Installing and configuring Arch Linux ARM on a Raspberry Pi

Marc van der Sluys
HAN University of Applied Sciences
Arnhem, The Netherlands
http://han.vandersluys.nl

February 22, 2021

Contents

1 Introduction

1.1 Arch Linux

Arch Linux [1] is one of the several hundred *Linux distributions* or *Linux flavours* in existence [2]. It sits near the top of my list of favourite Linux distros for at least two reasons. First, Arch is designed to be very flexible — you can use it in a lean form for a low-specs (headless) embedded system, use it on your PC or laptop with a nice desktop environment, run a heavy server, and many things in between. Seven out of nine of the laptops, PCs and barebones I currently administer run Arch Linux.[1]

Second, Arch has the *Arch User Repository* or *AUR* [3], where users can add packages themselves. If you need a package that is missing from the official repositories, you can add it to the AUR. The benefits over installing it manually is that the package will be installed by your package

[1] I'm a geek, so I run Gentoo Linux on my own laptop, because it is even more flexible than Arch. My mother in law currently runs Kubuntu until I find an opportunity to replace it with Arch.

manager, which takes care of dependencies and keeps track of which files belong to each package. Hence, you can fully remove or overwrite all files once you want to uninstall or update the package and there is no chance that you will overwrite other packages' files or vice versa. Also, other users (among which your colleagues, students and clients) can use the package and give feedback if there is room for improvement. Updating your package to a newer upstream version is often as simple as updating the version number.

1.2 Arch Linux ARM

Arch Linux ARM [4] is a fork of Arch Linux and shares these benefits. While the latter focuses on x86_64 architectures,[2] Arch ARM is adapted for ARM processors. Although there may be differences in the packages that are available for each architecture, the main difference between the *x86* and ARM distributions is the installation procedure.

In this document, I give step-by-step instructions to install a basic Arch Linux ARM system on a Raspberry Pi v4.[3] I have not made up these steps myself, but copied them selectively from the Arch Linux ARM website [4] and Arch Linux website [1], leaving out or adding steps or explanations where I deemed it useful or necessary. The instructions for the basic installation can be found on [4] by navigating through *Platforms* > *ARMv8* > *Broadcom*, *Raspberry Pi 4* > *Installation (ARMv7 installation)*.[4] In case of uncertainty, please consult the cited websites.

Arch Linux (ARM) needs the *command-line interface* (cli) or *shell* in order to be installed. If your knowledge of the (bash) shell is rusty, or you would like to improve your skills, please consult *Efficient use of the Linux command line in the Bash shell* [6]. Of course, you can always find the manual of the commands we type in the **man pages**, by typing `man <command>`. Quit a man page by pressing `q`. See [6] for more details, *e.g.* on how to make the man pages easier to read.

Below, the dollar sign `$` indicates the command prompt — it should not be typed itself, but the command that follows it should. A hash (`#`) indicates the start of a comment. Everything after it gives more information on the command, but is not executed. There is no point in typing this text in your terminal.

2 Preparing the SD card

Preparing the SD card took me nearly 30 minutes, mainly waiting for data to be copied to the SD card. The actual time will depend on the speed of your internet connection, card reader and SD card. I have an old laptop, so my card reader may be relatively slow.

For these steps, you have to be root or superuser by issuing *e.g.*

```
$ su
```

or

```
$ sudo bash
```

and giving the root password. Alternatively you can prefix *every* command with `sudo`.

2.1 Identifying your SD card

Insert the SD card in your PC, and find how the system identifies it. In my case:

[2] The i686 architecture was supported by mainstream Arch until November 2017 and forms a separate branch since then: Arch Linux 32 [5].

[3] And v2 and v3.

[4] For the RPi v2, use *Platforms* > *ARMv7* > *Broadcom* > *Raspberry Pi 2* > *Installation*, for v3 *Platforms* > *ARMv8* > *Broadcom* > *Raspberry Pi 3* > *Installation (ARMv7 installation)*.

```
$ dmesg | tail -30 | less  # [G] mmc0:  new high speed SDHC card at address ...
  mmcblk0:  mmc0:0007 SD16G 14.5 GiB

$ cat /proc/partitions  # mmcblk0 / mmcblk0p1
```

- Apparently my device is called **`mmcblk0`**, and has one partition called **`mmcblk0p1`**. Hence, the device can be found at **`/dev/mmcblk0`**. Below, replace `mmcblk0` with the name of your device.
- **Note: your SD card is almost certainly NOT called `/dev/sda`, `/dev/sda1`**, which is probably your hard disc or SSD.
- Your SD card could be one of **`/dev/sdb*`, `/dev/sdc*`** etc. (especially when using a USB dongle for your SD card), but this could also be a second disc! To verify this, remove and reinsert the SD card and look at `/proc/partitions`: the device that is listed when the SD card is present and is not listed when it is removed, is probably your SD card.
- If your SD card is *e.g.* `/dev/sdb`, the physical *device* is called `sdb`, while the *partitions* are called `sdb1`, `sdb2`, etc.
- If using an internal SD-card reader does not work, try an external USB dongle.

In the next step, we will **erase all data**, so make sure you are not wiping your hard drive!

2.2 Partitioning your SD card

Here we replace the existing partitions with the two partitions we need (a boot partition of 200 Mb for the boot loader and kernel, and a root partition spanning the rest of the SD card for the system). For this, we use **`fdisk`**. Make sure you edit your (empty) SD card, *not* your hard disc (typically called `sda`, `sdb`, *et cetera*.), since **all existing data will be wiped**. This step takes ~ 1 min.

```
$ fdisk /dev/mmcblk0  # Make sure you use *your* device name, and not your HD!
```

- Press [o] # create a new DOS partition table
- Press [p] # print the (empty) partition table
- Press [n] # create the first new partition
  - Press [Enter] # p: partition type is primary (default)
  - Press [Enter] # partition number is 1 (default)
  - Press [Enter] # start at first sector (default)
  - Type +200M [Enter] # 200 MiB in size
- Press [t] # set the partition type
  - Press [c]: choose W95 FAT32
- Press [n] # create the second new partition
  - Press [Enter] # partition type is p (default)
  - Press [Enter] # partition number is 2 (default)
  - Press [Enter] # partition starts at first available sector (default)
  - Press [Enter] # partition ends at last sector on disc (default)

- Press [p] # print the partition table, check the result
- Press [w] # write the partition table to disc and exit

```
$ cat /proc/partitions  # Should now look like e.g. mmcblk0 / mmcblk0p1 /
  mmcblk0p2 or sdb / sdb1 / sdb2
```

2.3 Formatting your SD card

Next, we will format the two partitions we just created (*i.e.*, add a file system) and mount them:

```
$ mkfs.vfat /dev/mmcblk0p1  # Apply VFAT to the boot partition, ∼ 1 s

$ mkfs.ext4 /dev/mmcblk0p2  # Apply ext4 to the home partition, ∼ 1 min

$ mkdir boot root  # make two directories

$ mount /dev/mmcblk0p1 boot/  # mount first partition in directory boot/

$ mount /dev/mmcblk0p2 root/  # mount second partition in directory root/
```

