Operating systems (OPS)
Final assignment

Installing Arch Linux ARM on a Raspberry Pi

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

March 14, 2019

1 Introduction

The final assignment for the Operating Systems (OPS) course at the HAN consists of the installation, configuration and system maintenance of an Arch Linux ARM system on your Raspberry Pi v3. For this you use a **separate SD card!** All data on the SD card will be wiped, so make sure you do not use the SD card that is used for IoT or any other purpose. When you have finished this project, you will hand in the SD card for grading. Once the grades have been established, your card will be returned to you.

In this document, we frequently refer to the document *Installing Arch Linux ARM on a Raspberry Pi (IALARP)*, which lists the actual steps needed to install and configure the system. Before you continue, read that document carefully and make sure you understand what is happening at every step. If you are in doubt, ask someone (*e.g.* me) for help!

While you are carrying out the installation steps, you will answer questions about your system asked in this document. Make sure you write down the answers to those questions in a small plain-text report in the file `/root/answers.txt`, as they will be part of the grading. You can redirect output from a command to that file (and later edit it with emacs if necessary) using *e.g.* (for the command `df`)

```
df -h >> /root/answers.txt
```

Make sure you use **two** '>'s, otherwise you will overwrite the existing file! Note that you must be root to write to this file. You can copy text between files using emacs by loading two files in different buffers of the *same* emacs instance, and then copy or cut in one buffer and paste in the other.[1] Also make sure that you create a git repository in `/root/` and add this file to it.

Note that in most graphical Linux environments, selecting a text with your mouse also copies it, while pressing the middle mouse button or scroll wheel pastes the selection. You can use this to copy commands from the PDFs to your ssh terminal. However, **always check** what you pasted before executing it. Also make sure you don't copy a newline, since it may be pasted as `Enter` and execute the command before you checked it.

[1] Open a second file (in a separate buffer) using `C-x C-f`, typing the file name and pressing `Enter`. Switch between buffers with `C-x ←` (one direction) and `C-x →` (the other direction).

2 Preparing the SD card

In the *IALARP* document:

1. Read Section 1 (Introduction)
2. Read and carry out the instructions in Section 2 (Preparing the SD card) on your PC or laptop. When you are done, you should be able to boot Arch Linux ARM on your Pi. Note that you need `bsdtar`, which you may have to install first (using `apt-get` in Ubuntu). If your version of bsdtar gives an error, you may have to upgrade it to the latest version.

Note that Linux may not be allowed to access your internal card reader when using a Virtual Box. An external card reader may provide a solution, because the USB connection is usually passed to the Linux client properly [1].

3 Booting and logging in to your Pi

Read and carry out the instructions in Section 3 of *IALARP* (Booting and logging in). We will log in through ssh, so make sure you connect the Pi to an Ethernet cable.

Action 3.1: Choose the user password `passOPS`.

Action 3.2: After you have changed your password, log out and in again with your new password.

Question 3.1: Disc use

You can see how much of your disc is free (and used) in human-readable form using

```
$ df -h  # Disk free, human-readable output
```

The `root partition` of your file system (SD card) is mounted in `/`.

a) How much of your disc (SD card) has been used by the base system?

b) What does the output look like without the `-h` flag?

Question 3.2: Memory use

Free (and used) memory can be shown using

```
$ free -h
```

- How much of your memory does the base system use?

4 Configuring your Pi

Read the entire Section 4 of the *IALARP* document.

4.1 Changing to root

Become `root` as explained in Sect. 4.1 of *IALARP*.

Action 4.1: Choose the root password `OPSrootPass`.

Action 4.2: After you have changed your password, log out as root and in again with your new password.

4.2 Initialising the package manager

Carry out the instructions in Sect. 4.2 of *IALARP*.

4.3 Installing emacs

NOTE: if you installed from an (older) disc image, read Sect. 5.2 of *IALARP* first, and update your system before installing emacs.

Action 4.3: Install emacs.

