Arch Linux on the Raspberry Pi

Ryan Matlock, Magzor Corp.*

2014/07/10

Abstract

Arch is a Linux distribution built around "The Arch Way,"[1] a philosophy of simplicity, code-correctness, user-centricity, openness, and freedom. Simplicity lends itself to a minimalist approach, which in turn leads to lower system resource overhead—exactly what one wants in an embedded system. Code-correctness means that the software is clean, correct, and simple, which implies a greater degree of comprehensibility and predictability, albeit sometimes accompanied with a steeper learning curve. User-centricity, not to be confused with user-friendliness, manifests itself as giving the user complete control over their system. Openness and freedom allow for greater control of the system; as Arch Linux's founder, Judd Vinet said, "[Arch Linux] is what *you* make it."

Contents

*Last commit: `adc7a35` by `Ryan Matlock` on `2014-07-10 17:27:00 -0700`

[1] https://wiki.archlinux.org/index.php/The_Arch_Way

1 Overview

This guide aims to show you how to

1. install Arch Linux for Raspberry Pi onto a blank SD card,
2. expand the root partition to fill the disk,
3. add a new user,
4. modify user groups and grant superuser privileges,
5. establish wireless connectivity,
6. enable SSH access,
7. install GNU Compiler Collection (GCC),
8. install Python 3,
9. ***INSTALL IPYTHON*** !!!!!!
10. install Git,
11. install GNU Emacs 24+,
12. set up Emacs,
13. install WiringPi library,
14. install pigpio library,
15. install RPi.GPIO library,
16. ...
17. ??? install watchdog dæmon – reboots Pi on failure http://pi.gadgetoid.com/article/who-watches-the-watcher
18. ??? install Lynx (text-based web browser)

For now, I'm assuming you're familiar with basic *nix commands like `ls`, `cd`, `mkdir`, etc, although eventually I'd like to create a small appendix to get novices up to speed.

2 Syntax Guide

In order to avoid any confusion, here's a brief overview of the special syntax—all of which is assumed to be rendered in `typewriter text`—used in this document:

Table 1: Syntax guide

Description	Example	Meaning
bracketed purple slanted text	`<username>`	something to be entered by the user, the exact choice of which is up to them (note that brackets are to be omitted)
green text	`n`	exact user input, often found in a large block of prompts and outputs
bracketed green text	`<RETURN>`	special key input
red hook right arrow	↪	line continuation character (i.e. in actual input/output, there is no linebreak)
plain text following `$` or `#` (shell prompt)	`reboot`	text between shell prompt and end of line should e entered by the user
slanted text	*emphasized*	emphasized text
bold sepia text	**Emphasized!**	strongly emphasized text (generally to highlight a letter that you might otherwise miss)
bracketed dark blue slanted text	`<cwd>`	

3 Configuring Arch: The Hard Way

3.1 Installation

Download the Arch Linux disk image from http://archlinuxarm.org/platforms/armv6/raspberry-pi and follow the instructions. (Note: for Mac OS X[2], the process is a little different[3]:

1. Plug in your SD card and run

   ```
   $ diskutil list
   ```

   to find the `/dev/diskN` node (e.g. `disk3`, which is the `sdX` in the linked instructions) on which it's located.

2. Unmount the drive by running

   ```
   $ diskutil unmountDisk /dev/diskN
   ```

   which will print

   ```
   Unmount of all volumes on diskN was successful
   ```

   if successful.

3. Write the Arch image by running

   ```
   $ dd bs=1m if=/path/to/ArchLinuxARM*-rpi.img of
      ↪ =/dev/rdiskN
   ```

   as root[4]. Personal testing revealed that

   tested on identical Class 4, 4 GB SD cards:

   ```
   matlocksmacbook:~ matlock$ sudo dd bs=1m if=~/
      ↪ Downloads/ArchLinuxARM-2014.04-rpi.img of=/
      ↪ dev/disk4
   1870+0 records in
   1870+0 records out
   ```

[2] The `bash` terminal is assumed to be used, so user input lines are started with `$`. Later, the `tty` prompt of Arch will start user input lines with `#`.

