Part 18
-
Backup & Restore Your Pi

Version: 2017-08-10

Backup & Restore your PI SD card to a NAS share using dd

Backup:
This turned out to be surprisingly easy providing your Pi is connected to a NAS drive.
Use the Raspbian `dd` command to make an image of the SD card, which you then schedule to run using `cron`.

Take a backup of the SD card using `dd` referencing the physical disk directory `/dev/mmcblk0` and a location on the NAS drive. It doesn't have to be a NAS drive but, obviously, it does need to be location not on the SD card, so an external USB drive should work just as well.

```
sudo dd if=/dev/mmcblk0 of=/path/to/NAS/drive/raspberryPiSDCardBackup.img
```

Depending on the size of your SD card and speed of the external drive you are connecting too, this will take a little time, but when complete it should have created a .img file which is the same size as the total size of your sd card e.g. if you have an 8GB sd card you will end up with an 8GB image regardless of how much capacity is actually used.

I also stop services which run in the background (for me these are lighttpd and noip) before starting the backup and restarted them when finished to minimise the risk of taking a backup mid way through an update to the SD card. As I also run other scheduled jobs using cron so I also suspend the cron service, so that other schedules jobs don't run while the backup is.

I created a script to do this called `RunSDBackup.sh`

```
nano RunSDBackup.sh
```

```
sudo /etc/init.d/lighttpd stop
sudo /etc/init.d/noip stop
sudo /etc/init.d/cron stop
sudo dd if=/dev/mmcblk0 of=/home/pi/NAS/Share/SDCardBackup.img
sudo /etc/init.d/cron start
sudo /etc/init.d/lighttpd start
sudo /etc/init.d/noip start
```

Make the script executable:

```
sudo chmod +x RunSDBackup.sh
```

I then schedule this job using `cron` and that was it, it runs once a week on a monday morning at 2:30am. Run `crontab` using the `-e` option for edit:

```
sudo crontab -e
```

Adding the following line to the bottom of the file, then using Ctrl-X to save:

```
30 2 * * 1 /home/pi/RunSDBackup.sh > /home/pi/RunSDBackup.log 2>&1
```

The first part "`30 2 * * 1`" is in the format "`minute hour dayOfMonth Month dayOfWeek`" tells `cron` to run when the minutes on the internal clock equal 30, the hour equals 2, for any day in the month, for any month when the day of the week is Monday.
The second part is the command which outputs the results to a log file..

Restore:
You'll end up with a .img file on your NAS share. So you can follow the usual instructions for writing to an SD card. Then, put the SD card into the Pi and start it up.

Backup & Restore your PI to a NAS share using rsync

Here's an intro to using rsync for back-up on the Pi. Once the initial back-up is created, keeping it up to date this way is much much faster than constantly ripping the entire image. You can do this to a local hard drive or over a network.

You actually do not want a complete copy of a running system as a back-up, since some of the stuff ostensibly in the filesystem exists only at runtime. Including that in a backup and then using it to recreate an image later may create problems for you.

There are some other exceptions too. Rsync can accept a list of (glob) patterns to exclude, and those can be read from a file, so let's first go thru what should be in such a file. First note that the entries are of the form `/directory/*` and not `/directory`. This is because we want them to exist, but we don't want to copy anything in them.

```
/proc/*
/sys/*
```

These do not really exist on disk. They're an interface to the kernel, which creates and maintains them in memory. If you copy these and then copy them back into a system and boot it, it will be (at best) meaningless, since the kernel uses those as mount points for the interfaces [If you want to see what happens when you mount a device on a directory with data in it, try. It works and won't do any harm, but the stuff that was in the directory is now inaccessible.]
Note that it is critical that the /sys and /proc mount points exist. But they should not contain anything. Next:

```
/dev/*
```

The dev directory is not quite the same thing as proc and sys but for our purposes it is. If you believe that you should save this so you can have the same device nodes in your backup or something, you're wrong. Don't bother. Do not copy dev. Once upon a long time ago linux did work that way, but it doesn't anymore.

```
/boot/*
```