Run the three `pacman` commands in Section 4.3 of *IALARP*, thus installing the console version of emacs. Look at the output carefully. Now search for the `emacs` packages again.

Question 4.1: How can you tell whether a package is installed?

Action 4.4: Set the `EDITOR` environment variable in your `.bashrc`.

Edit the `.bashrc` and add line to set the `EDITOR` variable. Log out and back in as root and check whether the variable is set correctly by typing

```
$ echo $EDITOR
```

4.4 System-configuration files

4.4.1 `/etc/`

Action 4.5: `cd` into `/etc/` at `su`.

In the document *IALARP*, carry out the instructions in Section 4.4.1. Log out and in as root to see whether you end up in `/etc/`.

4.4.2 Git and GitHub or Bitbucket

Read Section 4.4.2 of *IALARP*. We will here ignore the last recommendation, since we do not (yet) deal with a critical system, and push ('upload') our git commits to GitHub (or Bitbucket).[2]

[2] If you'd rather use Bitbucket than GitHub, please do. Read 'Bitbucket' whenever the text says 'GitHub', and read Appendix B as well as A.

Use your browser to create a free account on GitHub [2] or Bitbucket [3]. Use an email address and choose an alias/user name that are fit for me to see, and that allow me to recognise you. Do not yet create a repository. See Appendix A for instructions.

Action 4.6: Install the package `git`

Question 4.2: Which version did pacman install?

Action 4.7: Create a git repository in `/etc/`

Follow the instructions in Section 4.4.2 of *IALARP*. Use the **same** email address for the (global) configuration as you used for your GitHub account (*not* the `root@RPi` from the example in *IALARP*). Add the files to the repo and commit your changes as instructed.

Every file in `/etc/` or any of its subdirectories that is added to git will belong to *this* repository (unless you create a new git repo in one of those subdirectories).

Action 4.8: Create a git repository on GitHub

Use your browser and follow the instructions in Appendix A.[3] Call the repository **rpi-etc**. Add me (user MarcvdSluys)[4] as a collaborator on Github.[5] In your local repo in `/etc/`, add the **remote** for your GitHub repo and **push** your commit to the server. In your browser, reload the page and see whether your commits arrived in good order.

4.5 Timezone

Action 4.9: Set the system's timezone.

Carry out the instructions in Section 4.5 and set the timezone to "Amsterdam".

4.6 Installing and allowing `sudo`

Action 4.10: Install and allow `sudo`.

Follow the instructions in the first paragraph of Section 4.6 of *IALARP* to install `sudo`, but do not yet edit the configuration file. Add and commit the original sudoers file to your git repository:

```
$ git add sudoers
$ git commit -m 'Add original sudoers file to repo'
```

Now carry out the rest of Sect. 4.6 and commit your changes:

```
$ git diff  # Check whether everything is alright before committing
$ git commit -a -m 'Updated sudoers file'
```

Check the commits you have made so far:

```
$ git log
```

[3] Or Appendices A and B for Bitbucket.
[4] If you see a guy with long hair, a white beard and cool sunglasses, you're good.
[5] Or give me (user alias `han-ese`, name HAN ESE) read access to your repository on Bitbucket.

Question 4.3: When was the last commit made in your local repository? When was the last commit that was pushed to the server?

Push your commits to the server and check your browser to see whether they arrived:

```
$ git push
```

Question 4.4: How is your git log different from the previous one?

4.7 Configuring Pacman and AUR

Action 4.11: Configure Pacman and AUR.

Follow the instructions in Section 4.7 of *IALARP*, and diff, commit and push your changes.

5 System administration

In *IALARP*, read Section 5.

Action 5.1: Update your system.

Update your system to the latest version as described in Section 5.2 of *IALARP*.

Question 5.1: Which packages were updated?

Action 5.2: Install more packages

Install and enable `cronie` as described in *IALARP* Section 5.3 and install the packages described in Section 5.4. You can skip Section 5.5 (Wireless networking), unless you want to do this.