[3] source: http://www.embeddedarm.com/support/faqs.php?item=10

[4] Some guides recommend using `of=/dev/diskN` instead of `of=/dev/diskN` for increased security as `rdiskN` is the raw path, while `diskN` is a buffered device. (source: http://elinux.org/RPi_Easy_SD_Card_Setup#Flashing_the_SD_card_using_Mac_OS~X)

```
1960837120 bytes transferred in 452.680379 secs
   ↪ (4331615 bytes/sec)
```

```
matlocksmacbook:~ matlock$ sudo dd bs=1m if=~/
   ↪ Downloads/ArchLinuxARM-2014.04-rpi.img of=/
   ↪ dev/rdisk4
Password:
1870+0 records in
1870+0 records out
1960837120 bytes transferred in 394.117681 secs
   ↪ (4975258 bytes/sec)
```

3.2 Expanding the Root Partition

When you first boot up the Pi with a fresh Arch Linux installation, you will eventually be greeted with something like

```
Arch Linux 3.10.35-1-ARCH (tty1)

alarmpi login:
```

for which the username and password are simply `root`.

1. Begin[5] by logging in as `root`.
2. Run `fdisk` on the SD card with

   ```
   # fdisk /dev/mmcblk0
   ```

3. Print the partition table, which looks something like the following[6]:

   ```
   Command (m for help): p

   Disk /dev/mmcblk0: 3.7 GiB, 3965190144 bytes, 7744512
      ↪ sectors
   Units: sectors of 1 * 512 = 512 bytes
   Sector size (logical/physical): 512 bytes / 512 bytes
   ```

[5] source: http://jan.alphadev.net/post/53594241659/growing-the-rpi-root-partition

[6] This example was performed on a 4 GB class 4 SanDisk SDHC card. With the exception of the `Disk` and `Disk identifier` entries, all the numbers are in agreement with those posted on the previously referenced Jan's Stuff "Growing the RPi root partition" blog entry (but that concerned a 32 GB disk, and the identifier is presumably unique).

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x417ee54b

Device          Boot  Start       End   Blocks  Id System
/dev/mmcblk0p1         2048    186367    92160   c W95 FAT32
   ↪  (LBA)
/dev/mmcblkp2        186368   3667967  1740800   5 Extended
/dev/mmcblkp5        188416   3667967  1739776  83 Linux
```

The first partition is the boot partition. The second is an extended partition used to overcome the 4 primary partition limit. The third partition—that is, partition 5—is contained within partition 2, and holds only 849.5 MiB[7], which is only a fraction of the disk's available space.

4. Now we must delete partition 2:

```
Command (m for help): d

Partition number (1,2,5, default 5): 2
Partition 2 has been deleted.
```

If you print the partition table (i.e. enter `p`), you'll see that partition 5 is also gone because it was contained within partition 2.

5. We will now recreate the extended partition. Add a new partition in the following manner[8]:

```
Command (m for help): n

Partition type:
  p  primary (1 primary, 0 extended, 3 free)
  e  extended
Select (default p): e
Partition number (2-4, default 2): 2
```

[7]Note the distinction between MiB (1 mebibyte = $1024 \cdot 1024$ bytes) and MB (1 megabyte = 10^6 bytes). I've tried to be consistent in this document, but mistakes have a way of creeping in, and it's ultimately not terribly important.

[8]Rather than pressing ⟨RETURN⟩ where indicated, you could manually enter the number, make a mistake, and *ruin everything*, but I think the former way is easier since the latter still involves pressing ⟨RETURN⟩.

```
First sector (186368-7744511, default 186368):
<RETURN>
Last sector, +sectors or +size{K,M,G,T,P}
   ↪ (186368-7744511, default 7744511):
<RETURN>
Created a new partition 2 of type 'Extended' and
   ↪  a size of 3.6 GiB.
```

The extended partition has now been created, but this time it occupies the disk space not taken up by the boot partition.

6. The root partition will now be recreated following a similar process. For the sake of brevity, I won't detail each step but instead show it done all at once.

   Note: it is *absolutely critical* that the first block of the old and new partition match. The data within the old partition is still there; all we're doing is resizing the partition while keeping its data intact. Changing the starting block can (and almost assuredly will) render useless the data we want to preserve.

