Setup for booting from SSD (USB)

Greg Robson-Garth
Melbourne
Australia
greg@robson-garth.net

Version 1.0

Introduction

These are the steps I followed to make the Raspberry Pi 4 bootable from SSD. I have 4 systems running like this. My particular configuration is (on all 4 systems):

Hardware:

Raspberry Pi 4's with a mix of 8Gb and 4Gb memory

- X825-C6 case with momentary switch
- X825 2.5" SATA HDD/SSD Storage Board
- X735 Power Management Board
- Samsung EVO 860 SSD: 1Tb or 500Gb or 2x250Gb

Use cases:

The 4 systems run standard linux software:

- Web server : using Apache and php (250Gb SSD).
- Reverse Proxy: using Nginx and letsencrypt certificates (250Gb SSD).
- Database server: using Mysql (1Tb SSD).
- Backup server: linux scripts and cloud uploads to One-Drive using rclone (500Gb SSD).

Operating system:

I use Ubuntu "groovy gorilla" 20.10 64-bit ARM version of Linux on all 4 systems. I could not get 20.04 to work for me.

uname -a:

```
Linux pi011 5.8.0-1011-raspi #14-Ubuntu SMP PREEMPT Tue Dec 15 08:53:29 UTC 2020
aarch64 aarch64 aarch64 GNU/Linux
```

cat /etc/os-release:

```
NAME="Ubuntu"
VERSION="20.10 (Groovy Gorilla)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.10"
VERSION_ID="20.10"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=groovy
UBUNTU_CODENAME=groovy
```

All 4 servers boot from the SSD and have no SD card installed.

Disclaimer:

Be careful when following these steps. These are the steps I used to install the 4 raspberry Pis to boot from SSD, and they are all operational. However, you should check with original documentation as appropriate to be sure if they are right for you.

The instructions have been written at a certain level. If you have better ways of doing things, please do so. If you are not sure, you can email me or simply ignore this document. I am hoping it will take away some of the questions you have to get this up and running.

References:

https://peyanski.com/official-raspberry-pi-4-usb-boot/

https://www.raspberrypi.org/forums/viewtopic.php?t=268476

In the second reference it notes that vmlinuz is not encrypted so it's not necessary to zcat the file. See extract in Appendix 1. Full credit to these references for guiding me.

Steps to install

3 steps are required

1. Update the bootloader
2. Install Ubuntu Linux 20.10 operating system on a SD card
3. Clone the SD card to the SSD and set up for booting from SSD

Step 1: update the bootloader

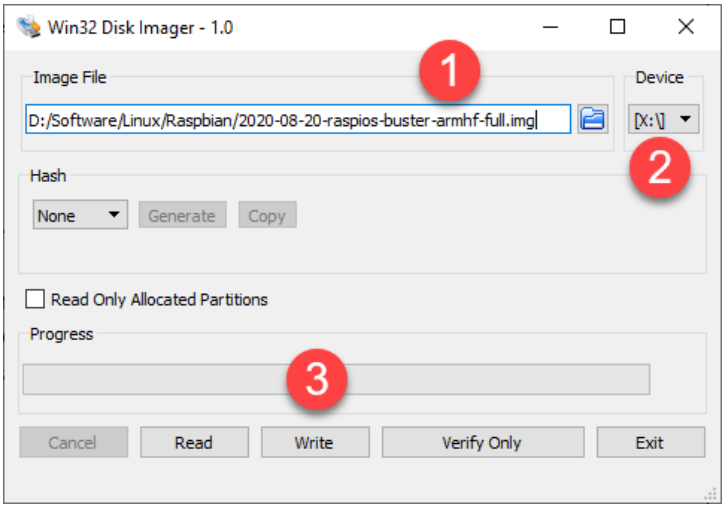
In the first step you update the *bootloader* firmware. This is software installed in the hardware (firmware) and not installed on any SD card. However, **it must be done using Raspbian**. It must not be done with ubuntu. Once it's been done, the Pi4 motherboard is updated and the changes are carried forward no matter which OS is subsequently used. If you subsequently update the bootloader, then any change you made to the bootloader config needs to be redone.