6 Installing packages from AUR

Exit from root and become a normal user again (or log in through a second ssh session — we will need the root session again soon).

Action 6.1: Install `yaourt` and `aurvote`.

Read Section 6 of *IALARP* and carry out its instructions to install `yaourt` and `aurvote`. Create an account on AUR [4], using some user name that keeps you anonymous and a simple password that you don't use for top-secret stuff. Vote for `aurvote`.

Go back and read Section 5.2.1 of *IALARP* on pacnew files.

Question 6.1: What happens when you try to install `etc-update` as root?

Action 6.2: Install `etc-update` as a normal user.

As root, add and commit `/etc/etc-update.conf` to git. Update the file as instructed and diff, commit and push your changes. Install `colordiff` if it isn't already installed.

Action 6.3: Run `etc-update` as root (from `/etc/`).

After each system or package update, you should run `etc_update` as described in Section 5.2.1 and Appendix B of *IALARP*, until all `.pacnew` files have been processed. This ensures that the versions of the config files match those of the package, so that you have a consistent system.

7 Security

Read Section 7 of *IALARP*.

7.1 User management

Action 7.1: Add user/group files to git

Add the files `passwd, shadow, group and gshadow` in `/etc/` to your git repo.

Action 7.2: Create the user `opsstudent`.

Create a home directory as well. Add as main group **`users`** and as additional group **`wheel`**. Use **`/bin/bash`** as shell. Give the new user the password **`passOPS`**.

Action 7.3: Log in as the new user

Use SSH. Type `su` in order to check that you can become root as this user. **Continue only when this works!**

Action 7.4: Block the old user `alarm`

Do not remove the account or home directory. Verify that you can indeed no longer log in.

Question 7.1: What message do you see when you try to log in over ssh as user `alarm`?

7.2 Configuring the SSH daemon

Set the SSH port to 32123, make sure that root cannot login and that *only* the user `opsstudent` can login through SSH. Restart `sshd` and use a **different terminal** to log in before you log out as root!

7.3 Preventing the accidental overwriting of files

Action 7.5: Test the current functionality of `rm`.

Create a temporary file called temp.txt and write a line of text using your text editor. Type

```
$ rm temp.txt
```

Question 7.2: Are you asked to confirm whether the file should be deleted?

Action 7.6: Add the three aliases to the system-wide `bashrc` file.

See Sect. 7.3 of *IALARP* for instructions. Log out and back in. Create another temporary file and remove it with the command above.

Question 7.3: Are you asked to confirm whether the file should be deleted this time?

8 Setting up your bash environment

Read Section 8 of *IALARP*.

8.1 Setting your bash prompt

Action 8.1: Set your bash prompts.

Set a coloured bash prompt as instructed. Use a different prompt for the user `opsstudent` and `root`. Log out and back in to see the effect, both for the normal user and root.

Question 8.1: Which colours are used for both users?

8.2 Adding colours to the man pages

Add the `_TERMCAP_` lines and the alias to the system-wide `bashrc` file. Log out and back in. Consult the man page for `man`.

Question 8.2: What colour do the flags (`-m`, etc.) in the SYNOPSIS line have?

Question 8.3: What colour do the options (`system`, etc.) in the same line have?

Action 8.2: Search for `search` in the man man page

Question 8.4: At which option does the first hit occur?

9 Printing boot, shutdown and login messages on the SenseHAT

We want to print a message at boot on the SenseHAT, stating the IP address(es) of the Pi. In addition, we want to print a goodbye message at shutdown, and another message when a user

logs in. We have created the Python scripts that do these things for you, in an Arch package called **han-ese-ops-arch-rpi** in the AUR. We will install that package (and its dependencies) and answer a few questions during the install process (read the questions before you start).

Action 9.1: Install the package han-ese-ops-arch-rpi using yaourt. Inspect the PKGBUILD during the process.

Question 9.1: From which website/URL will the package be downloaded?

Question 9.2: On which other packages does this package depend?

You can now finish the installation.