```
Command (m for help): n

Partition type:
  p  primary (1 primary, 1 extended, 2 free)
  l  logical (numbered from 5)
Select (default p): l

Adding logical partition 5
First sector (188416-7744511, default 188416):
<RETURN>
Last sector, +sectors or +size{K,M,G,T,P}
   ↪ (188416-7744511, default 7744511):
<RETURN>
Created a new partition 5 of type 'Linux' and of
   ↪  size 3.6 GiB.
```

   Success!

7. Well, not so fast. We haven't actually written any of our changes yet, and we also want to make sure that we got the first block of our root

partition right (see the note in step 6).

To do that, print the partition table:

```
Command (m for help): p

Disk /dev/mmcblk0: 3.7 GiB, 3965190144 bytes, 7744512
   ↪ sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x417ee54b

Device          Boot   Start       End  Blocks  Id System
/dev/mmcblk0p1          2048    186367   92160   c W95
   ↪ FAT32 (LBA)
/dev/mmcblk0p2        186368   7744511 3779072  5 Extended
/dev/mmcblk0p5        186416   7744511 3778048 83 Linux
```

Looks like everything checks out, so write the table to disk and exit (and don't worry about the failure warning):

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Re-reading the partition table failed.: Device
   ↪ or resource busy

The kernel still uses the old table. The new
   ↪ table will be used at the next reboot or
   ↪ after you run partprobe(8) or kpartx(8).
```

8. Reboot the system:

```
# reboot
```

9. When the system restarts, log back in as `root`.

10. (optional) We will use `resize2fs` to actually resize the partitions, but first, let's run `df` and see what our filesystem looks like currently (displayed in an abbreviated form):

```
# df
Filesystem       1K-blocks   Used Available Use% Mounted on
/dev/root          1679632 441176   1135084  28% /
```

```
...
/dev/mmcblk0p1       91962  25328     66634  28% /boot
...
```

11. Now it's time to use `resize2fs`:

```
# resize2fs /dev/mmcblk0p5
resize2fs 1.42.9 (28-Dec-2013)
Filesystem at /dev/mmcblk0p5 is mounted on /; on
   ↪ -line resizing requited
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mmcblk0p5 is now 944512
   ↪ blocks long.
```

12. (optional) Finally, we'll run a quick check with `df` to see how our filesystem looks now:

```
# df
Filesystem       1K-blocks    Used Available Use% Mounted on
/dev/root          3688608  442024   3065496  13% /
...
/dev/mmcblk0p1       91962   25328     66634  28% /boot
...
```

    Now only 13% of the root partition is being used instead of 28%, which is a quick and easy sanity check.

3.3 Enabling Wireless Connectivity

Because of Arch's minimalist approach, very little software is included out of the box—not even something as common as `sudo`! As a result, establishing an internet connection so that additional packages can be downloaded is a high priority in any new Arch installation. In this case, we're going to assume a wireless connection is being used[9], specifically a USB wifi adapter[10].

1. First, the wireless device driver must be determined to be correctly installed. The Arch wiki suggests checking the output of

```
# lsusb -v
```

[9] sources: https://wiki.archlinux.org/index.php/Wireless_network_configuration and http://raspberrypi.stackexchange.com/questions/7987/wifi-configuration-on-arch-linux-arm

[10] In my particular case, I'm using an Edimax EW-7811Un 802.11n USB wifi adapter

but it appears to produce screenfuls of output that would only be helpful to the kind of person not reading this document. The other suggestion is to look at the output of

```
# dmesg | grep usbcore
```

which for me output a few lines, one of which was

```
[    9.216794] usbcore: registered new interface
   ↪ driver rt18192cu
```

which is what the wiki said to expect.

2. Check the output of

```
# ip link
```

which in my case has five entries, the last of which is what we're looking for (i.e. something starting with a w, like `wlan0`):

```
...
5: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc
   ↪ noop state DOWN mode DEFAULT group default
   ↪ qlen 1000
    link/ether 80:1f:02:bf:11:68 brd ff:ff:ff:ff
       ↪ :ff:ff
```

3. Run

```
# ip link set wlan0 up
```

If you see the message

```
SIOCSIFFLAGS: No such file or directory
```

your wireless device requires firmware to be properly installed and configured. Since everything seems to be working for me, you can work out the details for yourself if you've encountered an issue at this point.

