Arch Linux on the Raspberry Pi

Ryan Matlock, Magzor Corp.

2014/05/06

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

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

1 Overview

This guide aims to show the reader 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 WiringPi library,
10. install pigpio library,
11. install RPi.GPIO library,
12. install GNU Emacs 24+,
13. set up Emacs,
14. ...
15. ??? install watchdog dæmon – reboots Pi on failure http://pi.gadgetoid.com/article/who-watches-the-watcher
16. ??? install Lynx (text-based web browser)

2 Syntax Guide

In order to avoid any confusion, here's a brief overview of the special syntax 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 be entered by the user

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

[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

```
$ 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
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

[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)

```
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
   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.

[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).

[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.

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
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

[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⟩.

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
   ```

   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`):

[9] source: https://wiki.archlinux.org/index.php/Wireless_network_configuration

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

```
...
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.

3.4 Adding a User

It's generally considered unsafe to log in as root[11], so we will add a user[12]. 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
```

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

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

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

```
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 and `sudo`

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`.