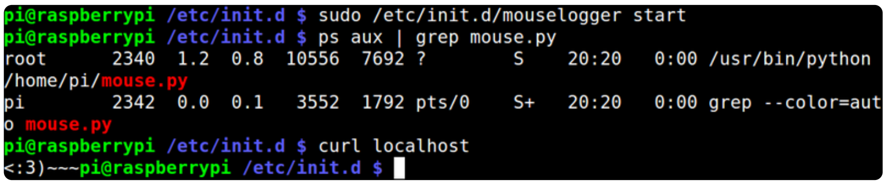
You should now be able to run `/etc/init.d/mouselogger` - but first, make sure that `/home/pi/mouse.py` isn't already running. If you have it open in a terminal, you can type Ctrl-c to kill the process. Otherwise, use `ps aux | grep mouse.py` to see if it has a running process:

```
pi@raspberrypi /etc/init.d $ ps a
```

Here, I checked with `ps` and found a copy of the Python interpreter running `mouse.py`, so I killed the corresponding process with `sudo kill 2052`. (2052 is the process id in the second column of the output from `ps`.)

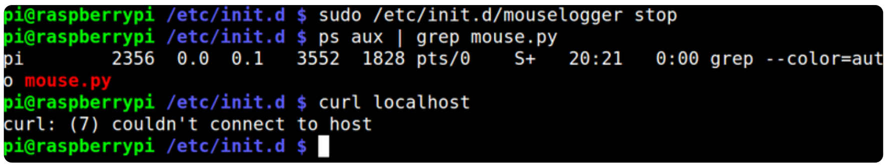
Now you can try starting and stopping the service and testing it out:

```
sudo /etc/init.d/mouselogger start
ps aux | grep mouse.py
curl localhost
```



```
pi@raspberrypi /etc/init.d $ sudo /etc/init.d/mouselogger start
pi@raspberrypi /etc/init.d $ ps aux | grep mouse.py
root      2340  1.2  0.8  10556  7692 ?        S    20:20   0:00 /usr/bin/python
/home/pi/mouse.py
pi        2342  0.0  0.1   3552  1792 pts/0    S+   20:20   0:00 grep --color=aut
o mouse.py
pi@raspberrypi /etc/init.d $ curl localhost
<:3)~~~pi@raspberrypi /etc/init.d $ 
```

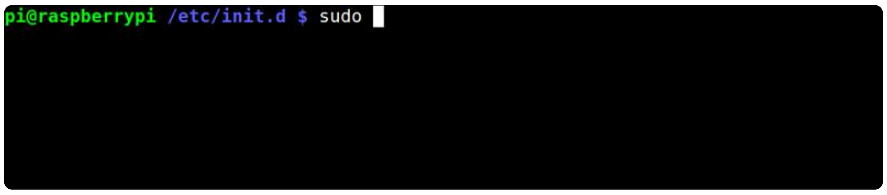
```
sudo /etc/init.d/mouselogger stop
```



```
pi@raspberrypi /etc/init.d $ sudo /etc/init.d/mouselogger stop
pi@raspberrypi /etc/init.d $ ps aux | grep mouse.py
pi        2356  0.0  0.1   3552  1828 pts/0    S+   20:21   0:00 grep --color=aut
o mouse.py
pi@raspberrypi /etc/init.d $ curl localhost
curl: (7) couldn't connect to host
pi@raspberrypi /etc/init.d $ 
```

Once you're confident that the init script works, you can install it in the default runlevels so that it starts at boot time:

```
sudo update-rc.d mouselogger defaults
```



```
pi@raspberrypi /etc/init.d $ sudo 
```