Update the bootloader	This first step makes a firmware change on the pi itself not to the file system. This bootloader will stay changed no matter what OS is used. ***Only use Raspian for this***. Don't try using ubuntu to update the bootloader. This first step can be done before or after you set up the Geekworm hardware. If it's after, then do not insert the USB connector (the supplied connector between the Pi and the X825). **1. Download Raspian Buster:** Download the Pi OS from https://www.raspberrypi.org/software/operating-systems/ For my systems I used:

2020-08-20-raspios-buster-armhf-full.img

2. Write the image to the SD card

Use Win32DiskImager (https://sourceforge.net/projects/win32diskimager/) or equivalent to write the image to a SD card. You will only need a SD card of around 16Gb at the most.



1. Select the image downloaded above
2. Select the drive letter of the SD card to be imaged
3. Click *Write*

3. Connect to the Pi

Once done, with the SD card is still in the USB holder still plugged into your PC/Mac, create a ssh file in the boot partition of the SD card. When you create an empty file named "*ssh*" inside the boot partition of your SD card – you will be able to connect to your Raspberry over the network without even attaching a monitor to it, using the SSH server.

On MacOS you can execute the following command in the terminal:

```
touch /Volumes/boot/ssh
```

On Windows run the command:

```
notepad <Drive>:\Volume\boot\ssh
```

(does not matter what is in the file)

Insert the Raspbian SD card into the Pi and disconnect any USB drives.
Switch on the power.

<u>Connect to the pi:</u>

Use ssh or connect a screen, keyboard and mouse.

If using ssh, the default user is *pi* and the password is *raspberry*. Otherwise follow the prompts.

<u>Get Updates:</u>

Execute these commands:

```
sudo apt-get update
sudo apt-get upgrade
sudo rpi-update
sudo reboot
```

Install the eeprom updater:

Login again and install a package that is used to update the bootloader to enable the Raspberry Pi 4 usb boot.

Run the following command. This will upgrade to the latest version if required:

```
sudo apt install rpi-eeprom -y
```

```
pi@raspberrypi:~ $ sudo apt install rpi-eeprom
Reading package lists... Done
Building dependency tree
Reading state information... Done
rpi-eeprom is already the newest version (7.2-1).
The following package was automatically installed and is no longer required:
  rpi-eeprom-images
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $
```

It may already be installed as shown above.

Then run this command to tell the Raspberry Pi to use beta firmware releases. This "one liner" will replace the word *critical* with *beta* in the /etc/default/rpi-eeprom-update file.

```
sudo sed -i 's/critical/beta/g' /etc/default/rpi-eeprom-update
```

```
pi@raspberrypi:~ $ cat /etc/default/rpi-eeprom-update
FIRMWARE_RELEASE_STATUS="critical"
pi@raspberrypi:~ $ sudo sed -i 's/critical/beta/g' /etc/default/rpi-eeprom-updat
e
pi@raspberrypi:~ $ cat /etc/default/rpi-eeprom-update
FIRMWARE_RELEASE_STATUS="beta"
pi@raspberrypi:~ $
```

…or use any text editor to make the change.

Updating the bootloader:

Now it is time to install the new bootloader to enable the Raspberry Pi 4 usb boot. First, get the list of bootloaders available.

```
cd /lib/firmware/raspberrypi/bootloader/beta
ls -al
```

```
pi@raspberrypi:/lib/firmware/raspberrypi/bootloader/beta $ ls -al
total 3284
drwxr-xr-x 2 root root   4096 Dec  4 14:02 .
drwxr-xr-x 5 root root   4096 Dec  4 14:02 ..
-rw-r--r-- 1 root root 524288 Jul 17 03:55 pieeprom-2020-07-16.bin
-rw-r--r-- 1 root root 524288 Aug  3 20:04 pieeprom-2020-07-31.bin
-rw-r--r-- 1 root root 524288 Sep  5 01:00 pieeprom-2020-09-03.bin
-rw-r--r-- 1 root root 524288 Nov  3 01:15 pieeprom-2020-10-02.bin
-rw-r--r-- 1 root root 524288 Nov  3 01:15 pieeprom-2020-10-28.bin
-rw-r--r-- 1 root root 524288 Nov 26 04:45 pieeprom-2020-11-24.bin
-rw-r--r-- 1 root root 106444 Nov 26 04:45 recovery.bin
-rw-r--r-- 1 root root  99224 Jul 17 00:14 vl805-000138a1.bin
pi@raspberrypi:/lib/firmware/raspberrypi/bootloader/beta $
```

