Ubuntu 14.04 Trusty Tahr Is Here • Peek Inside Android Source Code

ODROID Magazine

Year One
Issue #3
Mar 2014







Linux Gaming on the ODROID


Making Money with ODROID Web Development

Set up a HPC Head Node

Keep your kernels up to date

Boot a Linux Distro from an USB Drive

We've had an exciting first two months here at ODROID Magazine, and it's been a pleasure to read the many awesome submissions from our diverse team of international authors. The articles have been well received by the community, with nearly 10,000 downloads in our first month. Now that we've introduced you to the $59 U3 powerhouse board, and shown you how to be the first on your block with a Giant Android tablet, we're going to explore the fun side of Linux: its amazing GAMES.

Remember those early 8- and 16-bit computer games with amazing gameplay, unique stories, and really catchy music? Arcade and console classics such as Donkey Kong, Pac-Man, Super Mario Brothers, Maniac Mansion, Mortal Kombat, and Star Wars all run great on the ODROID in stereo sound and HD graphics. Hold on to your joystick!

Premiering in ODROID Magazine this month is Nanik, who is our new Android Developer columnist. He brings with him a deep understanding of software development, and presents us this month with an in-depth look into the Android source code. Ronaldo is another recent addition as our Android Gaming expert, with reviews, tips, and guides to the thousands of Android games available from the Play Store and other sources.

Also joining the regular contributor circle is Manuel, our new Multilingual Editor, providing a full Spanish translation of each month's issue for the benefit of our international community. We are also very proud to introduce our Proof-readers Venkat and Fabien, who review and test the magazine articles before publication from a reader's perspective.

Look for an announcement on the ODROID Magazine forum at `http://forum.odroid.com` for more details on the upcoming Spanish version. Welcome aboard, Nanik, Ronaldo, Manuel, Venkat and Fabien!

ODROID Magazine

Rob Roy, Chief Editor

I am a computer programmer living and working in Silicon Valley, CA, USA, designing and building websites such as Vevo, Hi5, Dolby Laboratories and Hyundai. My primary languages are jQuery, Angular JS and HTML5/CSS3. I also develop pre-built operating systems, custom kernels and optimized applications for the ODROID platform based on Hardkernel's official releases, for which I have won several Monthly Forum Awards. I own a lot of ODROIDs, which I use for a variety of purposes, including media center, web server, application development workstation, and gaming console.

Bo Lechnowsky, Editor

I am President of Respectech, Inc., a technology consultancy in Ukiah, CA, USA that I founded in 2001. From my background in electronics and computer programming, I manage a team of technologists, plus develop custom solutions for companies ranging from small businesses to worldwide corporations. ODROIDs are one of the weapons in my arsenal for tackling these projects. My favorite development languages are Rebol and Red, both of which run fabulously on ARM-based systems like the ODROID-U2. I have deep experience with many unique operating systems.

Bruno Doiche, Art Editor

Fetched a Lapdock Altrix to play with his ODROIDS so now his fiancée can stop sending him away from the living room TV to watch Netflix.

ODROID Magazine, published monthly at http://magazine.odroid.com, is your source for all things ODROIDian. • Hard Kernel, Ltd. • 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815 • Makers of the ODROID family of quad-core development boards and the world's first ARM big.LITTLE architecture based single board computer.
Join the ODROID community with members from over 135 countries, at http://forum.odroid.com, and explore the new technologies offered by Hardkernel at http://www.hardkernel.com.

BOOTING A LINUX DISTRIBUTION FROM AN EXTERNAL USB DRIVE

THE MOST POPULAR ARTICLE REQUEST FROM OUR USER FORUMS

by Suriyan Ramasami

A frequent contributor to the ODROID community, Suriyan is well known for his updated ODROID bootloader, which permits booting into a Linux root file system located on a USB or network partition. He graciously shares his expertise in response to a highly requested forum topic: how to set up an ODROID, particularly the U2 and U3, with a minimal boot and root file system partitions, so that all other operating system files can be accessed from an external USB hard drive.

The Need for an eMMC or SD card

Although the root file system can be stored on a network or USB drive, an eMMC or SD card is required to store the boot loader related files. The ODROID hardware always looks for the boot loader, known as u-boot, on the local eMMC module or SD card.

The ODROID XU has a DIP switch which lets the user choose between eMMC and SD card for the boot media. The U2 and U3 models, on the other hand, always tries to boot from eMMC first, if a disk is present, and falls back to booting from the SD card if not.

Getting the image

The popular Xubuntu image for the ODROID U3, which is available for free download at `http://odroid.in/ubuntu-u2-u3/`, will be used in this article to demonstrate how to move the root file system to an external USB drive. The same methods apply to any similar ODROID distribution, including Debian, OpenSUSE, ALARM, and Ubuntu.

Although some flavors of Linux may contain simpler tools to keep the root file system on a separate drive, the approach shown here uses common tools that are present in most (if not all) distributions.