2.4 Installing the base system

We will now download, unpack and copy the base system to the two partitions we created. We require a **recent version of `bsdtar` (v3.3 or higher)** for this. **Please refer to Appendix C** on how to verify your version of `bsdtar` and, if required, install a newer version and unpack your tarball with it in step 3 below. For the Raspberry 4:[5]

```
$ bsdtar --version  # Verify that v3.3 or newer is installed!

$ wget http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-4-latest.tar.gz
  # Download the tarball, ∼315Mb

$ bsdtar -xpf ArchLinuxARM-rpi-4-latest.tar.gz -C root && sync  # Unpack base
  system to root partition; ensure that SD card is synced; ∼ 2 min[6]

$ mv root/boot/* boot  # Move the boot files to boot partition; ∼ 25 s[7]

$ emacs boot/config.txt  # Add a new line with:  hdmi_force_hotplug=1

$ emacs root/etc/hostname  # Set unique hostname to recognise your device when
  searching for its IP address later.  Don't use spaces or special characters.
  I'll call mine MarcsPi.
```

2.5 Tidying up

We can now unmount the SD card and remove it from your PC or laptop and clean up (alternatively, you could tidy up once your Pi has booted correctly):

```
$ umount boot root
```

- Remove the SD card from your PC.

```
$ rm ArchLinuxARM-rpi-2-latest.tar.gz  # Remove the tarball
```

[5] For the Raspberry Pi v3 (ARMv8) (and for v2) the **ARMv7** installation (for the Pi v2) is currently recommended, and you need to download `http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-2-latest.tar.gz`.

[6] Note that you can get warnings like `bsdtar: Ignoring malformed pax extended attribute`. Please ensure you use bsdtar v3.3 or newer. Your SD may still work, despite the warnings/errors [7].

[7] Ignore warnings along the lines of *Failed to set attributes* when copying from `ext4` to `vfat`.

```
$ rmdir boot root  # Remove the two directories
```

If all went well, we now have an SD card that can boot a bare Arch Linux ARM system on a Raspberry Pi.

3 Booting and logging in to your Pi

3.1 Booting your Raspberry Pi

You can now put your SD card in your Raspberry Pi and power it up. It should boot Arch Linux ARM and start an SSH daemon.

3.2 Logging in to your Pi on the text console

If you have a screen and keyboard attached, you can log in at the text console, using **`alarm`** for both the username and password. In that case you can skip the next section, where we assume a headless system. However, you will still need a wired network connection to install software packages.

3.3 Logging in to your Pi using SSH

Here, we will assume that you have a headless system or some other reason to log in from your PC or laptop over the network.[8] We will use SSH for this. By default, wireless networking is not installed (since it wouldn't know the password of your wireless access point anyway), so connect your Pi to an Ethernet cable.

First, we need to find the IP address of your laptop or PC. You do this with the command `ifconfig`. Your Ethernet card may be called `eth0` or have a name that starts with `en`, your wireless adapter `wlan0` or something starting with `wl`. Ignore the entry called `lo`. The IP address is shown after the word `inet`. In my case, it is `192.168.1.80` Hence, all IP addresses on my network (probably) start with `192.168.1.` (my address space).

Next, we need to discover the IP address of the Pi. You can use **`nmap`** (as root!).[9] Since all the addresses on my network start with `192.168.1.`, I will search these addresses only:

```
$ nmap -sP 192.168.1.*
Nmap scan report for MarcsPi.home (192.168.1.160)
Host is up (7.5s latency).
MAC Address:  XX:XX:XX:XX:XX:XX (Raspberry Pi Foundation)
```

Note that the IP address is printed two lines *before* the name of the manufacturer; the next line would be the first line of the next block. You may have to repeat this command if the hostname you specified earlier does not show up immediately. My Raspberry turns out to have the address `192.168.1.160`. The default user on Arch Linux ARM is **`alarm`** (and so is its password), so I can log in using[10]

```
$ ssh alarm@192.168.1.160
The authenticity of host '192.168.1.160' can't be established.
Are you sure you want to continue connecting (yes/no)?  yes  # type yes here
```

[8] For example because you can use your own laptop for typing and/or because this way, you can copy commands from this PDF file into your terminal. Remember that in most graphical environments, selecting a text also copies it, and pressing the middle mouse button or scroll wheel pastes the selection. Also remember to never copy a command into your terminal (especially as root) without understanding what it does!

[9] Alternatively, briefly use a screen and keyboard on your Pi, log in and type `ifconfig` to find o.a. your IP address. In addition, there are smartphone apps that scan your network — Fing for example.

[10] We will discuss more sophisticated ways to connect over ssh in Section 9.

```
alarm@192.168.1.160's password:   # type the password alarm here
Welcome to Arch Linux ARM
```

It is a good idea to change the default password for the user `alarm` first:

```
$ passwd  # this will ask for your old password, then the new one twice
```

It would of course be even safer to rename or replace this user account to/with one which uses a non-default user name. See Sect. 7.1 and [8] for information on user management.

If you want to log out of your SSH session (once we're done, not quite yet), you can type

```
$ exit
```

4 Configuring your system

You should **never do anything** as root or superuser, unless it is necessary. Since we are dealing with **system administration** here, the best option is to do this as root.

4.1 Changing to root

The default root password is also **`root`** and the first thing we will do is change this to something more secure. Of course you have to make sure you can **remember this password!**

```
$ su  # Become root by typing the default password 'root'
$ passwd  # Type the new root password twice
```

If you want to log out as root, and become a normal user again, you can type

```
$ exit
```

4.2 Initialising the pacman keys

We need to initialise the pacman keyring and populate the Arch Linux ARM package signing keys, so that downloaded packages can be verified before installation:

```
$ pacman-key --init
$ pacman-key --populate archlinuxarm
```

4.3 DNS problems due to incorrect date and time

The Raspberry Pi by default has no hardware clock, which means that it does not know the correct date and time at boot, but has to get them from the internet. However, since 2020, it seems that the DNS service, translating an internet address name to an IP address, no longer (always) works if the system clock is wildly inaccurate. This also means you cannot install any packages (since they must be downloaded from the internet). If this works now, you can skip this section. However, if you later get problems installing packages with the error `Could not resolve host: mirror.archlinuxarm.org` or similar, you may want to get back to this section.

To temporarily fix this, you can use (as root):[11]

```
$ timedatectl set-ntp 0
$ timedatectl set-time "yyyy-MM-dd hh:mm:ss"
```

where of course in the second line you type the (roughly) correct date and time. Now, the DNS service should work, and you can install packages.