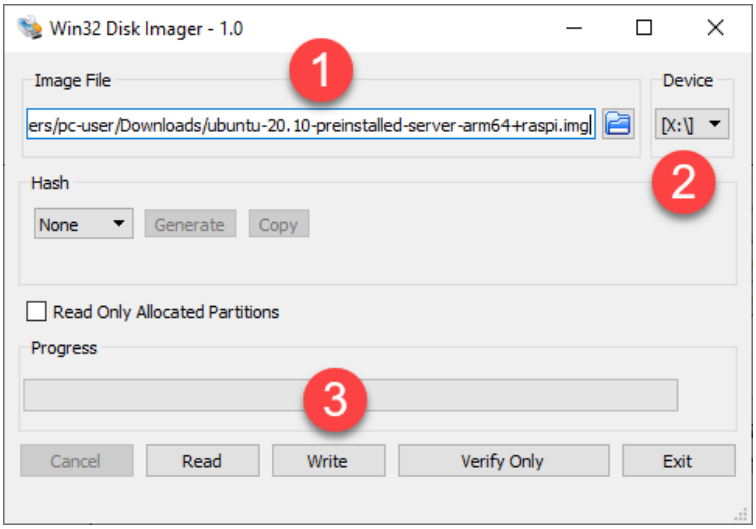
The above list shows all the available eeprom bootloader updates. Use the latest. I used a later version dated 2020-12-11 (not shown). Make sure the image is dated 3 Sept 2020 or later.

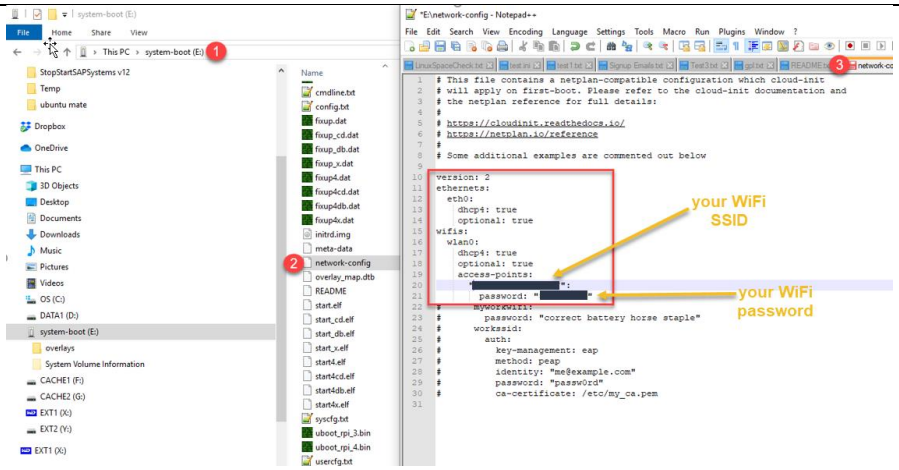
Now update the bootloader:

	`sudo rpi-eeprom-update -d -f /lib/firmware/raspberrypi/bootloader/beta/pieeprom-2020-12-11.bin`
	
	You should get a message similar to the one below with the name of the bootloader file you selected to install:
	 
	`pi@raspberrypi:~ $ sudo rpi-eeprom-update -d -f /lib/firmware/raspberrypi/bootloader/beta/pieeprom-2020-06-03.bin`
	`BCM2711 detected`
	`Dedicated VL805 EEPROM detected`
	`BOOTFS /boot`
	`*** INSTALLING /lib/firmware/raspberrypi/bootloader/beta/pieeprom-2020-06-03.bin ***`
	`BOOTFS /boot`
	`EEPROM update pending. Please reboot to apply the update.`
	
