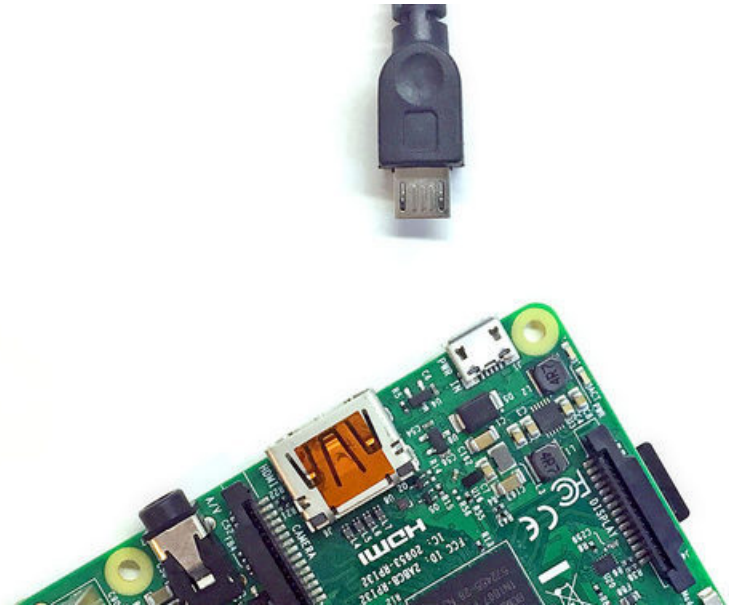
adafruit learning system

Read-Only Raspberry Pi

Created by Phillip Burgess



Last updated on 2019-06-07 01:29:52 AM UTC

Overview

Most microcontroller projects have an on/off switch or some quick way to cut power, while computers like the Raspberry Pi require an **orderly shutdown** procedure...otherwise the SD card may become **corrupted** and the system will no longer boot.

Sometimes just cutting power would be a convenient timesaver, or a system may be left to non-technical users. For installations that don't require creating or modifying files — such as a dedicated slideshow kiosk — we can configure the operating system to make it **more resistant to unplanned power cuts**.

The Concern

Linux — or any substantial computer operating system, Windows and Mac are the same way — behind the scenes they're reading and writing all manner of temporary data to drives (or the SD card with Raspberry Pi). This is why we normally use the **shutdown** command: all those files are put away in a known valid state. But if power is unexpectedly cut, these lingering half-written files can render a card **unbootable**...one can try patching it up, but sometimes there's no recourse but to **wipe the card and reinstall everything**.

The Solution

The script we provide here adapts Raspbian to work in a ***read-only*** mode. Temporary files are stored in RAM rather than on the SD card, making it more robust in this regard. You can just unplug the system when done. The tradeoff...as the *read-only* name implies...is that **nothing can be written to the card** in this state. Can't install software, can't record pictures. So it's *not* the solution to every situation, but may be helpful with certain passive tasks...a slideshow kiosk (https://adafru.it/zrA), a Fadecandy (https://adafru.it/zrB) server, a Halloween display (https://adafru.it/zrC), etc.

Optionally, you can use a jumper or switch to boot the system into normal read/write mode to install new software or data. And, as normal, you still have easy access to the /boot partition if the SD card is mounted on another computer.

This guide is based partly on instructions from petr.io (https://adafru.it/zrD), which in turn credits Charles Hallard (https://adafru.it/zrE) and Mario Hros (https://adafru.it/zrF)...along with community members' contributions in those threads and some changes and additions of our own.

Before You Begin

- This **does not work** with the **X11** desktop / PIXEL. It's strictly for **Raspbian Lite** right now. Graphical applications are still possible using SDL, Pygame and so forth, just not X11 at the moment.
- **This is *not* a "robust" solution for all Raspberry Pi tasks.** It messes up *crontab,* for instance. It's something we use for minor Raspberry Pi projects...RetroPie gaming, Halloween eyes, simple things of that nature.
- Setting up read-only mode should be the ***very last step*** before deploying a project. Get all your code and data on the system, get software auto-starting as needed, test it normally with the usual boot and shutdown methodology. It's easier up-front. Once you're 100% confident in its operation, *then* use the script.
- **Back up** the contents of your SD card first. We've tested on a couple versions of Raspbian, but maybe something's changed, or has been overlooked, and could leave the Pi in a weird intermediate state, or the script might break compatibility with some other software.
- No really, **back up your stuff**.