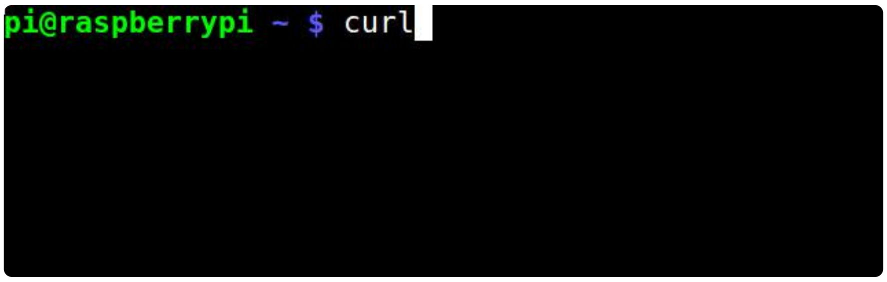
Reboot

And now, the moment of truth! Reboot and see what happens...

```
sudo reboot
```

```
pi@raspberrypi /etc/init.d $ sudo re
```

This part might take a bit, but once you can log back in, it should be simple to test whether `mouse.py` is running:



```
pi@raspberrypi ~ $ curl
```

Success!

Next up, we'll look at doing the equivalent thing with systemd on a BeagleBone Black.

---

systemd: Writing and Enabling a Service

This section uses a BeagleBone Black Rev C, running a Debian Wheezy release from July 2015.

We have a whole category of BeagleBone tutorials (). I found the specific image I installed on the elinux BeagleBoneBlack Debian page (). Recent editions of the BeagleBone likely come with something very similar installed.

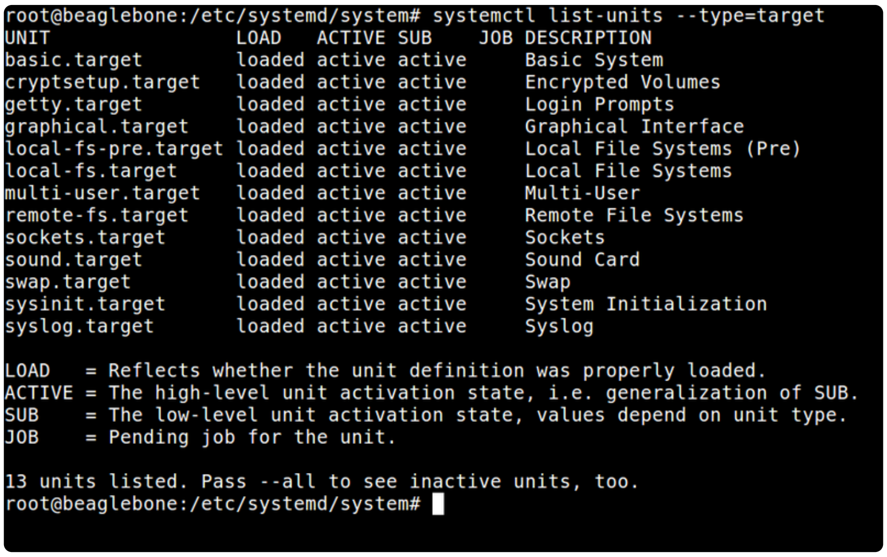
This section assumes that you are logged into your BBB as root.

Exploring systemd Basics

systemd takes a very different approach from the sysvinit scheme of organizing init scripts into directories by runlevel. Instead, it uses unit files to describe services that should run, along with other elements of the system configuration. These are organized into named targets, like `multi-user.target` and `graphical.target`.