[11] Thanks to R. Ruiterkamp for providing this information and fix

However, it is annoying to do this after every boot. To prevent this, you could create the following script (again as root) in *e.g.* `/usr/local/sbin/sync-clock-at-boot`:

```
#!/bin/sh
sleep 20  # Sleep for 20s to get an IP address first when run at boot
chronyd -q 'server 185.255.55.20 iburst'  # Sync the system clock
```

where `185.255.55.20` is the IP address of a Dutch ntp (time) server. Make your script executable with `chmod a+x /usr/local/sbin/sync-clock-at-boot`. In order to for this to work, you need to install the package `chrony` (however, you should probably switch to Section 4.4 between typing the two `timedatectl` commands above and `pacman` below):

```
$ pacman -S chrony
```

From now on, you can sync your clock by calling the script (as root; when running this manually, you can comment out the `sleep` line):

```
$ /usr/local/sbin/sync-clock-at-boot
```

However, it is nicer if this script were run at each boot (hence the `sleep` line). To do this, install `cronie`, start it and ensure it starts at boot time as explained in Sect. 5.3.

We can now create a **cron job** that runs at boot time by running (still as root; you may want to install and/or set your default editor as explained in Section 4.5 first):

```
$ crontab -e
```

which opens an editor with the **crontab** file, where you add to the bottom the line

```
@reboot /usr/local/sbin/sync-clock-at-boot
```

Save the file an exit the text editor. This should ensure that at boot time, your script waits for 20 s until you have an IP address, and then syncs the system clock.

4.4 Initialising the package manager

In Section 4.5 below, we will install our the first package. In some (many?) cases, this will result in an error (you can go there and try, and return here if this is the case for you). The steps in the current section will 'initialise' the package manager, so that `pacman` works properly.

First, we need to tell the system which package *mirrors* to download packages from. This can be done by editing the text file `/etc/pacman.d/mirrorlist`. Since your favourite editor may not yet be installed, you can use `nano`:

```
$ nano /etc/pacman.d/mirrorlist  # Uncomment the desired line(s)
```

The lines to uncomment would be for servers near you (your country or close). Servers close to the top of the list are usually better. You can quit `nano` by pressing `Ctrl-X` and pressing `Y` to confirm you want to save the file and `Enter` to keep the same name.

The second issue may be that the keys which `pacman` uses to verify package signatures are out of date. This can be remedied by

```
$ pacman-key -u
$ pacman-key --populate
```

4.5 Selecting and installing a text editor

Since Linux system administration includes a lot of text-file editing, it is useful to install your favourite text editor. In a Linux system, this is done using the **package manager**. In Arch

Linux (ARM), the **pac**kage **man**ager is simply called **pacman**. By default, `vi` and `nano` are installed. I recommend `emacs`[12], and since I don't want to install hundreds of megabytes of graphical packages as dependencies (the X server), I install the *no-X* version. Since we haven't updated the system for a while (*i.e.*, never), we should first update the package databases, so that the system knows about the latest versions:

```
$ pacman -Sy  # Update the package databases
$ pacman -Ss emacs  # Find emacs packages - should list community/emacs-nox
$ pacman -S emacs-nox  # Install the emacs-nox package
```

If you don't have a preferred text editor, I strongly suggest you use `emacs`.[13] I will type `emacs` below whenever a text editor is invoked — replace it with `vi` if you use that instead.

By default, the system will use `vi` for *e.g.* `cron` and `git`. If you want to set a different default editor, you must set the `EDITOR` environment variable. For `emacs`, do

```
$ export EDITOR='emacs'
```

However, this will only define the `$EDITOR` variable for the current session. If you want to use it always, add it to your `.bashrc`, in the case of `root`:

```
$ emacs /root/.bashrc  # Add to the last line:  export EDITOR='emacs'
```

Alternatively, if you know what you are doing (double '`>`'!) [6], do

```
$ echo "export EDITOR='emacs'" >> /root/.bashrc
```

Note the leading dot in the file name — it indicates a **hidden file**, which is not shown by `ls`, except when the `-a` option is specified. The file `.bashrc` is sourced whenever a user starts bash. Hence, your variable will be defined in every session, but only for root.

4.6 System-configuration files

4.6.1 `/etc/`

Most of the configuration of a Linux system takes place in the directory `/etc/`. Hence, I usually add the command `cd /etc/` to the last line of `/root/.bashrc`, so that it is executed every time I become root and I always start in that directory:

```
$ echo "cd /etc/" >> /root/.bashrc
```

Since this will only take effect next time we log in as root, we will have to do this manually now:

```
$ cd /etc
```

In the next sections, **I will assume that you are in this directory**. You can check this with

```
$ pwd  # Print the Working Directory
```

Most system-configuration files are plain-text files. Hence, we can use `git` to keep track of them.

[12] Unless you know `vi` and like it better.

[13] Once you know that you can edit a file with the command `emacs <filename>`, that you can access and exit the menu with `F10`, that undo is `Ctrl-/` and that you can always escape back to the editing buffer by pressing `Esc` three times, you're ready to go. Note that the menu shows the keyboard shortcuts for most actions, so that you can quickly learn them and become faster. Also, keep in mind that the options in the menu are only a *very* limited subset of the 10,000 or so commands in Emacs. See Appendix A for more hints to get started.

4.6.2 Installing and using git

System-configuration files are typically plain-text files sitting in (a subdirectory of) `/etc/`. It is therefore useful to create a **git** repository in this directory in order to track the changes that you or the system make.

First, we need to install `git`:

```
$ pacman -S git
```

Ensure that you are in the directory `/etc/`, then create a `git` repository in the current directory:

```
$ git init .
```

If you haven't used git before (as this user on this system), you have to supply a user name and email address. If this is going to be a *local* repository only, you can make something up. However, if you are planning to push your commits to a *remote* repository, for example as a backup, you should choose a name and email address that others are allowed to see. Since your configuration files may contain sensitive data, I **strongly** recommend using a **local** repository for `/etc/` if this system is vital to you!

```
$ git config --global user.name "root"
$ git config --global user.email "root@RPi"
```

Leave out the option `--global` to use these settings for this repository only.

It is good practice to add a config file to the repository before editing it the first time. Let's add (and then commit) some files that we will be adapting soon:

```
$ git add pacman.conf makepkg.conf  # Add files to repo
$ git commit -m "Add some config files to repo"  # Commit changes
```

If we later edit one of these files, we will:

```
$ emacs /etc/somepath/someconfigfile  # Edit your config file
$ git diff  # Check your changes before committing them
$ git commit -a -m "Update someconfigfile"  # Commit all your changes
$ git log  # Check your commit history
```

If you don't specify a commit message with `-m`, git will open the editor specified in `$EDITOR`. Write a single line (or a single line as title, a blank line and below that more details), save it and exit your editor. See *e.g. Git for coworkers* [9] for more basic information and links.