4. Copy a `netctl` profile[11]:

```
# cp /etc/netctl/examples/wireless-wpa /etc/
   ↪ netctl/<profile>
```

[11] In this case, I'm assuming that you're connecting to a network with WPA/WPA2 encryption. If you want to see what other examples are available, run `# ls /etc/netctl ↪ /examples`.

5. Edit the newly-copied <*profile*> with `nano` or `vi`[12]:

```
# nano /etc/netctl/<profile>
```

so that it looks something like

```
Description='<a descriptive phrase>'
Interface=<wifi card name>
Connection=wireless
Security=wpa

IP==dhcp

ESSID='<wifi network name>'
# Prepend hexadecimal keys with \"
# If your key starts with ", write it as '""<key>"'
# See also: the section on special quoting rules in
    ↪ netctl.profile(5)
Key='<wifi password>' # pick something strong that
    ↪ incorporates letters and numbers like 'password123'
# Uncomment this if your ssid is hidden
#Hidden=yes
```

where <*wifi card name*> is probably something like `wlan0`.

6. Start your `netctl` profile:

```
# netctl start <profile>
```

where <*profile*> is simply the profile name and *not* the full path, which will result in `netctl` exiting with an error code.

7. Enable `netctl` profile to run on startup[13]:

```
# netctl enable <profile>
```

3.3.1 Additional `netctl` Information

There are a few more helpful commands to know with `netctl`. In the event you encounter an error, you can generate status logs with

[12] There may be other editors included in Arch, but `# which ed` and `# which emacs` both failed, so I'm guessing our choices are very limited. For the record, between `nano` and `vi`, I recommend `nano`.

[13] Technically, this enables a `systemd` service that starts on startup, but why bother with the details when you're using Arch?

```
# journalctl -xn
```

and

```
# netctl status <profile>
```

If you modify your `netctl <profile>`, the changes do not propagate to the service file until you run

```
# netctl reenable <profile>
```

3.4 Adding a User

It's generally considered unsafe to log in as root[14], so we will add a user[15]. To see what users currently exist, run

```
# cat /etc/passwd
```

which lists users in the format

```
account:password:UID:GID:GECOS:directory:shell
```

where `UID` is the user ID, `GID` is the primary group ID, `GECOS` is an optional field usually containing the full user name, `directory` is the path of `$HOME`, and `shell` is the user's command interpreter, which defaults to `/bin/sh`.

Adding a user is straightforward, and uses the following syntax:

```
# useradd -m -g <initial group> -G <additional
  ↪ groups> -s <login shell> <username>
```

We'll worry about groups in the next section, so for now enter something like

```
# useradd -m -s /bin/bash matlock
```

although I generally suggest you pick a different username unless you're a relative or an Andy Griffith fan.

To change the password, enter

```
# passwd <username>
```

which in my case is set to **********.

To force a user to change this password on their first login, run

[14] see http://www.slackbook.org/html/shell.html and http://lmgtfy.com/?q=why+shouldn%27t+you+log+in+as+root

[15] source: https://wiki.archlinux.org/index.php/users_and_groups

```
chage -d 0 <username>
```

(Yes, that's right, it's `chage`, not change—remember that `chage` deals with password *age*, not password change.)

The `GECOS` field is edited by issuing the command

```
# chfn <username>
```

but doing so is not especially important.

If you're ever curious as to what user you are, it's a simple as

```
# whoami
```

which may be among the least arcane Linux commands.

To switch between users,

```
# logout
```

3.5 User Groups

To add a user to a group or groups, run

```
# usermod -aG <additional groups> <username>
```

Note that if the `-a` flag is omitted, the user is removed from all groups not explicity named in `<additional groups>`. For the sake of clarity, here are the groups to which I added `matlock`:

```
# usermod -aG users,rfkill,wheel matlock
```

None of documentation I found explictly stated that `<additional groups>` is a list of groups separated by commas without spaces, but that's probably obvious to most people.

You can verify that you've properly assigned groups to a user with the command

```
# groups <username>
```

Before you go about adding a user to a group, it's helpful to know what groups exist, the purpose of existing groups, and how to create/delete groups.