We need to load the i2c_dev kernel driver and make sure it is loaded on boot in order for the SenseHAT to work:

Action 9.2: Load the i2c_dev driver with modprobe i2c_dev (as root):

```
$ modprobe i2c_dev
```

Action 9.3: Load at boot: echo "i2c-dev" >> /etc/modules-load.d/raspberrypi.conf

Loading the driver should create, among others, the I^2C device file **/dev/i2c-1**. Check that it exists.

A normal user should have access to the display of the SenseHAT. In order to establish this, you must add your user to the video group:

Action 9.4: Add your user to the video group with usermod -a -G video <userName>

Action 9.5: Test the display of the SenseHAT as root with the command/script han-ese-ops_run-at-boot

Action 9.6: Test the display of the SenseHAT with the han-ese-ops_login script as a normal user.

One of the files that is installed is the system service **han-ese-ops** in the file /usr/lib/systemd/system/han-ese-ops.service.

Question 9.3: Which script does the service run when it is started? And when it is stopped?

Action 9.7: Enable the service han-ese-ops and reboot your Pi.

Question 9.4: Which message is printed on shutdown? And which on boot?

The file /usr/bin/han-ese-ops_login was also installed by the han-ese-ops-arch-rpi package.

Action 9.8: Add the line `/usr/bin/han-ese-ops_login &` to the end of your `/etc/bash.bashrc`, and log out of and back in to your Pi.

Question 9.5: What happens to the SenseHAT when you log in? What happens when you `su` to root?

10 Logging the SenseHAT weather to a file in a cron job

The SenseHAT can also measure temperature, relative humidity and air pressure. By default, a normal user is not allowed to access these values from the I^2C bus, but the `han-ese-ops` service sets the permissions to fix this at each boot.

Question 10.1: Which line in which file sets the permissions that allow anybody to access the I^2C device?

The script **`han-ese-ops_log-weather`** reads out the temperature (°C), humidity (%) and pressure (hPa), adds a timestamp and sends the output to `stdout`. Test whether the command works from the command line, as user and as root.

Question 10.2: What is the output of the command?

A GNU/Linux system logs its messages to `/var/log/`. We will need to be `root` in order to write to that directory.

Action 10.1: Execute the command `han-ese-ops_log-weather` >> `/var/log/weather.log` three times. Check the resulting weather log file.

Question 10.3: What happens if you replace >> with > in the previous command?

Of course, we do not want to log the weather by hand. Instead, the script should be executed every five minutes whenever the system runs. For this, we use **cron** [5]. Cron is a system service that runs a given command *e.g.* every hour, day or month, or on certain days of the week or certain minutes of the hour. The default cron service on an Arch system is called **`cronie`**. See Sect. 5.3 of *IALARP*.

Action 10.2: Install cronie.

Action 10.3: Enable the cronie system service.

A task that is run by cron is called a **cron job**. The file or table that lists cron job is called a **cron tab**. The command **`crontab -l`** lists the cron jobs for the current user, while the command **`crontab -e`** starts your favourite emacs to edit the cron tab. You can then edit the cron tab, save the file and quit emacs. Ensure the cron tab ends with an empty line. If there is an error in the format, `crontab` will offer to reedit the file to fix it. Be aware of **`crontab -r`**, which **removes** your cron tab without questions asked! More information on cron and cron tabs can be found in Appendix C. You should read that Appendix before you continue.

The command we want to run as a cron job is
`/usr/bin/han-ese-ops_log-weather >> /var/log/weather.log`.

Action 10.4: As root, create a cron tab with a cron job that logs the weather using the command above every five minutes. Check the results.