4.7 Setting the timezone

We need to set the correct timezone. For the Netherlands:

```
$ timedatectl set-timezone Europe/Amsterdam  # Set time zone to CE(S)T
$ timedatectl status  # Check
```

Check the current date and time:

```
$ date
```

As root you can set the current date/time if necessary (see `man date`), *e.g.*:

```
$ date -s "2025-09-25 16:49:36"
```
[14]

[14] Bonus question: what's special about these six numbers?

4.8 Installing and allowing sudo

The **sudo** command is used to allow privileged users to execute commands with root permissions, by prefixing `sudo` to the actual command. We will need this later to install packages from the Arch User Repository (AUR; see Sect. 6 and [3]):

```
$ pacman -S sudo
```

The file `/etc/sudoers` determines which users or groups have sudo access.

A simple way to allow normal users to use the sudo command is to allow every user from the group **wheel** sudo access. We can achieve this by uncommenting (by removing the leading '# ') the line **# %wheel ALL=(ALL) ALL** near the end of the sudoers file. Since the file is read-only, we will change the write permissions before and after editing:

```
$ chmod u+w sudoers  # Allow writing
$ emacs sudoers  # Uncomment line near the end:  %wheel ALL=(ALL) ALL
$ chmod a-w sudoers  # Forbid writing again
```

4.9 Configuring Pacman and AUR

The package manager **pacman** will look nicer with colours. Type *e.g.*

```
$ pacman -Ss emacs  # Search for packages containing emacs
```

We will enable colours by uncommenting the line `Color` in pacman.conf:

```
$ emacs pacman.conf  # Uncomment the line Color
```

When you try the pacman command above again, you should have a better overview of the output.

When installing packages from AUR, they need to be compiled. This is fastest if the command `make` is told to use all CPU cores. For four cores, uncomment and edit the line **MAKEFLAGS="-j4"** in **makepkg.conf**:

```
$ emacs makepkg.conf  # Set MAKEFLAGS="-j4"
```

5 System administration

5.1 Finding and installing packages

We have now set up our system, and can start using it. Of course, **system administration** doesn't stop here — we may want to install or update packages. We have already seen that we can search for packages matching <matchstring> by

```
$ pacman -Ss <matchstring>  # Find a package matching <matchstring>
```

and install the package <package> using

```
$ pacman -S <package>  # Install <package>
```

5.2 Updating your system

Even if we don't install new packages, we will want to update the system with any new versions of the packages we have already installed. In order to do this, we first need to **synchronise** the package database so see whether any new versions are available:

```
$ pacman -Sy
```

Once this is done, we can **update** all installed packages for which a newer version is available:

```
$ pacman -Su
```

In fact, we can do these two actions with one command:

```
$ pacman -Syu
```

You are advised to do system updates weekly. Waiting longer (months or more) increases the probability of complex **conflicts** between dependent packages of different versions.

5.2.1 Pacnew files

When new versions of system packages are installed, they may come with new configuration files. Since you may have changed these files, your changes would be overwritten if these config files were simply updated too. Instead, pacman prints a line like

```
warning: /etc/pam.d/usermod installed as /etc/pam.d/usermod.pacnew
```

and creates a new file with the extension `.pacnew`. It is your task as a maintainer to see whether the old file, the new file or a combination of both must be used, and then to remove the `.pacnew` file.

You can find all `.pacnew` files using (assuming you sit in `/etc/`):

```
$ find . -name "*.pacnew"
```

You then know that for each of these files, there is the original file without that suffix that must be updated. To see the differences between the two files, use:

```
$ diff -wd configfile configfile.pacnew
```[15]

You can then decide to replace the old version with the new one, or to ignore and remove the new version:

```
$ mv configfile.pacnew configfile  # Replace the old file
$ rm configfile.pacnew  # Remove the new file
```

A third option is to use your text editor to merge the two, *i.e.* copy *some* of the changes to the old file by hand and then remove the new file.

The Arch Linux Wiki [10] provides more details on how to keep track of `.pacnew` files and lists some tools that can aid you. If you don't know `vi`, `vim` or `vimdiff`, you can try `etc-update`, which can be installed from AUR (see Sect. 6.3). We discuss how to use `etc-update` in Appendix B.

5.3 Installing and starting the cron service

Cron can run commands at specified times (*e.g.* every minute, hour, day or week) or at boot, and is useful for nearly every system. Install a cron version called **`cronie`**:

```
$ pacman -S cronie
```

Cronie is a **system service** or **daemon**, which must be started. Arch Linux (ARM) uses **systemd** as service manager. We can use the command **`systemctl`** to start cronie and check its status to see whether it is indeed running:

```
$ systemctl start cronie
$ systemctl status cronie
```

[15] You could install the package `colordiff` using `pacman` and use it instead of `diff`.

In addition, we want to start cronie whenever we boot the system, by *enabling* the service:

```
$ systemctl enable cronie
```

To see which services are installed and/or started, do:

```
$ systemctl list-unit-files
```

5.4 More packages to install

Other packages you may want to install include

```
$ pacman -S bash-completion atop htop mlocate wget git colordiff
```

(use `pacman -Ss` on each of them to see what they do).

5.5 Setting up wireless networking

Start your wireless device and find your network. I'll assume here that your device is called `wlan0`. If not, replace `wlan0` with the name of your device everywhere below:

```
$ ip link  # wlan0 or similar is wireless - probably no UP between <>

$ ip link set wlan0 up  # set wireless to "up" - check with ip link

$ iwlist scan 2>&1 | less  # scan for access points - find yours
```

Most wireless access points (APs) use one of two methods to connect:

1. using a passphrase (without a user name) which is identical for all users. This is typically the case for home networks and requires a `psk="MyPassword"` line in `wpa_supplicant.conf`;
2. using a user name and password, which are unique for each user. This is often used at *e.g.* a university or large company. The `wpa_supplicant.conf` file needs two lines like `identity="user@uni.edu"` and `password="MyPassword"`.

For each entry in `wpa_supplicant.conf` (which is delimited with `network={...}`), you should choose one or the other — if you want to connect to the wireless network at home and at work, you will need two `network=` entries in `wpa_supplicant.conf`.

For both options, create a `network=` entry with

```
$ wpa_passphrase "your_SSID" "your_key" >>
  /etc/wpa_supplicant/wpa_supplicant.conf
```

Only if you're using the second option (user name + password), in addition, do[16]

```
$ echo -n MyPassword | iconv -t utf16le | openssl md4 >>
  /etc/wpa_supplicant/wpa_supplicant.conf   # Note:  UTF16LE (first the number
  one, then a lowercase L)
```

You should edit `/etc/wpa_supplicant/wpa_supplicant.conf`. If you are connecting without a user name (the first option) you may find two `psk=` lines in the relevant `network=` scope, of which one is commented out and contains your clear-text password, while the other contains

[16]Note that this leaves your clear-text password in `~/.bash_history` — if desired, edit that file after leaving the current shell and remove this line. Alternatively, before issuing the `echo...` command, do `export HISTCONTROL="ignoreboth"` and prepend a space to the sensitive command (before `echo`, so that it will not be recorded in your bash history).

the same password, but hashed. If everything works (after testing below), you could remove the line containing the clear-text password.