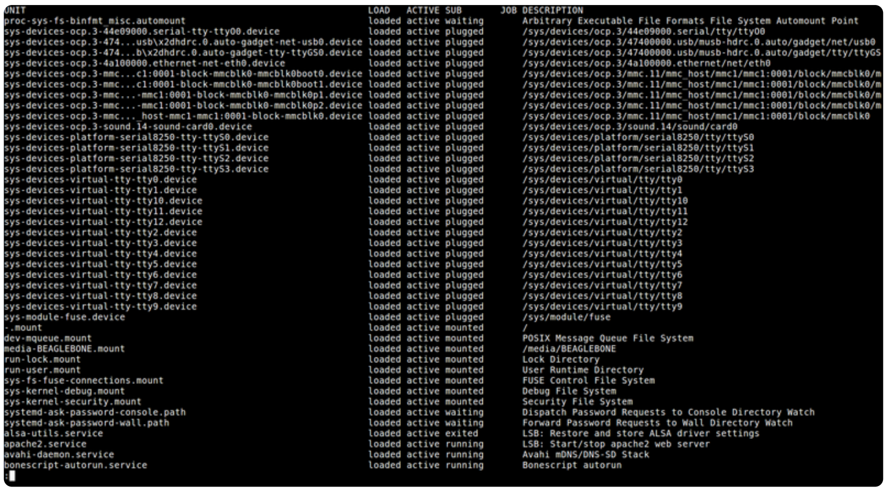
You can list targets on the current system like so:

```
sudo systemctl list-units --type=target
```



```
root@beaglebone:/etc/systemd/system# systemctl list-units --type=target
UNIT                 LOAD   ACTIVE SUB    JOB DESCRIPTION
basic.target         loaded active active     Basic System
cryptsetup.target    loaded active active     Encrypted Volumes
getty.target         loaded active active     Login Prompts
graphical.target     loaded active active     Graphical Interface
local-fs-pre.target  loaded active active     Local File Systems (Pre)
local-fs.target      loaded active active     Local File Systems
multi-user.target    loaded active active     Multi-User
remote-fs.target     loaded active active     Remote File Systems
sockets.target       loaded active active     Sockets
sound.target         loaded active active     Sound Card
swap.target          loaded active active     Swap
sysinit.target       loaded active active     System Initialization
syslog.target        loaded active active     Syslog

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.
JOB    = Pending job for the unit.

13 units listed. Pass --all to see inactive units, too.
root@beaglebone:/etc/systemd/system#
```

Just running `systemctl` by itself will show you a list of all active units, including configuration for devices, services, filesystem mountpoints, timers, and sockets.