11 OPS programming exercises

You can use your Raspberry Pi to make OPS programming exercise 8. Ensure you do this as a **normal user**, not as root. In the home directory of your user, do

```
$ mkdir exercises
$ cd exercises
$ mkdir ex08
```

Create a git repository in the directory `~/exercises`. Create an online repository in GitHub and push the local version to the server. Add me (user `MarcvdSluys`) as a collaborator on GitHub.[6]

11.1 Exercise 8

For Exercise 8, you start with the result of Exercise 5. **Before you start**, copy these files into your directory `ex08`, add the `.c` and `.h` files to git and commit (and push) them. Then start working on Exercise 8. Whenever you finish a part of your code, compile and test it, and if it works, commit it. I want to see multiple commits when I assess your work. You can push your commits to the server after each commit, at the end of the day, or somewhere in between.

12 Creating tarballs and handing in your project

12.1 Creating a tarball

Source code is often shipped in a file ending in `.tar.gz`, called a **tarball**.[7] Creating a tarball happens in two stages: first, a tar file is created containing the desired files (and directories), which is then compressed.

The `tar` program was designed to backup files on tape. It groups files and directories, preserving their permissions, into a single file, which is uncompressed. To create (`c`) a tar file containing your whole `/etc/` directory, as root, cd to the root directory (`cd /`) and execute

```
tar c /etc/ > etc.tar
```

In order to compress a tar file and create a *tarball*, `gzip` is the default option:

```
gzip -9 etc.tar
```

This uses the strongest compression available (`-9`) and creates `etc.tar.gz`.[8] We have already seen how to unpack tarballs when we installed `yaourt` (you shouldn't try that here though).

12.2 Handing in your project

Before you hand in your project, check that you used the correct:

[6] Or give me (HAN ESE, user alias `han-ese`) read permission on Bitbucket.

[7] Not to be mistaken for a smokers' disease in men.

[8] In order to achieve stronger compression, use `bzip2` or `xz`. See their man pages for more details.

- ssh port;
- username and password for your user;
- root password.

Also, please download my public SSH key from the OPS Final Assignment web page and add it (on its own line) to the `~/.ssh/authorized_keys` file of your normal user.

Hand in your micro-SD card in its adaptor, and in a sealed envelope with your name and number, so that I can return it to you. If you drop it in my mailbox, please send me an email as well, so that I have a record of this.

Finally, ensure that your GitHub or Bitbucket repositories are uptodate.

Appendices

Appendix A Using GitHub with git

If you want free, private (*i.e.*, others can't see it unless you want them to) `git` repositories, you can use GitHub on a student account.[9] I use private repositories as backup and/or to share my code with collaborators and public repositories for open-source projects. At the end of a day, GitHub can show me what I achieved that day.

Before you create an account at GitHub, find an email address that you are willing to share with others. Your email address is how GitHub recognises you. When signing up, give your full name (or as much of it as you want to share) and make up a good password — you'll have to type it each time you *push* your commits to GitHub.[10]

Once your account has been set up, you are ready to create your first repository. In order to do so, in your home screen, click the **plus** in the top right of the page, choose **New repository**, choose a good (not too long, without spaces) **name**, add a description for your repo, choose a **private** repository, **do not** initialise the repository with a README, **do not** add a `.gitignore` or licence file, and click **Create repository**.

GitHub will now show you how to push an **existing repository** from the command line:

```
$ git remote add Server git@github.com:<user>/<repo>.git
$ git push -vu Server master
```

The command **`git remote add`** adds a remote location called `Server`. I usually use the name `Server` (or `GH` if I intend to push my commits to multiple remote locations) rather than `origin`.[11] The **`git push`** command pushes your master branch to `origin` (or `Server`). You have to specify the name of the branch only once — after this

```
$ git push
```

suffices (though I add the option `-v` to see what happens in more detail).

After pushing your commits to the GitHub server, reload their web page with instructions to see your code.

The opposite of pushing, *i.e. pulling* changes that others made from the remote location to your local repository, is then simply done with

```
$ git pull
```

If you want to share your private repo with other GitHub users, go to the page of the desired repository, click **Settings** (top bar, right) / **Collaborators** (left menu), type and select the **user name**, and click on **Add collaborator**. The other user will receive an email and can now **clone** (*i.e.* do an "initial pull") your repo and push their changes. See [7] for detailed help.