This is sort of a special case with the pi, or at least, the raspbian image I'm using (and I believe others such as xbmc use a similar strategy). It's actually a mount point for the first, vfat, partition. We are going to deal with that separately. Whatever you do, don't bother including it here, because again, it's a mount point.

```
/tmp/*
/run/*
```

With the (debian based) pi `/run` is not on disk either, it's in memory. Perhaps /tmp could be too (this would save a bit of SD card action), but in any case, as the names imply, these are not places for storing persistent data. Copying and restoring these can cause problems because various userland tools store state information there and if you reboot with that, they may end up confused.

```
/mnt/*
```

This one is pretty important if you are planning to back up to a hard drive and the hard drive is in /mnt, hopefully for obvious reasons. Rsync might be smart enough to spot something that dumb (I have not tested it) but if not, you might end up filling the drive with the same thing over and over and over. Or something. So exclude mnt.

```
/media/*
```

The GUI File Manager mounts devices in `/media`, so this should be excluded.

So create a file `/rsync-exclude.txt`

```
sudo nano /rsync-exclude.txt
```

and add all this to it

```
/proc/*
/sys/*
/dev/*
/boot/*
/tmp/*
/run/*
/mnt/*
```

I have enhanced the rsync-exclude.txt to eliminate unnecessary files.

```
.bash_history
/etc/fake-hwclock.data
/var/lib/rpimonitor/stat/
```

If you might find other files or directory that aren't usefull to you, just add them to the `/rsync-exclude.txt file`

Now create a directory to back up to, eg, "pi_backup". If the above file is `/rsync-exclude.txt` and your drive is `/mnt/usbhd`, to do the actual backup:

```
mkdir /mnt/usbhd/pi_backup
rsync -aHv --delete-during --exclude-from=/rsync-exclude.txt / /mnt/usbhd/pi_backup/
```

Notice that there is a trailing slash on `pi_backup`

This will take a while and produce a lot of output (if you want to examine that in a log instead, append `> rsync.log`). `--delete-during` is meaningless the first time, but for keeping the backup updated use it. This ensures that stuff you've later deleted on the pi also gets removed from your backup.
`-a` sets recursion into directories and makes sure all the file attributes match.
`-H` is to preserve hard links.
`-v` is for verbose which is why you get some output, otherwise rsync is quiet.
See man rsync for more.

There is a shortcut whereby you can skip the `--exclude-from` file. If you are sure that all of the things you don't want to copy (`/tmp` etc.) are on separate filesystems, you can just use:

```
rsync -axHv --delete-during / /mnt/usbhd/pi_backup/
```

`-x` has been inserted. This is the short form of `--one-file-system`, which tells rsync not to cross filesystem boundaries. Personally I prefer the `--exclude-from`, but on e.g., default Raspbian, `--one-file-system` will work fine.

That's not quite a complete backup. It's enough if you haven't put anything in boot and you are fine with using the back up to just restore the system by sticking the card in a computer and running:

```
rsync -av --delete-during /mnt/usbhd/pi_backup/ /mnt/sdcard_partition2/
```

You could also do this with a card with a new image on it (presuming it's the same as your base image) although that's a little inefficient if you have to create the image (because you're

then going to overwrite most of it). You could also connect another SD card via a usb adapter with such an image on it, and use the above method to maintain a duplicate card.

If you've put stuff in `/boot` (eg, a custom kernel), including `/boot/config.txt`, you'll want to back that up too (pretty simple -- there's not much to it). Just do it separately, and when you restore, that stuff goes in the first partition.

Backup:

```
rsync -axHv --delete-during /boot/ /mnt/usbhd/pi_backup/boot/
```

Restore:

```
rsync -av --delete-during /mnt/usbhd/pi_backup/boot/ /mnt/sdcard_partition1/
```

Notice that there is a leading and trailing slash on `boot`

Using rsync, once a backup exists, updating it is much much faster, so you can set this up to happen painlessly everyday via cron.

You can use rsync even over a network using rsync via ssh
Here's an example:

```
rsync [options] --rsh="ssh [ssh options]" root@[the pi ip]:/ /backup/rpi/
```

"`Options`" would be, eg, `-av --delete-during --exclude-from=/rsync-exclude.txt` and "`ssh options`" is whatever you normally use (if anything). You must have root access via ssh to do this for the purposes of a system backup.