First, listing groups is similar to listing users; it's simply

```
# cat /etc/group
```

The main groups we care about are `users`, `rfkill`, and `wheel`. Granting membership to `users` is a nice thing to do, `rfkill` lets you turn off RF devices (e.g. Bluetooth and wifi), and `wheel` is the typical name for the administrative group.

3.6 Installing and Configuring `sudo`

`sudo` grants the temporary privilege of `root` to a non-root user and logs all commands and failed access attempts. Working as a non-root user ensures that you're less likely to cause damage to your system through a mistake or a bug in a command you execute.

When you run a command through `sudo`, you'll receive the following warning:

```
We trust you have received the usual lecture from the local
   ↪ System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.
```

It's a message that bears repeating.

1. Install[16] `sudo`:

   ```
   # pacman -S sudo
   ```

2. Print the current `sudo` configuration:

   ```
   # sudo -ll
   ```

   and do nothing with that information. Hey, the wiki said to do it. The output for me was as follows:

   ```
   User root may run the following commands on alarmpi:

   Sudoers entry:
       RunAsUsers: ALL
       Commands:
           ALL
   ```

[16] source: https://wiki.archlinux.org/index.php/sudo

3. The configuration file for `sudo`, `/etc/sudoers`, should ***always*** be edited with `visudo` because it locks the `sudoers` file, saves edits to a temporary file, and checks the syntax before copying it back to `/etc/sudoers`.

   Unfortunately but unsurprisingly, the default editor for `visudo` is `vi`. You can fix this problem with

   ```
   # EDITOR=nano visudo
   ```

   which will open `sudoers` for editing.

   Uncomment the line

   ```
   # %wheel ALL=(ALL) ALL
   ```

   to give members of the `wheel` group `sudo` privileges.

   To permanently set `nano` as your `visudo EDITOR`, add the following to your `sudoers` file:

   ```
   # Reset environment by default
   Defaults env_reset
   # set default EDITOR to nano, and do not allow
      ↪ visudo to use EDITOR/VISUAL
   Defaults editor=/usr/bin/nano, !env_editor
   ```

   Upon saving and exiting, running `# sudo -ll` again produces

   ```
   User root may run the following commands on alarmpi:

   Sudoers entry:
       RunAsUsers: ALL
       Commands:
           ALL
   Sudoers entry:
       RunAsUsers: ALL
       Commands:
           ALL
   ```

   which isn't necessarily what you'd expect.

At this point, I highly encourage you to log out as `root` and log in as the user you created. Because we set `bash` as our default terminal, lines will now start with `$` instead of `#`.

3.6.1 Disabling `root` Login

https://wiki.archlinux.org/index.php/sudo#Disable_root_login (actually detail how to do this in here)

3.7 Enabling SSH Access

1. Now that we have an internet connection, we should sync our repo database and upgrade our system:

   ```
   $ sudo pacman -Syu
   ```

   which should ensure that `openssh` is up-to-date.

2. Check `systemctl`[17] to see if an SSH dæmon is running[18]:

   ```
   $ systemctl list-units | grep -i 'ssh'
   ```

   which for me produced

   ```
   sshd.service          loaded active running   OpenSSH Daemon
   ```

3. Edit the SSH dæmon configuration file[19]:

   ```
   $ sudo nano /etc/ssh/sshd_config
   ```

   and add the lines

   ```
   AllowGroups users wheel
   PermitRootLogin no
   ```

   If you want a custom greeting, add the line

   ```
   Banner /etc/issue
   ```

   and edit the file `/etc/issue` (you'll need root permissions).

[17] `systemctl` seems to have replaced `initscripts`, so ignore any advice that says you need to look at your `/etc/rc.conf` file, which you probably don't even have!

[18] Trust me, you don't want to leave off the `| grep -i 'ssh'`—it's a bit much to read through. Also, the `-i` isn't strictly necessary in this case, but it gives you a case-insensitive search, which is nice.

[19] If you omit `sudo`, you won't have the necessary permissions to edit the file. In case you forget (as I did upon doing this myelf), you can save it elsewhere (e.g. `./sshd_config`, and if you're unsure what your current working directory is, run `$ pwd`), exit `nano`, and run `$ sudo mv ./sshd_config /etc/ssh/sshd_config`.