	Reboot the Pi:
	`sudo reboot`

Edit the bootloader config	Edit the bootloader config:
	The bootloader config contains the parameters which control the boot process. You need to add another parameter to this:
	
	Issue the command:
	`sudo -E rpi-eeprom-config --edit`
	
	Then add this line and save:
	`BOOT_ORDER=0xf41`
	
	Reboot the Pi:
	`sudo reboot`
	
	Check the result:
	Login again and execute these two commands:
	`greg@pi011:~$ sudo vcgencmd bootloader_version`
	`Dec 11 2020 11:15:17`
	`version c3f26b6070054bca030366de2550d79ddae1207a (release)`
	`timestamp 1607685317`
	`greg@pi011:~$ sudo vcgencmd bootloader_config`
	`[all]`
	`BOOT_UART=0`
	`WAKE_ON_GPIO=1`
	`POWER_OFF_ON_HALT=0`
	`BOOT_ORDER=0xf41`
	
	The first command will show you the bootloader version, the one that you updated in the *rpi-eeprom-update* command and the second will show the result of the edit of the config file. Make sure the date is correct and BOOT_ORDER=0xf41 has been added to the config file.
	
	These commands affect the bootloader in firmware not the SD card and are overwritten each time the bootloader is updated.
	
	The default boot order of 0xf41 means continuously try SD then USB mass storage. Refer to the following link for more information about the bootloader config. (*https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711_bootloader_config.md*)

Step 2: install ubuntu 20.10

Download and image the SD card	Download the image from ubuntu. At the time of writing it was available from : https://cdimage.ubuntu.com/releases/20.10/release/  Use Win32DiskImager once more to image a SD card. A 16Gb SD card will do. The file named *ubuntu-20.10-preinstalled-server-arm64+raspi.img.xz* must be unpacked using a program such as 7-Zip. Extract the file named *ubuntu-20.10-preinstalled-server-arm64+raspi.img.*  1. Select the extracted .img file 2. Select the USB device containing the SD card 3. Click *Write*
Edit the network config	Use notepad or notepad++ to edit the file network.config and fill in your Wifi SSID and password as shown. Or use an ethernet cable to connect to the Pi.

	 1. Select the USB drive containing your SD card 2. Edit network.config 3. Update your SSID and password. 4. Save
Connect to the Pi	Insert the SD card into the Pi and turn on. If you intend using SSH or putty to connect to the Pi, then the wifi, in my experience does not work on first boot. Wait 5 mins and switch the Pi off and on again. Then check the router for the IP address, and connect using ssh/putty. Otherwise you can connect a keyboard, mouse and screen. Enter the logon credentials: User: ubuntu Password: ubuntu Change the password. The pi session will log you off. Connect again and log on using your new password.

Update the Pi	Run these 2 commands to update the Pi.
	
	`sudo apt update`
	
	`sudo apt upgrade`
	
	If you get
	`Waiting for cache lock: Could not get lock /var/lib/dpkg/lock-frontend. It is held by process nnnn (unattended-upgr)`
	
	Then wait for the process (in this case nnnn) to complete. It can take up to an hour to complete. Alternatively kill the process and issue the commands above.

Optional steps	These steps are all optional and run them as you need:
	
	Install net tools. This installs ifconfig amongst others:
	`sudo apt install net-tools`
	
	Install the ubuntu desktop:
	`sudo apt install ubuntu-desktop`

Install the remote desktop so you can connect using windows RDP:

```
sudo apt-get install xrdp
```

Install the Synaptic Package Manager (GUI software installer)

```
sudo apt-get install synaptic
```

Install disk partitioning software:

```
sudo apt install gparted
```

Add a new user (for example “greg”)