Install and Run

Pi should be booted and on the network...like mentioned above, everything already configured and fully functional (and

backed up) before taking this step.

THIS SEQUENCE IS IRREVERSIBLE. We don't have an uninstall script. There's an *option* to boot into read/write mode, but *nothing* to back out *all* these changes.

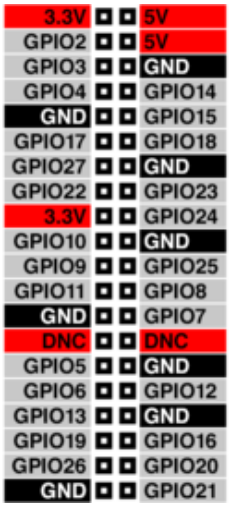
From a command line prompt:

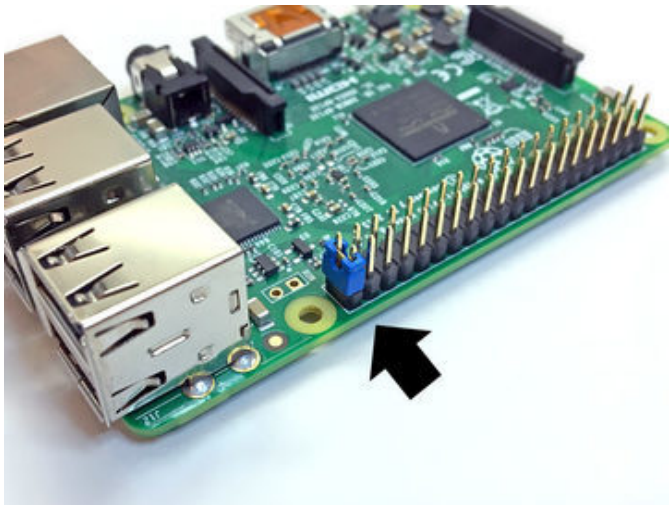
```
wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/read-only-fs.sh
sudo bash read-only-fs.sh
```

The script will repeat all these stern warnings and make you verify at several steps whether to continue. Along the way you'll be presented with a few options:

Enable boot-time read/write jumper? [y/N]

This gives you the option to run the system in read/write mode by inserting a jumper across two pins...





If you answer yes to this question, you'll also be asked for a **GPIO pin number**. When there's a jumper between this pin and **ground**, the system will boot into **read/write mode** and you can make changes (but remember to **do a proper shutdown**). Enter just the number, not the "GPIO" part.

Make sure the pin isn't used by anything else (like if you have a PiTFT display attached). **GPIO21** (enter "21") is easy to remember because it's right at the end of the header. If you're using **I2S audio** though, that requires GPIO21 for its own use, so you'll want to pick another.

Install GPIO-halt utility? [y/N]

This installs a utility that initiates a proper shutdown when another GPIO pin is touched to ground.

For a read-only system, **you probably don't need this**...but I'm a little paranoid...or, if you have the system booted in read/write mode, this provides an option if you can't log in and run a manual shutdown.

This likewise will ask for a GPIO pin number. Use the map above. How about GPIO16? (Enter "16") It's right next to a ground pin and we can use one of these button quick-connects.

Enable kernel panic watchdog? [y/N]

This option enables software to automatically reboot the system in the event of a kernel panic (a low-level system crash). This is actually pretty rare though, and is not the only way in which programs may crash on the system. You might want to set up your code to auto-restart using **systemd**, but that's a whole book in itself.

(The selections here are based on comments in the previously-mentioned blog posts...but in practice the impression I'm getting is that it may be related more to the OS version than the specific hardware. This option might change or be removed in the future.)

One last confirmation before the script runs. It may take 5 to 10 minutes, depending on the processor, network and SD card speed. You'll see a few warnings along the way, these can be ignored.

You're Not Finished Yet

Test the modified system to make sure that the system boots and your application runs as intended. Try a pass with the read/write jumper and/or the gpio-halt button, if you've enabled either of those options.