Gathering the Equipment

Choose any ODROID from the X, U or XU series along with a pre-flashed SD card containing the official Hardkernel Xubuntu image linked above. Any type of USB drive, such as a flash or external USB, can be attached to the ODROID for storing the external file system. To begin, boot up the default image in order to access the files involved in the boot process.

System Partitions

Let's look at the system partitions to get an understanding of where the boot and the root file system partitions reside.

```
root@odroid:~# df | grep mmc
/dev/mmcblk0p2   4489896 3954264    307556  93% /
/dev/mmcblk0p1    129039   12662    116377  10% /media/boot
root@odroid:~#
```

`/dev/mmcblk0p1` is the first VFAT partition which hosts the boot related files

`/dev/mmcblk0p2` is the EXT partition which is the root file system

The boot partition and the root file system both reside on the SD card, as indicated by the letters "mmc" in the device name.

Exploring the boot partition

```
root@odroid:~# ls -l /media/boot
total 5182
-rwxrwxrwx 1 root root     459 Dec 20 23:15 boot-1024x768-noedid.scr
-rwxrwxrwx 1 root root     450 Dec 20 23:15 boot-1080p-edid.scr
-rwxrwxrwx 1 root root     460 Dec 20 23:15 boot-1080p-noedid.scr
-rwxrwxrwx 1 root root     450 Dec 20 23:15 boot-720p-edid.scr
-rwxrwxrwx 1 root root     459 Dec 20 23:16 boot-720p-noedid.scr
-rwxrwxrwx 1 root root     380 Dec 20 23:16 boot-auto_edid.scr
-rwxrwxrwx 1 root root     380 Dec 20 23:16 boot.scr
-rwxrwxrwx 1 root root 2920148 Dec 20 23:15 uInitrd
-rwxrwxrwx 1 root root 2381904 Dec 20 23:14 zImage
```

`boot.scr:` The boot loader (u-boot) uses this file for its input variables

`boot-*.scr`: Sample files which can be used as replacements for boot.scr

`zImage`: Linux kernel

`uInitrd`: initial ramdisk used by the Linux kernel

The boot loader's job is to load zImage and uInitrd, then pass control to

zImage along with some boot parameters that are set in `boot.scr`.

Looking at the boot.scr file

The `root= variable` is passed as a parameter to the Linux kernel by the boot loader, instructing it to use the file system matching the given ID.

root assignments can take three forms:

```
root=UUID=...
root=LABEL=...
root=/dev/<device>
```

The first two variations can only be used if an init ramdisk is used along with the Linux kernel. The ArchLinuxArm distribution does not use an init ramdisk for booting, and so the third form should be used to ensure a that the boot loader remains compatible with ALARM.

In order to determine the UUID of the EXT partition, use the dumpe2fs command.

```
root@odroid:~# dumpe2fs /
dev/mmcblk0p2 |grep UUID
dumpe2fs 1.42.8 (20-Jun-
2013)
Filesystem UUID:
e139ce78-9841-40fe-8823-
96a304a09859
```

As shown above, the UUID listed should match the `root=UUID=...` parameter in boot.scr. If they don't match, Linux will not be able to identify the root file system, and will be unable to mount it.

The problem with the UUID approach is that, if a new file system is created and the root file system is copied over to it, it will fail to boot up, so the UUIDs need to remain manually synchronized.

A better approach is to use file system labels instead. To read the label of an existing EXT file system, use the e2label command.

```
root@odroid:~# e2label /dev/
mmcblk0p2
rootfs
```

To change the label of an existing file sytem, the tune2fs command should be used.

```
root@odroid:~# tune2fs -L
"RootFS" /dev/mmcblk0p2
tune2fs 1.42.8 (20-Jun-2013)

root@odroid:~# e2label /dev/
mmcblk0p2
RootFS
```

The form `root=LABEL=RootFS` will work as well, and is the most flexible method of identification, since the label can be easily changed using tune2fs.

Using the root=LABEL=... parameter

Consider the case where the ability to boot to different distributions is needed (Debian, Ubuntu, etc.) while using the same kernel for each of them.

When setting up the partitions, the first VFAT partition would remain unchanged, since it simply stores the boot loader files, along with a modified boot.scr containing the entry `root=RootFS`. In a triple-boot system, the second EXT partition could be used as the root file system for Ubuntu, and the third EXT partition as the root file system for Debian.

When using this ideal setup, switching between distributions involves simply changing the label of the intended partition to RootFS, and updating the other EXT partition labels to anything except RootFS. After rebooting, the partition with the RootFS label would be recognized as the root file system, and the corresponding Linux image would boot to its desktop or command prompt.

The following guide details the steps involved in customizing an ODROID in order to implement this ideal scenario. First, the boot loader must be modified to support USB drives.

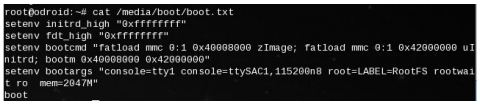
Modifying boot.scr

The newer boot loaders, such as the one included with the XU and the modified U2/U3, are able to read variables from a boot.ini file as a plain text file. However, the previous version of the Hardkernel boot loader read from a boot.scr file instead, which is a processed text file. Therefore, the boot.scr file needs an additional conversion step when modifying it.