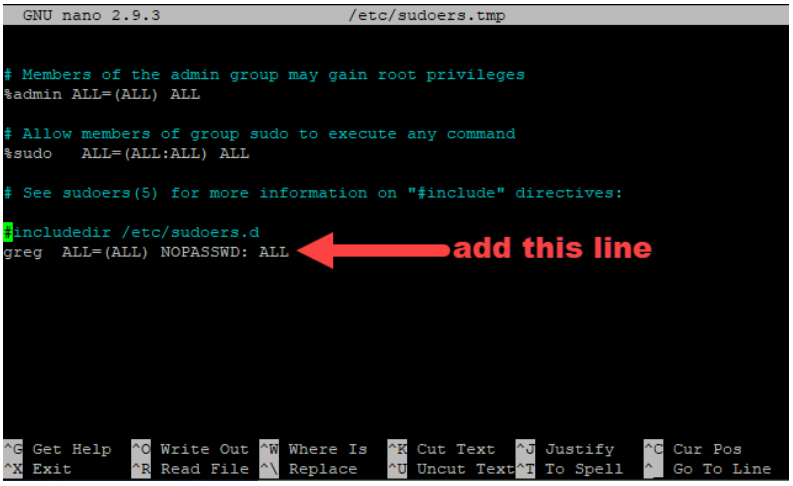
```
sudo adduser greg
```

```
sudo gpasswd -a greg sudo
```

```
sudo visudo
```

Add this line:

```
greg ALL=(ALL) NOPASSWD: ALL
```



Change the host name to a name of your choice (eg pi008)

```
sudo hostnamectl set-hostname pi008
```

Change the date and time (in my case to Melbourne/Australia:

```
timedatectl set-timezone Australia/Melbourne
```

Check again

```
timedatectl
               Local time: Sat 2020-07-18 10:20:43 AEST
           Universal time: Sat 2020-07-18 00:20:43 UTC
                 RTC time: n/a
                Time zone: Australia/Melbourne (AEST, +1000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

Use this command to get a list of all the timezones:

```
root@pi010:/etc/netplan# timedatectl list-timezones
Africa/Abidjan
Africa/Accra
Africa/Algiers
Africa/Bissau
Africa/Cairo
```

etc

Required Software	Install VC tools:
	`sudo apt install libraspberrypi-bin`
	
	Then check the bootloader config:
	Execute the following 2 commands to check the current bootloader version and config of the pi
	`vcgencmd bootloader_version`
	`vcgencmd bootloader_config`
	
	This must be the result
	`root@raspberrypi:~# vcgencmd bootloader_version`
	`Dec 11 2020 11:15:17`
	`version c3f26b6070054bca030366de2550d79ddae1207a (release)`
	`timestamp 1607685317`
	`update-time 1609978768`
	`capabilities 0x0000001f`
	`root@raspberrypi:~# vcgencmd bootloader_config`
	`[all]`
	`BOOT_UART=0`
	`WAKE_ON_GPIO=1`
	`POWER_OFF_ON_HALT=0`
	`BOOT_ORDER=0xf41`

Reboot	`sudo reboot`

Step 3: Prepare the SSD

Mount the SSD	By this time you must have set up the Pi together with the x735 and the X825, with the momentary switch connected, the X825 powered from the power board and the power supply plugged into the X725. The Pi must be booted without the USB connector between the Pi and the X825 plugged in. Once the Pi is up and running in the above configuration, now connect the USB connector between the X825 and the Pi: This is the USB connector: 

Clone the SD card to the SSD.	Find the SD and disk path using fdisk
	
	`fdisk -l`
	
	SD Card:
	
	`Disk /dev/mmcblk0: 119.08 GiB, 127865454592 bytes, 249737216 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: 0xc017cbaa`
	
	SSD:

```
Disk /dev/sda: 465.76 GiB, 500107862016 bytes, 976773168 sectors
Disk model: 2115
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 33553920 bytes
Disklabel type: dos
Disk identifier: 0xc017cbaa
```

Clone the sd card to the SSD (modify this command if your /dev devices are different):