If you backup to an ext4 partition on a HDD mounted on the Pi and if the HDD is not mounted, rsync would copy to the mount directory until the SD Card is full.
If the HDD is not mounted in rw mode copious error messages are produced.
Neither of these is desirable, so let's check that the partition is mounted in rw mode before proceeding.

```
#!/bin/bash
# script to synchronise Pi files to backup
BACKUP_MOUNTED=$(mount | awk '/usbhd / {print $6}' | grep "rw")
if [ $BACKUP_MOUNTED ]; then
 echo $BACKUP_MOUNTED
 echo "Commencing Backup"
 rsync -avH --delete-during --delete-excluded --exclude-from=/rsync-exclude.txt /
/mnt/usbhd/pi_backup/
    rsync -axHv --delete-during /boot /mnt/usbhd/pi_backup/boot/
else
 echo "Backup drive not available or not writable"
fi
```

Restore (or update another Pi) with the following:

```
rsync -avH --delete-during /mnt/usbhd/pi_backup/boot/ /mnt/sdcard_partition1/
rsync -avH --delete-during /mnt/usbhd/pi_backup/ /
```

Backup & Restore your PI using rpi-clone

rpi-clone is a shell script that is for cloning a running Raspberry Pi booted source disk (SD card or USB disk as you can boot from USB stick as well) to a destination disk which will be bootable. Destination disks can be SD cards in the SD card slot or a USB card reader, USB flash disks, or USB hard drives.

I will only describe what I have tested succedful. See the README.md for detailed information. My aim was to clone the SD card to and SD card inserted in a USB slot with an adapter. This would give me a ready-to-replace bootable spare SD card of my PI.

Install
rpi-clone is on github and is downloaded by cloning the repository. It is a standalone script and the install is a simple copy to a bin directory. When run it checks its program dependencies and offers to install needed packages.

```
git clone https://github.com/billw2/rpi-clone.git
cd rpi-clone
chmod +x rpi-cl*
sudo cp rpi-clone rpi-clone-setup /usr/local/sbin
```

Make sure /usr/local/sbin is in your $PATH

```
echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
```

and then run rpi-clone or rpi-clone-setup with no args to print usage.

Note: rpi-clone-setup is for setting the different hostname in /etc/hostname and /etc/hosts files of the clone. So your backup SD card will have a different hostname. This can be handy if you want to setup several Raspberry Pi's eg: a training class and make sure all have a different hostname. It is run automatically by rpi-clone if -s args are given, but before your first clone using a -s option, test run rpi-clone-setup with:

```
sudo rpi-clone-setup -t testhostname
```

And check the files under /tmp/clone-test to be sure the files have been edited correctly.
If you need additional customizations to a clone, add them to the rpi-clone-setup script.

Usage
To get a usage screen showing available options, run rpi-clone without any arguments:

```
sudo rpi-clone
```

Typical two partition clones - SD card to USB disk (= backup of SD card to USB disk):

```
 rpi-clone sda
```

USB boot , clone back to SD card slot (= restore from USB disk to SD card) :

```
rpi-clone mmcblk0
```

Backing up my SD Card (with same hostname on backup)

Note 1: make sure you have installed the package usbmount. If not, run first the following

```
apt -y install usbmount
sed -i "s/^PrivateMounts=yes/PrivateMounts=no/" /lib/systemd/system/systemd-
udevd.service
```

Note 2: Remove any partition from the destination USB stick, create 1 FAT32 formatted partition on it. You can use the Diskmanager of Windows to do so.