When using a user name and password (the second option), edit the relevant `network=` group in `/etc/wpa_supplicant/wpa_supplicant.conf` as follows:

1. remove the (two) `(#)psk=` lines;
2. move the `(stdin)= <hashed password>` line into the `network=` scope;
3. replace `"(stdin)= "` with `"password=hash:"`. The line should now read something like `password=hash:a6f851dfe732fb00ee19d512201e9a`;
4. add an `identity="user@uni.org"` line with your user name to the `network=` scope;
5. check that your `network=` entry contains at least the three lines containing `ssid=`, `identity=` and `password=`, and *no* `psk=`;

Note that for both methods, additional lines, may be required in `wpa_supplicant.conf`, depending on the AP.[17] After finalising your `wpa_supplicant.conf` with either method, test your configuration with

```
$ wpa_supplicant -D nl80211,wext -i wlan0 -c
  /etc/wpa_supplicant/wpa_supplicant.conf
```

If this works (no error), kill and restart wpa_supplicant adding -B (to run in background) and obtain an IP address using DHCP:

```
$ wpa_supplicant -B -D nl80211,wext -i wlan0 -c
  /etc/wpa_supplicant/wpa_supplicant.conf

$ dhcpcd wlan0  # Obtain an IP address
```

We need to create a system service in order to start our network on boot. You can download a template using wget:

```
$ wget http://arch.astrofloyd.org/wireless/network-wireless@wlan0.service -O
  /etc/systemd/system/network-wireless@wlan0.service

$ killall wpa_supplicant dhcpcd  # ensure wpa_supplicant & dhcpcd are killed
  before (re)starting the network

$ systemctl daemon-reload  # needed if configuration files have changed

$ systemctl start network-wireless@wlan0.service  # Start the service now

$ systemctl enable network-wireless@wlan0.service  # Enable the service at
  boot
```

5.6 More system administration and system maintenance

General recommendations on how to administer your Arch Linux (ARM) system and what to do after a fresh install can be found in the Arch Wiki [11]. A quick reminder of the most important configuration settings can be found in `man 7 archlinux`.

Tips and tricks for system maintenance can be found in the Arch Wiki [12].

Much more information on `pacman` can be found in `man pacman` and in the Arch Wiki [13].

[17] Common entries are `proto=`, `key_mgmt=`, `eap=`, `pairwise=`, `ca_cert=` and `phase2=`.

6 Installing packages from AUR

The Arch User Repository (AUR) [3] is one of the great strengths of Arch Linux. It currently (2017) contains over 40,000 packages. You can add packages yourself, which can be installed and tracked by the package manager by all Arch Linux (ARM) users. However, there is also a **risk**. Since *anyone* can add packages to AUR, so can evil minds. You should therefore always check what you are doing, and whether you are indeed installing the package you intended to install!

In any case, installing an AUR package requires some handiwork: a `PKGBUILD` script must be downloaded, and used to download, unpack, prepare, configure and compile the source code. Then, a `pacman` package is created, which can be installed by `pacman` itself. If a few dependencies are needed, this quickly becomes very tiring.

Fortunately, there are *unofficial* helpers (on AUR) that can do these tedious jobs for you. There is a long list available in the Arch Wiki [14], and though I didn't try them all, my (current) favourite is `yaourt`.

6.1 Installing `yaourt`

The program **`yaourt`** is a pacman front end that can install both official Arch and AUR packages. `Yaourt` itself is available in the AUR, and we would need `yaourt` in order to install `yaourt`. The solution is to install this one (actually two, there is the dependency `package-query`) package by hand and from then on use `yaourt` to install and update all other AUR packages. We follow the blog *Installing Yaourt on Arch Linux* [15]. First, we will install some dependencies that are part of the official Arch release and can be installed by `pacman`.

In order to install packages from AUR, we need to compile them. The necessary packages (`gcc`, `make`, etc.) are collected in the **`base-devel`** group. To save effort in case some packages from this group are already installed, we will only install the `--needed` packages:

```
$ pacman -S --needed base-devel # Press [Enter] - select all needed packages
```

We will also need:

```
$ pacman -S --needed wget yajl
```

We are now ready to install **`package-query`** from AUR. For security reasons, we do this as a **normal user**. Hence, use `exit` to log out as root, or `ssh` to log in as user to your Raspberry Pi in a different terminal.

```
$ mkdir -p ~/tempAUR/ && cd ~/tempAUR/  # Use a new directory

$ wget https://aur.archlinux.org/cgit/aur.git/snapshot/package-query.tar.gz
  # download PKGBUILD source tarball from AUR

$ tar xfz package-query.tar.gz  # unpack tarball

$ cd package-query && ls -al  # cd & see what was in the tarball (PKGBUILD)

$ makepkg  # create package from PKGBUILD

$ sudo pacman -U package-query*.pkg.tar.xz  # install package as root

$ cd -
```

Finally, we install **`yaourt`** itself in a similar manner:

```
$ wget https://aur.archlinux.org/cgit/aur.git/snapshot/yaourt.tar.gz
```

```
$ tar xfz yaourt.tar.gz
$ cd yaourt && makepkg
$ sudo pacman -U yaourt*.pkg.tar.xz
$ cd -
```

If everything worked, you can remove the temporary directory and its contents:

```
$ cd - && rm -rf tempAUR/ && cd /etc/
```

In order to use somewhat more reasonable colours than the default scheme, you can type

```
$ export YAOURT_COLORS="nb=1:pkg=1:ver=1;32:lver=1;45:installed=1;42:
grp=1;34:od=1;41;5:votes=1;44:dsc=0:other=1;35"
```

in your terminal and in your `~/.bashrc` (you only) or in `/etc/bash.bashrc` (system wide).

6.2 Installing packages using `yaourt`

We install packages using **`yaourt`** also as a **normal user**. `Yaourt` will refuse to run as root. Only for the installation phase, root permissions are needed, `sudo` will be called and your **user password** will be asked.

As an example, we will install the package `aurvote`, which you can use to vote for your favourite packages to show your appreciation to the package maintainer.

As a **normal user**:

```
$ yaourt aurvote
```

This will:

1. List all packages in Arch and the AUR that match `aurvote`. If you don't see what you were looking for, choose 0 and press `Enter`. Otherwise, choose the number(s) of the package(s) you want to install.
2. Download the `PKGBUILD` file from AUR and print recent user comments. Check the comments to see whether there are issues. `Yaourt` may ask the same question about the `install` file for some packages.
3. Warn *Unsupported package: Potentially dangerous.*
4. Ask whether to edit the `PKGBUILD`. You should always answer `Y`. This will open the `PKGBUILD` in your favourite editor (using the `$EDITOR` variable and allow you to check whether nothing fishy is going on, whether the download URL (in the line `source`) looks like what you expect, etc. Quit the editor without changing anything.
5. Ask whether you want to continue building. If you do, `yaourt` will download the source code tarball of the package (not the AUR `PKGBUILD`), and unpack, configure and compile it. Finally, it will create an Arch Linux package, which can be installed by `pacman`.
6. Ask whether you want to continue installing. You can first view the package contents to see what will be installed.
7. Ask for your `sudo` (user) password.
8. Check the system one more time and ask to proceed with the installation.