```
dd if=/dev/mmcblk0 of=/dev/sda
```

This takes upwards of an hour depending on the size of the SD card.

Update the boot config.

Once the clone is complete:

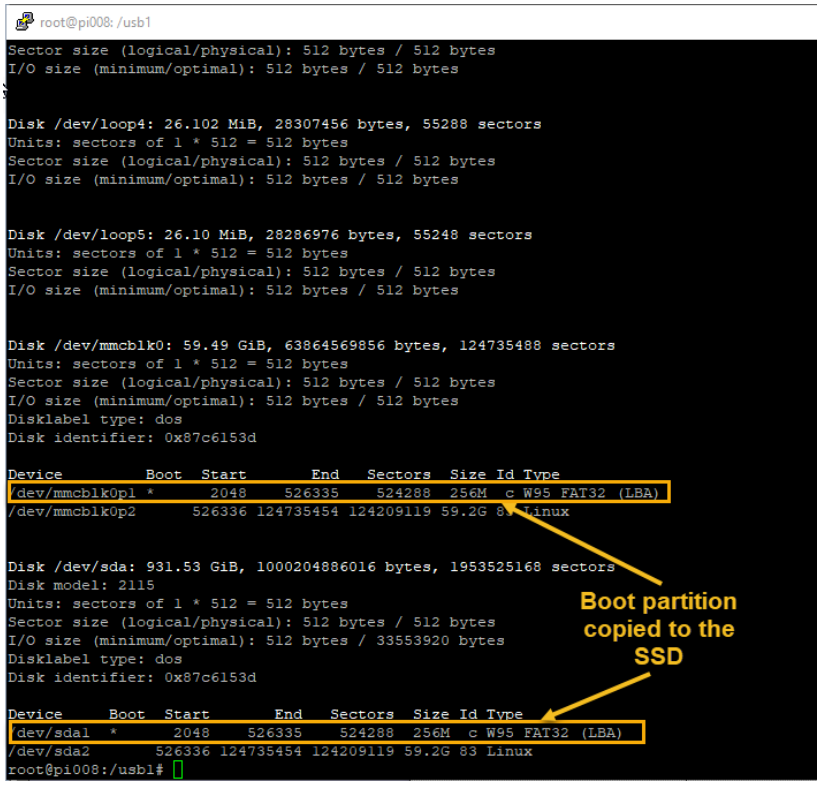
- Remove the USB – SSD connector !
- Reboot !

You must to the 2 steps above. Make sure you remove the connector before rebooting.

After the reboot, replace the USB SSD connector.
Now it's time to modify the boot config on the newly cloned SSD.

Mount the boot partition of the SSD.

```
fdisk -l
```



Mount the USB– **to the boot partition**:
(Check the partition number)

Using the above screen shot as an example:

```
mount /dev/sda1 /usb1
```

	The following file is found in the root directory of the boot partition **vi config.txt** Comment out the [pi… sections Add the 2 lines **kernel=vmlinuz** **initramfs initrd.img followkernel**  <u>Reboot:</u> Shutdown the pi: **sudo shutdown -h now** Remove the SD card, but leave the USB Pi to X825 connector in place. Switch the pi off Switch the pi on
Extend the disk	If you have installed *gparted*, *ubuntu-desktop* and *xrdp* you can remote desktop to the Pi and use gparted to extend the linux partition to make use of the whole disk or to partition the remaining space as you wish. You can also use the command line tool *parted* to do the same.

Appendix: 1

https://www.raspberrypi.org/forums/viewtopic.php?t=268476

Re: Ubuntu from SSD – easy method

Sun Mar 29, 2020 4:14 pm

We're working slowly towards support for USB boot, but the situation is complicated by our use of u-boot. In Eoan I changed the kernel command line and fstab (as **procount** and **smoore** have noticed) to use FS labels to identity the boot and root partitions, so those no longer need changing. I also updated the u-boot script so that *in theory* it can now boot from USB as well as SD/MMC.

Unfortunately there's still one missing bit: u-boot itself, despite having USB support compiled in (and despite apparently having USB mass-storage compiled in) ... doesn't see any mass storage devices whenever I've tested it! I've not yet had time to dig into this (beyond examining the compile time flags), but if anyone fancies a delve I'd love to hear any ideas!

In the meantime, if you want to try USB boot your best bet is to do so on a Pi3+ (where USB mass storage boot is supported "out of the box"), and to skip u-boot on startup (it isn't strictly required on classic - however it is on core for the A/B booting capability and the classic & core boot sequences derive from the same source). To do this:

- burn the image to your chosen USB device
- un-plug and re-plug the device in to mount it cleanly
- open config.txt on the boot partition and comment out/delete all the [pi*] sections
- add kernel=vmlinuz in the [all] section
- add initramfs initrd.img follow kernel in the [all] section

You should wind up with something looking like this:

Code: Select all

```
# Please DO NOT modify this file; if you need to modify the boot config,
the
# "usercfg.txt" file is the place to include user changes. Please refer to
# the README file for a description of the various configuration files on
# the boot partition.