Now make an image of the SD card (using *dd* or *Apple Pi Baker* or whatever your backup tool of preference) and, if it's a critical application, **burn *at least* one spare**. There are other ways cards can go bad...static, brown-outs, falling out and getting lost...this read-only setup won't always save you. SD cards are cheap now! Spares help if you've left a system in someone else's care (let's say a museum kiosk) and it fails for some reason, you can ask them to just swap out the card until you can get there to troubleshoot. I know at least one Burning Man project rendered useless *in the first few minutes* of the event because their one and only card fell out and was lost on the playa.

Script code

Here's the code 'in line' in case you want to review it here (also, it will print in the PDF)

```
#!/bin/bash

# CREDIT TO THESE TUTORIALS:
# petr.io/en/blog/2015/11/09/read-only-raspberry-pi-with-jessie
```

```
# hallard.me/raspberry-pi-read-only
# k3a.me/how-to-make-raspberrypi-truly-read-only-reliable-and-trouble-free

if [ $(id -u) -ne 0 ]; then
 echo "Installer must be run as root."
 echo "Try 'sudo bash $0'"
 exit 1
fi

clear

echo "This script configures a Raspberry Pi"
echo "SD card to boot into read-only mode,"
echo "obviating need for clean shutdown."
echo "NO FILES ON THE CARD CAN BE CHANGED"
echo "WHEN PI IS BOOTED IN THIS STATE. Either"
echo "the filesystems must be remounted in"
echo "read/write mode, card must be mounted"
echo "R/W on another system, or an optional"
echo "jumper can be used to enable read/write"
echo "on boot."
echo
echo "Links to original tutorials are in"
echo "script source. THIS IS A ONE-WAY"
echo "OPERATION. THERE IS NO SCRIPT TO"
echo "REVERSE THIS SETUP! ALL other system"
echo "config should be complete before using"
echo "this script. MAKE A BACKUP FIRST."
echo
echo "Run time ~5 minutes. Reboot required."
echo
echo -n "CONTINUE? [y/N] "
read
if [[ ! "$REPLY" =~ ^(yes|y|Y)$ ]]; then
 echo "Canceled."
 exit 0
fi

# FEATURE PROMPTS ----------------------------------------------------------
# Installation doesn't begin until after all user input is taken.

INSTALL_RW_JUMPER=0
INSTALL_HALT=0
INSTALL_WATCHDOG=0

# Given a list of strings representing options, display each option
# preceded by a number (1 to N), display a prompt, check input until
# a valid number within the selection range is entered.
selectN() {
 for ((i=1; i<=$#; i++)); do
  echo $i. ${!i}
 done
 echo
 REPLY=""
 while :
 do
  echo -n "SELECT 1-$#: "
  read
  if [[ $REPLY -ge 1 ]] && [[ $REPLY -le $# ]]; then
   return $REPLY
```

```
   return $REPLY
  fi
 done
}

SYS_TYPES=(Pi\ 3\ /\ Pi\ Zero\ W All\ other\ models)
WATCHDOG_MODULES=(bcm2835_wdog bcm2708_wdog)
OPTION_NAMES=(NO YES)

echo -n "Enable boot-time read/write jumper? [y/N] "
read
if [[ "$REPLY" =~ (yes|y|Y)$ ]]; then
 INSTALL_RW_JUMPER=1
 echo -n "GPIO pin for R/W jumper: "
 read
 RW_PIN=$REPLY
fi

echo -n "Install GPIO-halt utility? [y/N] "
read
if [[ "$REPLY" =~ (yes|y|Y)$ ]]; then
 INSTALL_HALT=1
 echo -n "GPIO pin for halt button: "
 read
 HALT_PIN=$REPLY
fi

echo -n "Enable kernel panic watchdog? [y/N] "
read
if [[ "$REPLY" =~ (yes|y|Y)$ ]]; then
 INSTALL_WATCHDOG=1
 echo "Target system type:"
 selectN "${SYS_TYPES[0]}" \
  "${SYS_TYPES[1]}"
 WD_TARGET=$?
fi

# VERIFY SELECTIONS BEFORE CONTINUING -------------------------------------

echo