Appendix B Using Bitbucket with git

Using Bitbucket is similar to using GitHub. Hence, read that section first, create an account on the Bitbucket website, and come back here for the Bitbucket specifics.

[9] See [6] to apply for a student pack using your school email address. You can do this after creating an account.
[10] Unless you use an ssh key. We will encounter ssh keys when setting up an Arch Linux system.
[11] The name `origin` for the GitHub server doesn't make sense when the repository originates on my laptop.

In order to create a repository on Bitbucket, in your home screen, click the **plus** in the blue bar on the left, choose **Repository**, choose a good (not too long) **name**, answer **NO** to the question whether to include a README, ensure that your repository is **private** and uses **Git**, and click "Create repository".

If you already have a local repository, **do *NOT* get started "the easy way"**, but follow the two steps/three commands under **Get started with command line / I have an existing project**:

```
$ git remote add Server https://<user>@bitbucket.org/<user>/<repo>.git
$ git push -vu Server master
```

After pushing your commits to the Bitbucket server, reload their web page to see your code.

If you want to share your private repo with other Bitbucket users, click **Settings / User and group access**, type the **user name**, select **Read** or **Write** and click on **Add**. The other user will receive an email and can now **clone** (*i.e.* do an "initial pull") your repo, and, if given write permission, also push their changes.

Appendix C Using `cron`

A task that is run by cron is called a **cron job**. The file or table that lists cron job is called a **cron tab**. The command `crontab -l` lists the cron jobs for the current user, while the command `crontab -e` starts your favourite emacs to edit the cron tab. You can then edit the cron tab, save the file and quit emacs. Ensure the cron tab ends with an empty line. If there is an error in the format, `crontab` will offer to reedit the file to fix it. Be aware of `crontab -r`, which **removes** your cron tab without questions asked!

The cron tab is a table in a text file, with six columns separated by spaces — the exact alignment is unimportant. Each line defines one cron job. The first five columns (or words) in a line specify when the cron job should run: the minute, hour, day of month, month, and day of week. An asterisk (∗) indicates that the job should run at *every* minute/hour/etc. The table below shows a few examples.

Min	Hr	DoM	Mnt	DoW	Effect
∗	∗	∗	∗	∗	Run every minute (of every hour/day/month)
5	∗	∗	∗	∗	Run 5 minutes past the hour (of every hour/day/month)
∗/5	∗	∗	∗	∗	Run every 5 minutes (of every hour/day/month)
5	12	∗	∗	∗	Run at 12:05 (on every day/month)
5	10-12	∗	∗	∗	Run at 10:05, 11:05 and 12:05 (on every day/month)
5	12	1	∗	∗	Run at 12:05 on the first day of every month
0	0	1	1	∗	Start fireworks once a year
5	12	∗	∗	0	Run at 12:05 on every Sunday

More details on the format of a cron tab can be found in `man 5 crontab`. The sixth column contains the command to execute, as you would type it on the command line. It is a good idea to use **full paths** to executable and output files, since the environment variables in the cron tab may be different from those in Bash.

References

[1] **JinnKo**. *How to access an SD card from a virtual machine*. URL `https://superuser.com/a/458085/450123`. Visited 2018-09-26.

[2] **Preston-Werner, T., Wanstrath, C. & Hyett, P.** *GitHub*. URL `https://github.com`. Visited 2017-08-29.

[3] **Atlassian**. *Bitbucket*. URL `https://bitbucket.org`. Visited 2017-08-29.

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

[5] **Wikipedia**. *Cron*. URL `https://en.wikipedia.org/wiki/Cron`. Visited 2018-03-12.

[6] **GitHub**. *Applying for a student developer pack*. URL `https://help.github.com/articles/applying-for-a-student-developer-pack/`. Visited 2018-12-18.

[7] ——. *Inviting collaborators to a personal repository*. URL `https://help.github.com/articles/inviting-collaborators-to-a-personal-repository/`. Visited 2018-12-18.