4. (optional) To change the name of your Raspberry Pi[20]:

```
$ sudo hostnamectl set-hostname <new name>
```

That's it! You can now connect to your Pi[21] by running the following on a computer connected to the same network as your Pi[22]:

```
$ ssh -p 22 <username>@<hostname>
```

where <hostname> is `alarmpi` by default or <new name> if you elected to change it.

Alternately, if you know your Pi's IP address, you can connect to it with[23]

```
$ ssh -p 22 <username>@<IP address>
```

You can supposedly make your life more convenient with an `ssh_config` file. If that sounds appealing, see Simplify your life with an SSH config file[24].

3.7.1 SSH Keys Versus Password Logins

Using SSH keys is more secure[25], but for now, it's easier to go with password logins.

3.8 Making Use of SSH

Now we've set up SSH, I'd suggest that's how you connect to your Pi because it's just easier.

If you want to figure out how to copy files, see the subsection on Emacs Setup (§ 3.9.8).

[20] source: https://wiki.archlinux.org/index.php/Network_configuration#Set_the_hostname

[21] If you've changed your Pi's hostname but previously connected to it through SSH before doing so, the computer through which you're trying to connect will likely have a line in `$HOME/.ssh/known_hosts` that prevents you from connecting through the new name. (source: https://bbs.archlinux.org/viewtopic.php?id=168055) In my case, running `$ dscacheutil -flushcache` fixed the issue.

[22] I'm assuming you're connecting through port 22

[23] Again, assuming that you're going through port 22.

[24] http://nerderati.com/2011/03/simplify-your-life-with-an-ssh-config-file/

[25] source: https://wiki.archlinux.org/index.php/SSH_Keys

3.9 Recommended Software

The following subsections make heavy use of Arch's package management system, `pacman`.

3.9.1 Installing the GNU Compiler Collection (GCC)

Installing `gcc` is as simple as

```
$ sudo pacman -S gcc
```

To verify your `gcc` installation works, type the following into a file named `hello-world.c`

```
#include <stdio.h>

void main()
{
    printf("Hello, world!\n");
    return;
}
```

run

```
$ gcc hello-world.c -o hello-world.app
$ ./hello-world.app
```

Congratulations! If you haven't already, you can now add "C programmer" to your résumé.

3.9.2 Installing Python 3

```
$ sudo pacman -S python
```

If you also want Python 2, replacing `python` with `python2` is all you need to do.
To test your Python installation, run

```
$ python
```

you should see something like

```
Python 3.4.0 (default, Apr 27 2014, 10:47:09)
[GCC 4.8.2 20131219 (prerelease)] on linux
Type "help", "copyright", "credits" or "license" for
   ↪  more information.
```

```
>>>
```

Of course, it's more satisfying if you try a hello world, which is just

```
>>> print("Hello, world!")
```

3.9.3 Installing GNU Make

```
$ sudo pacman -S make
```

3.9.4 Installing Git

```
$ sudo pacman -S git
```

3.9.5 Installing WiringPi

There doesn't appear to be an Arch Linux ARM-compatible packaged release of WiringPi, so this installation process looks a little different. First, clone WiringPi using `git`[26]:

```
$ git clone git://git.drogon.net/wiringPi
```

then go into the folder and build the library:

```
$ cd wiringPi
$ ./build
```

(I still need to look up where the installation puts everything to ensure that

```
$ rm -r ~/wiringPi
```

is safe to do, but I have a feeling that because `./build` invokes `sudo`, everything is put in a place that requires special permission, that is, not your home folder.)

3.9.6 Installing I2C Tools

```
$ sudo pacman -S i2c-tools
```

Looks like you get Python 2 for free with this. I'm not sure how you invoke this, but according to Sonny, we need it.

[26] source: https://projects.drogon.net/raspberry-pi/wiringpi/download-and-install/

3.9.7 Installing GNU Emacs

```
$ sudo pacman -S emacs
```

Be forewarned: Emacs is quite large—over 330 MiB when I installed it—but it's worth it.