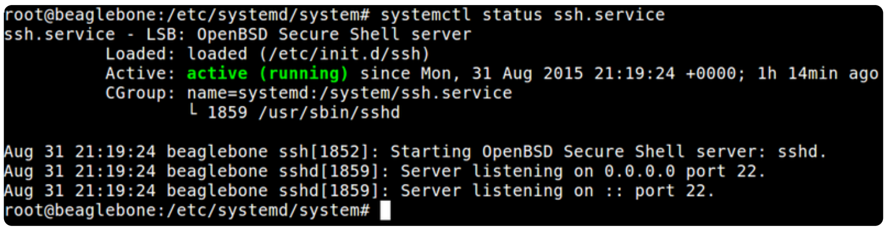


```
UNIT                                                              LOAD   ACTIVE SUB       JOB DESCRIPTION
proc-sys-fs-binfmt_misc.automount                                 loaded active waiting       Arbitrary Executable File Formats File System Automount Point
sys-devices-ocp.3-44e09000.serial-tty-ttyO0.device                loaded active plugged       /sys/devices/ocp.3/44e09000.serial/tty/ttyO0
sys-devices-ocp.3-474...usb\x2dhdrc.0.auto-gadget-net-usb0.device loaded active plugged       /sys/devices/ocp.3/47400000.usb/musb-hdrc.0.auto/gadget/net/usb0
sys-devices-ocp.3-474...b\x2dhdrc.0.auto-gadget-tty-ttyGS0.device loaded active plugged       /sys/devices/ocp.3/47400000.usb/musb-hdrc.0.auto/gadget/tty/ttyGS
sys-devices-ocp.3-4a100000.ethernet-net-eth0.device               loaded active plugged       /sys/devices/ocp.3/4a100000.ethernet/net/eth0
sys-devices-ocp.3-mmc...c1:0001-block-mmcblk0-mmcblk0boot0.device loaded active plugged       /sys/devices/ocp.3/mmc.11/mmc_host/mmc1/mmc1:0001/block/mmcblk0/m
sys-devices-ocp.3-mmc...c1:0001-block-mmcblk0-mmcblk0boot1.device loaded active plugged       /sys/devices/ocp.3/mmc.11/mmc_host/mmc1/mmc1:0001/block/mmcblk0/m
sys-devices-ocp.3-mmc...-mmc1:0001-block-mmcblk0-mmcblk0p1.device loaded active plugged       /sys/devices/ocp.3/mmc.11/mmc_host/mmc1/mmc1:0001/block/mmcblk0/m
sys-devices-ocp.3-mmc...-mmc1:0001-block-mmcblk0-mmcblk0p2.device loaded active plugged       /sys/devices/ocp.3/mmc.11/mmc_host/mmc1/mmc1:0001/block/mmcblk0/m
sys-devices-ocp.3-mmc..._host-mmc1-mmc1:0001-block-mmcblk0.device loaded active plugged       /sys/devices/ocp.3/mmc.11/mmc_host/mmc1/mmc1:0001/block/mmcblk0
sys-devices-ocp.3-sound.14-sound-card0.device                     loaded active plugged       /sys/devices/ocp.3/sound.14/sound/card0
sys-devices-platform-serial8250-tty-ttyS0.device                  loaded active plugged       /sys/devices/platform/serial8250/tty/ttyS0
sys-devices-platform-serial8250-tty-ttyS1.device                  loaded active plugged       /sys/devices/platform/serial8250/tty/ttyS1
sys-devices-platform-serial8250-tty-ttyS2.device                  loaded active plugged       /sys/devices/platform/serial8250/tty/ttyS2
sys-devices-platform-serial8250-tty-ttyS3.device                  loaded active plugged       /sys/devices/platform/serial8250/tty/ttyS3
sys-devices-virtual-tty-tty0.device                               loaded active plugged       /sys/devices/virtual/tty/tty0
sys-devices-virtual-tty-tty1.device                               loaded active plugged       /sys/devices/virtual/tty/tty1
sys-devices-virtual-tty-tty10.device                              loaded active plugged       /sys/devices/virtual/tty/tty10
sys-devices-virtual-tty-tty11.device                              loaded active plugged       /sys/devices/virtual/tty/tty11
sys-devices-virtual-tty-tty12.device                              loaded active plugged       /sys/devices/virtual/tty/tty12
sys-devices-virtual-tty-tty2.device                               loaded active plugged       /sys/devices/virtual/tty/tty2
sys-devices-virtual-tty-tty3.device                               loaded active plugged       /sys/devices/virtual/tty/tty3
sys-devices-virtual-tty-tty4.device                               loaded active plugged       /sys/devices/virtual/tty/tty4
sys-devices-virtual-tty-tty5.device                               loaded active plugged       /sys/devices/virtual/tty/tty5
sys-devices-virtual-tty-tty6.device                               loaded active plugged       /sys/devices/virtual/tty/tty6
sys-devices-virtual-tty-tty7.device                               loaded active plugged       /sys/devices/virtual/tty/tty7
sys-devices-virtual-tty-tty8.device                               loaded active plugged       /sys/devices/virtual/tty/tty8
sys-devices-virtual-tty-tty9.device                               loaded active plugged       /sys/devices/virtual/tty/tty9
sys-module-fuse.device                                            loaded active plugged       /sys/module/fuse
-.mount                                                           loaded active mounted       /
dev-mqueue.mount                                                  loaded active mounted       POSIX Message Queue File System
media-BEAGLEBONE.mount                                            loaded active mounted       /media/BEAGLEBONE
run-lock.mount                                                    loaded active mounted       Lock Directory
run-user.mount                                                    loaded active mounted       User Runtime Directory
sys-fs-fuse-connections.mount                                     loaded active mounted       FUSE Control File System
sys-kernel-debug.mount                                            loaded active mounted       Debug File System
sys-kernel-security.mount                                         loaded active mounted       Security File System
systemd-ask-password-console.path                                 loaded active waiting       Dispatch Password Requests to Console Directory Watch
systemd-ask-password-wall.path                                    loaded active waiting       Forward Password Requests to Wall Directory Watch
alsa-utils.service                                                loaded active exited        LSB: Restore and store ALSA driver settings
apache2.service                                                   loaded active running       LSB: Start/stop apache2 web server
avahi-daemon.service                                              loaded active running       Avahi mDNS/DNS-SD Stack
bonescript-autorun.service                                        loaded active running       Bonescript autorun
:
```