When no conflicts are found, the package is now installed and ready to use.

You will need to create an account on AUR [3]. You may want to use an anonymous username, and should not use a top-secret password, as it will be saved in plain text by `aurvote`. You can configure `aurvote`, providing the user name and password of your `AUR` account and vote for the aurvote package:

```
$ aurvote --configure
$ aurvote aurvote
```

6.3 Installing and using `etc-update`

After every (package or) system update, we should check whether any `.pacnew` files exist, and, if so, process them (see Section 5.2.1). The program `etc-update` can help you with this bookkeeping. You can `yaourt` to install the package `etc-update`. Appendix B shows the principles of running `etc-update`.

7 Security

In this section, we will take some simple measures to make your system more secure. While security may not be an urgent matter for a Raspberry Pi, it is a good idea to consider it whilst setting up or administrating a system.

7.1 User management

We can increase the security of our system by giving separate accounts to different users. Firstly, this allows us to remove the default user `alarm`, which any informed hacker knows should exist on a freshly installed system. Secondly, having multiple user accounts will prevent users from messing up each other's files. In addition, we can give root access only to those users who need it.

Since only the system administrator can do user management, we must do these actions as `root`. Typically, when creating a user account, we must of course specify a user name, but also the **groups** the user will be a member of, and whether we should create a **home directory** as well. After creating a user, we will give her a password. User and group accounting information is stored in the files `passwd, shadow, group and gshadow` in `/etc/`. If you are using git, you could add these files now and commit them. The details of user and group management can be found in the Arch Wiki page *Users and groups* [8]. See also Section 4 of *Efficient use of the Linux command line in the Bash shell* [6] for more info on users and groups.

7.1.1 Creating a new user

We use the command `useradd` to create a user. A typical command looks like

```
$ useradd -m -g maingroup -G group2,group3,groupN -s shell username
```

Here, `-m` tells the command to create a home directory, by default in `/home/username`. The flag `-g` specifies the main group the user will be a member of. Typically, this will be `users`. Other groups the user should be added to are specified with `-G`. We have before encountered the group `wheel`, which allows the user to become `root` and use `sudo`. Other groups may give access to *e.g.* video or audio output, an optical drive, *et cetera*.

Here, we will create the user `jane` with a home directory, add her to the group wheel and give her `bash` as the default shell:

```
$ useradd -m -g users -G wheel -s /bin/bash jane
```

Jane cannot yet log in, because we haven't given her a password:

```
$ passwd jane
```

You can either let Jane type her own password (twice), or select a random one for her to change at first login. See `man useradd` for more info.

You can check a user's settings, in particular the groups she belongs to, with the **`id`** command:

```
$ id jane
```

7.1.2 Modifying a user's settings

In order to modify the settings for an existing user, you can use **`usermod`**. For example, to add `jane` to the groups `audio` and `video`, do:

```
$ usermod -a -G audio,video jane
```

Note that you must use the `-a` option to **append** the new groups to the existing ones. If omitted, the existing groups will be **overwritten** and `jane` may *e.g.* lose her root access! It is therefore a good idea to run `id` before and after `usermod` in order to verify the changes and allow restoration of the original settings if necessary. See `man usermod` for more details.

7.1.3 Disabling or removing a user account

If you want to (temporarily) **suspend** a user's access to the system, but keep her home directory, you can replace her login shell with **`nologin`**:

```
$ usermod -s /sbin/nologin jane
```

(run `which nologin` to verify where `nologin` sits on your system). Rather than giving the user a shell prompt at login, she will be refused access.

Removing a user's account can be done using the **`userdel`** command. See `man userdel` for information.

7.2 Configuring the SSH daemon

Secure SHell (**SSH**) is a secure way to connect to computers over a network. It uses a client on a local machine (the **`ssh`** command) and a *daemon* called **`sshd`** on the remote side to receive connection requests. By default `sshd` listens to port 22, and until recently root login was allowed by default. Since the `root` user is available on any UNIX-like system and port 22 is a default value, many scanners and hackers may try to login on port 22 as root. Changing this behaviour is therefore a small step to a more secure system.

The `sshd` configuration is stored in `/etc/ssh/sshd_config`, so add and commit this file to your git repo if you use it. Then open it with your favourite `emacs` and edit the following lines:

```
Port 32123
PermitRootLogin no
```

If you want, you can also set the line

```
AllowUsers jane
```

to allow SSH access *only* to `jane`.[18]

Note that text after a hash (#) is a comment. Make sure the lines you edited are **uncommented**! We have set the port to a high number (the maximum is 65535), since the lower numbers are used more often by default and scanners may only scan the first 100 or 1000 ports or so.

[18] Note that for more security, you can block logins using passwords, allowing logins using public keys only.

We will need to tell `systemd` that the configuration files have changed and restart sshd:

```
$ systemctl daemon-reload
$ systemctl restart sshd
```

I usually use a new terminal on my local machine to log in using the new port rather than logging out as root:

```
$ ssh jane@123.456.789.0 -p 32123
```

7.3 Preventing the accidental overwriting of files

Especially as root, when copying or moving/renaming a file, I want to be prompted if the destination file already exists. By default, it is overwritten. I also want to be prompted when deleting a file. This can be overruled/achieved by always giving the `-i` flag to **`cp`**, **`mv`** and **`rm`**. Instead, I have redefined these commands using an **alias** in my `.bashrc` file:

```
alias rm='rm -i'
alias mv='mv -i'
alias cp='cp -ip'
```

Note that I added the `-p` flag to preserve the permissions and time stamp of a file upon copy. You can find the `.bashrc` file in you home directory (`/home/jane` for `jane` and `/root/` for root). Alternatively, you can add these lines to `/etc/bash.bashrc` to apply them to all users on the system. When you are using these aliases but don't want to be prompted for once because you have a long list of files and you *want* to overwrite the existing destination files, you can overrule this behaviour by specifying the `-f` (force) flag.

8 Setting up your bash environment

Setting up your bash prompt and in particular adding colours may seem like putting up a kitschy Christmas tree at first (it did to me), my it can actually improve the readability of your terminal dramatically. For example, after screenfulls of white error messages, you can easily find back the coloured prompt where the initial error occurred. In addition, man pages become much better readable with colours. See Section 8 of *Efficient use of the Linux command line in the Bash shell* [6] for more information on how to set up your bash environment.