if [ $INSTALL_RW_JUMPER -eq 1 ]; then
 echo "Boot-time R/W jumper: YES (GPIO$RW_PIN)"
else
 echo "Boot-time R/W jumper: NO"
fi
if [ $INSTALL_HALT -eq 1 ]; then
 echo "Install GPIO-halt: YES (GPIO$HALT_PIN)"
else
 echo "Install GPIO-halt: NO"
fi
if [ $INSTALL_WATCHDOG -eq 1 ]; then
 echo "Enable watchdog: YES (${SYS_TYPES[WD_TARGET-1]})"
else
 echo "Enable watchdog: NO"
fi
echo
echo -n "CONTINUE? [y/N] "
read
if [[ ! "$REPLY" =~ ^(yes|y|Y)$ ]]; then
 echo "Canceled."
```

```
 exit 0
fi

# START INSTALL -------------------------------------------------------------
# All selections have been validated at this point...

# Given a filename, a regex pattern to match and a replacement string:
# Replace string if found, else no change.
# (# $1 = filename, $2 = pattern to match, $3 = replacement)
replace() {
 grep $2 $1 >/dev/null
 if [ $? -eq 0 ]; then
  # Pattern found; replace in file
  sed -i "s/$2/$3/g" $1 >/dev/null
 fi
}

# Given a filename, a regex pattern to match and a replacement string:
# If found, perform replacement, else append file w/replacement on new line.
replaceAppend() {
 grep $2 $1 >/dev/null
 if [ $? -eq 0 ]; then
  # Pattern found; replace in file
  sed -i "s/$2/$3/g" $1 >/dev/null
 else
  # Not found; append on new line (silently)
  echo $3 | sudo tee -a $1 >/dev/null
 fi
}

# Given a filename, a regex pattern to match and a string:
# If found, no change, else append file with string on new line.
append1() {
 grep $2 $1 >/dev/null
 if [ $? -ne 0 ]; then
  # Not found; append on new line (silently)
  echo $3 | sudo tee -a $1 >/dev/null
 fi
}

# Given a filename, a regex pattern to match and a string:
# If found, no change, else append space + string to last line --
# this is used for the single-line /boot/cmdline.txt file.
append2() {
 grep $2 $1 >/dev/null
 if [ $? -ne 0 ]; then
  # Not found; insert in file before EOF
  sed -i "s/\'/ $3/g" $1 >/dev/null
 fi
}

echo
echo "Starting installation..."
echo "Updating package index files..."
apt-get update

echo "Removing unwanted packages..."
#apt-get remove -y --force-yes --purge triggerhappy logrotate dbus \
# dphys-swapfile xserver-common lightdm fake-hwclock
# Let's keep dbus, that includes avahi-daemon, a la 'raspberrypi.local'
```

```
# Let's keep dbus...that includes avahi-daemon, a la "raspberrypi.local",
# also keeping xserver & lightdm for GUI login (WIP, not working yet)
apt-get remove -y --force-yes --purge triggerhappy logrotate \
 dphys-swapfile fake-hwclock
apt-get -y --force-yes autoremove --purge

# Replace log management with busybox (use logread if needed)
echo "Installing ntp and busybox-syslogd..."
apt-get -y --force-yes install ntp busybox-syslogd; dpkg --purge rsyslog

echo "Configuring system..."

# Install boot-time R/W jumper test if requested
GPIOTEST="gpio -g mode $RW_PIN up\n\
if [ \`gpio -g read $RW_PIN\` -eq 0 ] ; then\n\
\tmount -o remount,rw \/\n\
\tmount -o remount,rw \/boot\n\
fi\n"
if [ $INSTALL_RW_JUMPER -ne 0 ]; then
 apt-get install -y --force-yes wiringpi
 # Check if already present in rc.local:
 grep "gpio -g read" /etc/rc.local >/dev/null
 if [ $? -eq 0 ]; then
  # Already there, but make sure pin is correct:
  sed -i "s/^.*gpio\ -g\ read.*$/$GPIOTEST/g" /etc/rc.local >/dev/null

 else
  # Not there, insert before final 'exit 0'
  sed -i "s/^exit 0/$GPIOTEST\\nexit 0/g" /etc/rc.local >/dev/null