Mostly, systemd units are defined in various files in directories like `/lib/systemd/system`.

If you want to know the status of an existing service, you can use something like `systemctl status foo.service` - for example, let's check up on SSH:

```
sudo systemctl status ssh.service
```



```
root@beaglebone:/etc/systemd/system# systemctl status ssh.service
ssh.service - LSB: OpenBSD Secure Shell server
          Loaded: loaded (/etc/init.d/ssh)
          Active: active (running) since Mon, 31 Aug 2015 21:19:24 +0000; 1h 14min ago
          CGroup: name=systemd:/system/ssh.service
                  └ 1859 /usr/sbin/sshd

Aug 31 21:19:24 beaglebone ssh[1852]: Starting OpenBSD Secure Shell server: sshd.
Aug 31 21:19:24 beaglebone sshd[1859]: Server listening on 0.0.0.0 port 22.
Aug 31 21:19:24 beaglebone sshd[1859]: Server listening on :: port 22.
root@beaglebone:/etc/systemd/system#
```

You might notice something a bit odd here - this status report still mentions `/etc/init.d/ssh`. Isn't that a sysvinit thing? Well, yeah. What is happening is that the system we're on is still a hybrid between old-school and new-school init systems. systemd itself is actually capable of acting as a replacement (mostly) for sysvinit.

This all looks a little daunting, and there are a lot of concepts to untangle if you want to understand everything that's going on (your humble narrator is still quite a ways from this), but just configuring a new service is very little work.

Writing a Unit File

Let's reuse our example service, `mouse.py`. Run `sudo nano /root/mouse.py`, and paste the following code (notice that I've changed the port from 80 to 8888, so as not to step on the toes of other services already running on the BeagleBone):

```
#!/usr/bin/python
import SocketServer
from BaseHTTPServer import BaseHTTPRequestHandler
```

```
class MyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/plain")
        self.end_headers()
        self.wfile.write('&lt;:3)~~~\n')

httpd = SocketServer.TCPServer(("", 8888), MyHandler)
httpd.serve_forever()
```

Then make sure this is executable with `sudo chmod +x /root/mouse.py`. Next, create `/lib/systemd/system/mouselogger.service` and use Nano to paste the following:

```
[Unit]
Description=Mouse Logging Service

[Service]
ExecStart=/root/mouse.py
StandardOutput=null

[Install]
WantedBy=multi-user.target
Alias=mouselogger.service
```

Now you can enable your new service:

```
sudo systemctl enable mouselogger.service
sudo systemctl start mouselogger.service
```

```
root@beaglebone:/etc/systemd/system# syst
```

Did it work?

```
curl localhost:8888
```

raspberry_pi_screencast-2015-08-31-18_20_31.gif

Sure enough! As a last step, you should probably reboot and repeat this test to make sure.

Further Reading

sysvinit

- The Debian Manual - The system initialization ()
- Wikipedia: Init ()
- start-stop-daemon: --exec vs. --startas ()
- Getting a Python script to run in the background (as a service) on boot ()

systemd

- Getting Started with systemd on Debian Jessie ()
- systemd for Administrators ()
- How To Use Systemctl to Manage Systemd Services and Units ()