8.1 Setting your bash prompt

In your `.bashrc`, add a line like

```
export PS1="[\[\033[1;31m\]\u\[\033[0m\]@\[\033[1;34m\]\h\[\033[0m\] \W]\$ "
```

This will use nice colours (the stuff between `\[` and `\]`) and display your user name (`\u`), host name (`\h`) and abbreviated current directory (`\W`). I use this as a normal user, and change the colours between different machines, so that I can quickly see when I am in a remote shell.

For root, I use eye-catching background colours to indicate that I must be doubly careful when typing something:

```
export PS1="[\[\033[1;41m\]\u\[\033[0;1;7m\]@\[\033[0;1;44m\]\h\[\033[0m\] \W]# "
```

8.2 Adding colours to the man pages

It can be hard to see the structure in **man pages**. However, if we add colours, navigating and reading them becomes much easier. Since `less` is used as the default pager, we need to configure it. In your `.bashrc` (or `/etc/bash.bashrc`), add:

```
# Colour in man pages (when using less as a pager - see man 5 termcap):
export LESS_TERMCAP_mb=$'\E[01;34m'  # Blinking -> bold blue
export LESS_TERMCAP_md=$'\E[01;34m'  # Bold (section name, cl option) -> blue
export LESS_TERMCAP_me=$'\E[0m'      # End bold/blinking
export LESS_TERMCAP_so=$'\E[01;44m'  # Standout pager -> bold white on blue
export LESS_TERMCAP_se=$'\E[0m'      # End standout
export LESS_TERMCAP_us=$'\E[01;31m'  # Underline - variables -> bold red
export LESS_TERMCAP_ue=$'\E[0m'      # End underline
export GROFF_NO_SGR=1
```

In addition, in order to not wrap long lines by default (`-S`) and make search case insensitive (`-i`) in `less`, I use the alias

```
alias less='less -Si'
```

Searching in less is done by typing `/` followed by the search term and `Enter`. Pressing `n` finds the next match. See `man less` (and search for 'search') for more details. See also *Efficient use of the Linux command line in the Bash shell* [6] for more information on `less` (Sect. 3.2.1), man pages (Sect. 7) and colours (Sect. 8).

9 Connecting to your Pi

We have seen how you can use ssh to log into your Pi. This section contains some useful tips to make that easier and exchange files between your local machine and your Pi.

9.1 Configuring your ssh client

Your personal configuration file for the ssh client is located in `~/.ssh/config`. You can create, open and edit it with your favourite text editor to simplify your login to your Pi. Let's go back to Jane logging in:

```
$ ssh jane@123.456.789.0 -p 32123
```

Jane only needs to specify her username if it is different from her user name on the local machine. But if this is the case, she has to specify it *every* time. In addition, typing the full IP address and port at every login is annoying. Instead, Jane can make the following entry in her `ssh config` on her **local machine** (*e.g.* her laptop):

```
Host rpi
     HostName 123.456.789.0
     User jane
     Port 32123
```

From now on, all Jane needs to type to log in is

```
$ ssh rpi
```

which is a lot more convenient.[19] Of course, if the Pi's IP address changes regularly, this option has less effect. See `man 5 ssh_config` for more details.

9.2 Copying stuff to your Pi

You may have a `.bashrc` (or other file) on your local machine that you would like to copy to your Pi. We can use **secure copy** or **scp** to do this using an ssh connection. To copy a local file to your Pi, type

[19] She could set the `alias rpi='ssh rpi' though`.

```
$ scp -P 32123 myFile jane@123.456.789.0:
```

Note the capital `-P`, and the colon to indicate that this is a *remote* location. I'll assume that you configured your ssh client as explained in the previous section, so that all you need to type is

```
$ scp myFile rpi:
```

Since we didn't specify a directory, the file will end up in the home directory of the user `jane`. If we want to send it to her directory `Documents` instead, we change the command to

```
$ scp myFile rpi:Documents/
```

Recursively copying directories is also possible:

```
$ scp -r Documents rpi:
```

Copying things *from* your Pi is analogous:

```
$ scp rpi:Documents/myFile .
```

I use the `alias scp='scp -rpC'`. See `man scp` to see what that means and for more details and `man rsync` to sync only the files that have *changed* rather than a whole directory tree (I use `alias rsync='rsync -urvtplhPz'` and use the `-n` option to check what will be synced before actually copying anything).

9.3 Using an ssh key

SSH keys can be used to access a remote machine over ssh (and hence scp, rsync) without a password. SSH keys can be protected by a passphrase, but here I will assume that your Raspberry Pi is currently at such a low security level that you want to access without a passphrase. Note however, that whoever has access to your local machine could have access to your Pi. The lemma *SSH keys* on the Arch Wiki [16] has many more details than discussed here, including how to use a passphrase only once per local session by using **`keychain`**.

If the files `~/.ssh/id_rsa*` already exist, you don't need to generate them. If they don't, you can generate a private/public key pair by typing

```
$ ssh-keygen
```

and pressing `Enter` three times to accept the default file name and to keep the passphrase empty. This creates the files `id_rsa` and `id_rsa.pub` in `~/.ssh/`. The first one is your *private* key, and should be kept secret. The second one is your *public* key, and you can send it to whomever you like. In particular, we will add it to the file `~/.ssh/authorized_keys` on the Raspberry Pi. If that last file doesn't yet exist, simply do

```
$ scp ~/.ssh/id_rsa.pub rpi:.ssh/authorized_keys
```

If all went well you will not be asked for a password when logging in to your Pi from the current machine through ssh (or using scp or rsync).

Appendices

Appendix A Basic use of Emacs

I treat GNU Emacs [17] here, not XEmacs, Aquamacs or any other flavour.

A.1 Emacs basics

You can open a text file by specifying its name as an argument:

```
$ emacs hello.c
```

If a graphical version of emacs and graphical desktop are available, it will open in a new window and you may want to append an ampersand (&) in order to start it in the background. You can force the terminal version of emacs with the option `-nw`.

A default Emacs screen consists, from top to bottom, of:

1. The menu (if shown);
2. The main text-edit panel (called *buffer*);
3. A status line;
4. A *minibuffer*, where commands show up (usually one line).

The menu can be accessed (and exited) with F10, and navigated with the arrow keys and Enter. For most options, the keyboard shortcut is shown, so that you can quickly learn it. Once you know the shortcut, you'll probably never use the menu again. Note that only a small fraction of Emacs's functionality is available through the menu!

The most important key strokes to know in order to start learning Emacs are:

- F10 Access or exit the **menu**;
- C-/ **Undo** text edit;
- C-g **Cancel** current command (press more than once if needed);
- Esc Esc Esc **Cancel everything** and go back to edit mode;
- C-x C-c **Exit** Emacs, prompting to save changed buffers.

Here, C-/ stands for Ctrl /, *i.e.* holding Ctrl whilst pressing /, then releasing both.[20] Similar combinations exist for Alt, which is written as M-.[21] Most commands have a (long) name as well as a shortcut, which can be typed using M-x, followed by the name of the command and Enter. Tab-completion works while typing the command. For example Undo can also be achieved by M-x `undo` Enter.