 fi
fi

# Install watchdog if requested
if [ $INSTALL_WATCHDOG -ne 0 ]; then
 apt-get install -y --force-yes watchdog
 # $MODULE is specific watchdog module name
 MODULE=${WATCHDOG_MODULES[($WD_TARGET-1)]}
 # Add to /etc/modules, update watchdog config file
 append1 /etc/modules $MODULE $MODULE
 replace /etc/watchdog.conf "#watchdog-device" "watchdog-device"
 replace /etc/watchdog.conf "#max-load-1" "max-load-1"
 # Start watchdog at system start and start right away
 # Raspbian Stretch needs this package installed first
 apt-get install -y --force-yes insserv
 insserv watchdog; /etc/init.d/watchdog start
 # Additional settings needed on Jessie
 append1 /lib/systemd/system/watchdog.service "WantedBy" "WantedBy=multi-user.target"
 systemctl enable watchdog
 # Set up automatic reboot in sysctl.conf
 replaceAppend /etc/sysctl.conf "^.*kernel.panic.*$" "kernel.panic = 10"
fi

# Install gpio-halt if requested
if [ $INSTALL_HALT -ne 0 ]; then
 apt-get install -y --force-yes wiringpi
 echo "Installing gpio-halt in /usr/local/bin..."
 cd /tmp
 curl -LO https://github.com/adafruit/Adafruit-GPIO-Halt/archive/master.zip
 unzip master.zip
 cd Adafruit-GPIO-Halt-master
```

```
 make
 mv gpio-halt /usr/local/bin
 cd ..
 rm -rf Adafruit-GPIO-Halt-master

 # Add gpio-halt to /rc.local:
 grep gpio-halt /etc/rc.local >/dev/null
 if [ $? -eq 0 ]; then
  # gpio-halt already in rc.local, but make sure correct:
  sed -i "s/^.*gpio-halt.*$/\/usr\/local\/bin\/gpio-halt $HALT_PIN \&/g" /etc/rc.local >/dev/null
 else
  # Insert gpio-halt into rc.local before final 'exit 0'
  sed -i "s/^exit 0/\/usr\/local\/bin\/gpio-halt $HALT_PIN \&\\nexit 0/g" /etc/rc.local >/dev/null
 fi
fi

# Add fastboot, noswap and/or ro to end of /boot/cmdline.txt
append2 /boot/cmdline.txt fastboot fastboot
append2 /boot/cmdline.txt noswap noswap
append2 /boot/cmdline.txt ro^o^t ro

# Move /var/spool to /tmp
rm -rf /var/spool
ln -s /tmp /var/spool

# Move /var/lib/lightdm and /var/cache/lightdm to /tmp
rm -rf /var/lib/lightdm
rm -rf /var/cache/lightdm
ln -s /tmp /var/lib/lightdm
ln -s /tmp /var/cache/lightdm

# Make SSH work
replaceAppend /etc/ssh/sshd_config "^.*UsePrivilegeSeparation.*$" "UsePrivilegeSeparation no"
# bbro method (not working in Jessie?):
#rmdir /var/run/sshd
#ln -s /tmp /var/run/sshd

# Change spool permissions in var.conf (rondie/Margaret fix)
replace /usr/lib/tmpfiles.d/var.conf "spool\s*0755" "spool 1777"

# Move dhcpd.resolv.conf to tmpfs
touch /tmp/dhcpcd.resolv.conf
rm /etc/resolv.conf
ln -s /tmp/dhcpcd.resolv.conf /etc/resolv.conf

# Make edits to fstab
# make / ro
# tmpfs /var/log tmpfs nodev,nosuid 0 0
# tmpfs /var/tmp tmpfs nodev,nosuid 0 0
# tmpfs /tmp     tmpfs nodev,nosuid 0 0
replace /etc/fstab "vfat\s*defaults\s" "vfat    defaults,ro "
replace /etc/fstab "ext4\s*defaults,noatime\s" "ext4    defaults,noatime,ro "
append1 /etc/fstab "/var/log" "tmpfs /var/log tmpfs nodev,nosuid 0 0"
append1 /etc/fstab "/var/tmp" "tmpfs /var/tmp tmpfs nodev,nosuid 0 0"
append1 /etc/fstab "\s/tmp"   "tmpfs /tmp    tmpfs nodev,nosuid 0 0"

# PROMPT FOR REBOOT ---------------------------------------------------

echo "Done."
echo
```

```
echo "Settings take effect on next boot."
echo
echo -n "REBOOT NOW? [y/N] "
read
if [[ ! "$REPLY" =~ ^(yes|y|Y)$ ]]; then
 echo "Exiting without reboot."
 exit 0
fi
echo "Reboot started..."
reboot
exit 0
```