3.9.8 Emacs Setup

I recommend actually ignoring this section for the time being.
Emacs is a very personal thing, but my general recommendation is to try

```
$ mkdir ~/elisp
$ mkdir ~/.emacs.d
```

through SSH, and then, in another terminal, try

```
<remote-comp>: <cwd> <remote username>$ scp -P 22 ~/.emacs <
   ↪ username>@<hostname>:~/
<remote-comp>: <cwd> <remote username>$ -scp -r -P 22 ~/elisp
   ↪ /* <username>@<hostname>:~/elisp/
<remote-comp>: <cwd> <remote username>$ -scp -r -P 22 ~/.
   ↪ emacs.d/* <username>@<hostname>:~/.emacs.d/
```

which in my case looks like

```
matlockmbp:~ matlock$ scp -P 22 ~/.emacs matlock@archrpi:~/
matlockmbp:~ matlock$ scp -r -P 22 ~/elisp/* matlock@archrpi
   ↪ :~/elisp/
matlockmbp:~ matlock$ scp -r -P 22 ~/.emacs.d/*
   ↪ matlock@archrpi:~/.emacs.d/
```

3.10 File system visualization tools

3.10.1 `tree`

This lightweight program gives a nice view of folders and files

```
$ sudo pacman -S tree
```

3.10.2 GNU Midnight Commander

Midnight Commander is an orthodox file manager (whatever that means) that lets you do many cool things that even I don't know about just yet.

```
$ sudo pacman -S mc
```

4 Configuring Arch: The Easy Way

Haha, there is no "easy" way (yet—I'm working on it).

5 Using your newly-configured Arch device

5.1 Configuring Git

1. Tell `git` your name[27] with

   ```
   $ git config --global user.name "<your name>"
   ```

   and your email address with

   ```
   $ git config --global user.email "<your email>"
   ```

2. Generate SSH keys for `git`[28]

   (a) Check if you have SSH keys (you won't yet, but check anyway)

   ```
   $ ls -al ~/.ssh
   ```

   If you actually have something, that's weird. If you want to do this from scratch, I'd suggest doing something like

   ```
   $ mv ~/.ssh ~/existing-ssh-files
   ```

   and then it's like you're starting fresh, but you don't lose something that you potentially need.

   (b) Generate a new SSH key

   ```
   $ ssh-keygen -t rsa -C <your email address>
   ```

   You'll be prompted to enter a passphrase, and because this involves Github and important code, *you should actually choose a good password.* Seriously.

   > ! In fact, if you don't have a password management system alredy, I recommend some form of KeePass. As of this writing, my personal computer runs OS X, and I've had absolutely no problems with KeePassX. Better still, I keep my passwords on Dropbox, so they're synced on my phone viewable with KeePass2Android.

[27] source: https://help.github.com/articles/set-up-git
[28] source: https://help.github.com/articles/generating-ssh-keys

(c) Add your new key to the `ssh-agent`

```
$ eval "$(ssh-agent -s)"
Agent pid (some number)
$ ssh-add ~/.ssh/id_rsa
```

(d) Now you need to add your SSH key to GitHub. The way the linked article recommends doing it is all well and good if you're using your local machine, but since you're SSH'd into the Pi, things are a little different. I recommend

```
$ cat ~/.ssh/id_rsa.pub
```

and copying the resulting key text to your clipboard. The guide warns that it's important that the key is copied exactly without extra whitespace or newline characters, so copy from the beginning of `ssh-rsa` to the end of <*your email*>. From there, go to your GitHub account, click on Account Settings (the screwdriver and wrench symbol in the upper right), click on SSH Keys on the left, click Add SSH Key, paste the key into the field, Add Key, and confirm with your GitHub password.

(e) Test that this works by entering

```
$ ssh -T git@github.com
```

It'll prompt you about connecting despite being unable to establish the authenticity of github.com, but that's ok. Type `yes` and you'll see

```
Hi <your GitHub username>! You've
   ↪ successfully authenticated, but GitHub
   ↪ does not provide shell access.
```

if everything is working.

Now that everything is set up with Git and GitHub, you're ready to clone a repo and get to work!

6 Miscellaneous Useful Commands

6.1 Shutdown and reboot as user other than `root`

Shutdown:

```
$ systemctl poweroff
```

Reboot

```
$ systemctl reboot
```