The Undo function in Emacs is different from those in many other programs. Consider for example the case where you type `a`, remove it, and then type `b`, then press Undo, then type `c`. If you then start pressing Undo, many programs will never recover `b` — that version of the document has been lost forever. Instead, when pressing C-/ repeatedly, Emacs will Undo *exactly* what happened before, including undoing previous Undo's. While perhaps unorthodox,

[20] For C-x C-c you can hold Ctrl while pressing X and then C.

[21] Pressing and releasing Esc has the same effect as pressing and holding Alt.

this means that you will always be able to get back to any previous version by pressing Undo (within the limits of your Undo history).

This information is an excerpt from *Getting started with Emacs* [18]. See that document for more information, to get started with Emacs and for more references.

Appendix B Pacnew and `etc-update`

When new versions of system packages are installed, they may come with new configuration files. Since you may have changed these files, your changes would be overwritten if these config files were simply updated too. Instead, pacman prints a line like

```
warning: /etc/pam.d/usermod installed as /etc/pam.d/usermod.pacnew
```

and creates a new file with the extension `.pacnew`. It is your task as a maintainer to see whether the old file, the new file or a combination of both must be used, and then to remove the `.pacnew` file.

You can find all `.pacnew` files using (assuming you sit in `/etc/`):

```
$ find . -name "*.pacnew"
```

You then know that for each of these files, there is the original file without that suffix that must be updated. To see the differences between the two files, use:

```
$ diff -wd configfile configfile.pacnew
```[22]

The Arch Linux Wiki [10] provides more details on how to keep track of `.pacnew` files and lists some tools that can aid you. If you don't know `vi`, `vim` or `vimdiff`, you can try `etc-update`, which can be installed from AUR (see Sect. 6.3). As a normal user:

```
$ yaourt etc-update
```

After installing it, you can make `etc-update` use `colordiff` by updating the respective lines in `/etc/etc-update.conf` to

```
pager="less -R"
diff_command="colordiff -wduN %file1 %file2"
using_editor=0
```

Before you deal with any `.pacnew` files, I suggest that you add them to your git repo in `/etc/`.

```
$ git add $(find . -name "*.pacnew" |sed -e 's/\.pacnew$//g')
$ git commit -a
```

Then run `etc-update`:

```
$ etc-update
```

You will see a list of configuration files that need to be updated (if there are any). You can always exit by typing `-1` Enter.

a) From the list, choose the number of a file followed by Enter to process it.

b) View diff — press SPACE to scroll, and q to quit. Then choose from:

   1) Replace the original with the new version. This will **overwrite** the original.

   2) Delete the update.

[22] You could install the package `colordiff` using `pacman` and use it instead of `diff`.

3) Merge the two versions. In that case, for each diff chunk:

   - choose `l` to select the left (original) chunk, or `r` for right
   - when done, select 1) to use the newly merged file (or any of the other options if you want to check the result)

c) Go back to a).

Appendix C Using a newer version of `bsdtar`

Unpacking the tarball for the base system (see Section 2.4) requires a recent version of **`bsdtar`**. You need **version 3.3** or higher. You can check whether `bsdtar` is installed, and if so, which version, with:

```
$ bsdtar --version
```

Operating systems like Ubuntu, and in particular their long-term-support (LTS) releases, are known to ship older versions of `bsdtar`. If this is the case for your system, and updating the system does not provide you with a newer version, you can use the steps below to install a recent version of `bsdtar` and use it to unpack your tarball.

```
$ wget https://www.libarchive.org/downloads/libarchive-3.4.3.tar.gz
$ tar xzf libarchive-3.3.2.tar.gz
$ cd libarchive-3.3.2
$ ./configure
$ make -j4
$ sudo ./bsdtar -xpf ../ArchLinuxARM-rpi-4-latest.tar.gz -C root/ && sync
```

This will download and unpack the package, enter its source directory, configure and build the code, but **not** install it on your system. The last step uses the newly compiled program (hence the dot-slash) rather than the system version of `bsdtar` to unpack your Arch Linux base system.

Appendix D References

[1] *Arch Linux*. URL `https://www.archlinux.org`. Visited 2017-08-03.

[2] **Wikipedia**. *List of Linux distributions*. URL `https://en.wikipedia.org/wiki/List_of_Linux_distributions`. Visited 2017-08-03.

[3] *Arch User Repository (AUR)*. URL `https://aur.archlinux.org`. Visited 2017-10-24.

[4] *Arch Linux ARM*. URL `https://archlinuxarm.org/`. Visited 2017-08-03.

[5] *Arch Linux 32*. URL `https://archlinux32.org/`. Visited 2018-01-23.

[6] **van der Sluys, M.** *Efficient use of the Linux command line in the Bash shell*. URL `http://eubs.sf.net`. Visited 2016-08-10.

[7] **vo6oh3@Arch Forum**. *bsdtar: Ignoring malformed pax extended attribute*. URL `https://archlinuxarm.org/forum/viewtopic.php?t=11396&start=16`. Visited 2018-12-18.

[8] **Arch Wiki**. *Arch Linux Users and groups*. URL `https://wiki.archlinux.org/index.php/Users_and_groups`. Visited 2017-10-25.

[9] **AstroFloyd**. *Git for coworkers*. URL `https://astrofloyd.wordpress.com/2012/12/16/git-for-coworkers/`. Visited 2017-08-29.

[10] **Arch Wiki**. *Arch Linux Pacman/Pacnew and pacsave*. URL https://wiki.archlinux.org/index.php/Pacman/Pacnew_and_Pacsave. Visited 2017-10-31.

[11] ——. *Arch Linux General recommendations*. URL https://wiki.archlinux.org/index.php/General_recommendations. Visited 2017-10-25.

[12] ——. *Arch Linux System maintenance*. URL https://wiki.archlinux.org/index.php/System_maintenance. Visited 2017-10-31.

[13] ——. *Arch Linux Pacman Tips and tricks*. URL https://wiki.archlinux.org/index.php/Pacman/Tips_and_tricks. Visited 2017-10-25.

[14] ——. *Arch Linux AUR helpers*. URL https://wiki.archlinux.org/index.php/AUR_helpers#Build_and_search. Visited 2017-10-31.

[15] **AstroFloyd**. *Installing Yaourt on Arch linux*. URL https://astrofloyd.wordpress.com/installing-yaourt-on-arch-linux/. Visited 2017-10-31.

[16] **Arch Wiki**. *Arch Linux SSH keys*. URL https://wiki.archlinux.org/index.php/SSH_keys. Visited 2017-12-05.

[17] **Stallman, R.** *Emacs*. URL https://www.gnu.org/software/emacs/. Visited 2016-03-20.

[18] **van der Sluys, M.** *Getting started with Emacs*. URL http://han.vandersluys.nl/?title=Publications. Visited 2017-11-13.