# The unusual ordering below is deliberate; older firmwares (in particular
the
# version initially shipped with bionic) don't understand the conditional
# [sections] below and simply ignore them. The Pi4 doesn't boot at all with
# firmwares this old so it's safe to place at the top. Of the Pi2 and Pi3,
the
# Pi3 uboot happens to work happily on the Pi2, so it needs to go at the
bottom
# to support old firmwares.

#[pi4]
#kernel=uboot_rpi_4_32b.bin
#max_framebuffers=2
#
#[pi2]
#kernel=uboot_rpi_2.bin
#
#[pi3]
#kernel=uboot_rpi_3_32b.bin
```

```
[all]
device_tree_address=0x03000000
kernel=vmlinuz
initramfs initrd.img followkernel

# The following settings are "defaults" expected to be overridden by the
# included configuration. The only reason they are included is, again, to
# support old firmwares which don't understand the "include" command.

enable_uart=1
cmdline=nobtcmd.txt

include syscfg.txt
include usercfg.txt
```

At this point you can simply plug your USB device into your Pi3+, remove any SD card, and it should boot happily (the rainbow screen takes a bit longer to go away as it takes time for the firmware to notice there's no SD, to fire up USB, enumerate devices, etc. but it does get there - I've tried this with an SSD and a flash drive).

A warning: right now this will work on both armhf and arm64. However, in future this *will* break on arm64. The reason is to do with kernel compression: the armhf image is currently a self-extracting gzipped image; that's fine as both u-boot and the pi's firmware bootloader can handle that. However, the arm64 image is currently uncompressed. The reason is simply that self-extracting kernels aren't supported on arm64 (upstream's view is roughly "bootloaders should handle that, and we never should have added self-extractors on all those other archs in the first place").

While it's uncompressed, things will work okay (both u-boot and the pi's firmware bootloader can read that), but we're planning to ship it compressed (not self-extracting, just compressed) in future because the uncompressed kernel is pretty large and that's causing issues on some of the images with minimal boot partitions. Once it's shipped compressed, u-boot will be required to uncompress it before launching it (the pi's firmware bootloader doesn't currently know how to deal with a compressed kernel), and obviously that's going to break the instructions above which skip u-boot. You could work around this by un-compressing the kernel image after an upgrade, but it's an annoying hoop to jump through (I'd rather just fix u-boot to read USB devices properly).

For those wanting to try this on a Pi4:
(Greg: the recent bootloader as of 03-09-2020) allows booting from mass storage)
I would caution against trying this on a Pi4 at the moment. Booting from USB mass storage isn't (currently) supported on that model (although it is planned), so you wind up having to use an SD card to load the kernel and initrd, then continue the boot on the USB device.

It is possible to get this working (though because things are now relying on FS label you have to be careful not to duplicate those!), but you will wind up with issues every time the kernel gets upgraded or the initrd rebuilt because flash-kernel gets confused about where the boot partition "really" is (FWIW I wound up with similar problems when I tried this in Raspian even without flash-kernel - basically having two boot partitions, or a boot partition on a separate device to the root partition winds up painful whichever way you go). Yes, it's possible to work around these issues if you remember to jump through some more hoops, but while it was an interesting experiment I'd rather just wait for official USB boot support on the 4 which will make everything much simpler and should mean you can follow a similar path to the 3+ above.

Dave.
Author of / contributor to a few pi related things (picamera, Sense HAT emulator, gpio-zero, piwheels, etc.), and currently a software engineer at Canonical responsible for Ubuntu Server and Core on the Raspberry Pi.