Display your filesystem using the command

```
df -h
```

You should see something like this

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        14G  1.5G   12G  11% /
devtmpfs        841M     0  841M   0% /dev
tmpfs           970M     0  970M   0% /dev/shm
tmpfs           100M  8.4M   92M   9% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           970M     0  970M   0% /sys/fs/cgroup
tmpfs           100M     0  100M   0% /tmp
tmpfs            10M     0   10M   0% /var/tmp
tmpfs           100M  620K  100M   1% /var/log
/dev/mmcblk0p1  253M   53M  200M  21% /boot
tmpfs           194M     0  194M   0% /run/user/0
```

Now insert the USB stick in a USB slot, wait 10 seconds and type again

```
df -h
```

Now you should see that the USB drive is mounted as sda. If there are several partitions on the USB drive, you will see sda1, sda2, …

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        14G  1.5G   12G  11% /
devtmpfs        841M     0  841M   0% /dev
tmpfs           970M     0  970M   0% /dev/shm
tmpfs           100M  8.5M   92M   9% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           970M     0  970M   0% /sys/fs/cgroup
tmpfs           100M     0  100M   0% /tmp
tmpfs            10M     0   10M   0% /var/tmp
tmpfs           100M  632K  100M   1% /var/log
/dev/mmcblk0p1  253M   53M  200M  21% /boot
tmpfs           194M     0  194M   0% /run/user/0
```

```
/dev/sda1       253M   53M  200M  21% /media/usb0
```

Now we are ready to clone our PI

```
rpi-clone sda
```

Note: only type sda not sda1

```
  Destination disk partition /dev/sda1 is mounted on /media/usb0.
  The clone cannot proceed unless it is unmounted.
Do you want to unmount /media/usb0?  (yes/no): yes

Booted disk: mmcblk0 15.5GB               Destination disk: sda 15.8GB
------------------------------------------------------------------------------
Part      Size    FS     Label           Part   Size    FS     Label
1 /boot   256.0M  fat32  --              1       14.7G  fat32  --
2 root     14.2G  ext4   rootfs
------------------------------------------------------------------------------
== Initialize: IMAGE partition table - partition number mismatch: 2 -> 1 ==
1 /boot               (52.0M used)   : MKFS  SYNC to sda1
2 root                (1.5G used)    : RESIZE  MKFS  SYNC to sda2
------------------------------------------------------------------------------
Run setup script       : no.
Verbose mode           : no.
-----------------------:
** WARNING **          : All destination disk sda data will be overwritten!
-----------------------:

Initialize and clone to the destination disk sda?  (yes/no): yes
Optional destination ext type file system label (16 chars max): boot

Initializing
  Imaging past partition 1 start.
  => dd if=/dev/mmcblk0 of=/dev/sda bs=1M count=8 ...
  Resizing destination disk last partition ...
    Resize success.
  Changing destination Disk ID ...
  => mkfs -t vfat -F 32  /dev/sda1 ...
  => mkfs -t ext4 -L boot /dev/sda2 ...

Syncing file systems (can take a long time)
Syncing mounted partitions:
  Mounting /dev/sda2 on /mnt/clone
  => rsync // /mnt/clone with-root-excludes ...
  Mounting /dev/sda1 on /mnt/clone/boot
  => rsync /boot/ /mnt/clone/boot  ...

Editing /mnt/clone/boot/cmdline.txt PARTUUID to use f5f0de94
Editing /mnt/clone/etc/fstab PARTUUID to use f5f0de94
================================
Done with clone to /dev/sda
   Start - 17:43:37    End - 17:50:37    Elapsed Time - 7:00

Cloned partitions are mounted on /mnt/clone for inspection or customizing.

Hit Enter when ready to unmount the /dev/sda partitions ...
  unmounting /mnt/clone/boot
  unmounting /mnt/clone
================================
```

Appendix : Partitioning and formatting a USB disk using Raspbian (Buster)

Insert USB disk and find it

```
fdisk -l

Device         Boot  Start      End  Sectors  Size Id Type
/dev/mmcblk0p1        8192   532479   524288  256M  c W95 FAT32 (LBA)
/dev/mmcblk0p2      532480 30318591 29786112 14.2G 83 Linux

Disk /dev/sda: 14.8 GiB, 15836643328 bytes, 30930944 sectors
Disk model: STORAGE DEVICE
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: 0x039f1660
```

Next, partition it

```
fdisk /dev/sda
> n
> p
> 1
(size the partition)
> a
> 1
(toggles boot flag)
> t
> c
(filesystem type)
> p
(prints current configuration)
> w
(write the new table and quit)
```

Now format the partition

```
mkfs /dev/sda1
```