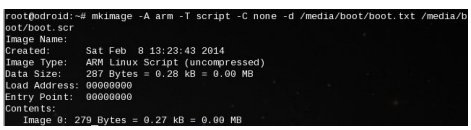
The utility mkimage is used for this purpose.

```
root@odroid:~# cp /media/
boot/boot.scr /media/boot/
boot.scr.org
root@odroid:~# strings /me-
dia/boot/boot.scr > /media/
boot/boot.txt
root@odroid:~# vi /media/
boot/boot.txt
```

Modify `boot.txt` with to match the file shown below. Note that the first two lines are deleted, and that the root= parameter has been changed.



Convert the boot.txt to a boot.scr using the utility mkimage.



It would seem that if the root file system is copied over to a partition in the USB drive and its file system label changed to RootFS, then a reboot would subsequently select that USB partition

as the root file system. However, there are two small obstacles:

1. The Linux kernel does not have USB storage access built into the kernel

2. The modules which enable USB storage access are not yet present in initrd.

If, in the future, Hardkernel distribution images default to having `USB_STORAGE` enabled in the kernel, or the init ramdisks already contain the USB storage modules, the following step can be skipped.

Exploring the uInitrd file

`initrd` is a gzip image, and uInitrd is a format recognized by the boot loader, called u-boot. U-boot presently has a 64 byte header, though this size can vary. Use the `mkimage -l uInitrd` command to determine the exact length.

Extract the gzip image from uInitrd and gunzip it. It is a cpio archive.

```
root@odroid:~# dd if=/media/boot/uInitrd of=/media/boot/initrd.gz bs=1 skip=64
2920084+0 records in
2920084+0 records out
2920084 bytes (2.9 MB) copied, 16.7887 s, 174 kB/s
root@odroid:~# gunzip /media/boot/initrd.gz
root@odroid:~# file /media/boot/initrd
/media/boot/initrd: ASCII cpio archive (SVR4 with no CRC)
```

Once it's uncompressed, you can view and modify the files. The next step should be performed using an EXT partition, instead of the VFAT partition where it will eventually reside.

```
root@odroid:~# cp /media/boot/initrd .
root@odroid:~# mkdir tmp
root@odroid:~# cd tmp
root@odroid:~/tmp# cpio -id < ../initrd
11594 blocks
root@odroid:~/tmp# ls
bin  conf  etc  init  lib  run  sbin  scripts
```

The goal here is to update the initrd ramdisk image to include the modules needed for the Linux kernel to mount the root file system from the USB storage.

A rebuild of kernel or modules isn't required, since all the modules, with the correct versions, are already present in the current root file system.

The modules required are: `usb_storage, sd_mod, scsi_mod`. They are located in the current root file system in `/lib/modules`:

`/lib/modules/3.8.13.14/kernel/drivers/usb/storage/usb-storage.ko`

`/lib/modules/3.8.13.14/kernel/drivers/scsi/sd_mod.ko`

`/lib/modules/3.8.13.14/kernel/drivers/scsi/scsi_mod.ko`

```
root@odroid:~/tmp# mkdir -p lib/modules/3.8.13.14/kernel/drivers/usb/storage
root@odroid:~/tmp# mkdir -p lib/modules/3.8.13.14/kernel/drivers/scsi
root@odroid:~/tmp# cp /lib/modules/3.8.13.14/kernel/drivers/usb/storage/usb-stor
age.ko lib/modules/3.8.13.14/kernel/drivers/usb/storage/
root@odroid:~/tmp# cp /lib/modules/3.8.13.14/kernel/drivers/scsi/sd_mod.ko lib/m
odules/3.8.13.14/kernel/drivers/scsi/
root@odroid:~/tmp# cp /lib/modules/3.8.13.14/kernel/drivers/scsi/scsi_mod.ko lib
/modules/3.8.13.14/kernel/drivers/scsi/
root@odroid:~/tmp# depmod -b ./ -a
```

Copy these over to the initrd tree and update the module information by running depmod. It is essential to run depmod in the initrd tree as it updates many files related to loading modules.

Regenerating uInitrd

```
root@odroid:~/tmp# find . | cpio --create --format='newc' > /media/boot/initrd
18307 blocks
root@odroid:~/tmp# gzip /media/boot/initrd
root@odroid:~/tmp# mkimage -A arm -O linux -T ramdisk -C gzip -a 0x0 -e 0x0 -n i
nitramdisk -d /media/boot/initrd.gz /media/boot/uInitrd
Image Name:   initramdisk
Created:      Sat Feb  8 13:44:16 2014
Image Type:   ARM Linux RAMDisk Image (gzip compressed)
Data Size:    4268804 Bytes = 4168.75 kB = 4.07 MB
Load Address: 00000000
Entry Point:  00000000
```

The init ramdisk now contains the required USB storage-related modules.

Select the USB drive as the root filesystem

The USB drive requires some preparation before storing the root file system. First, changing its label to RootFS, as mentioned above. The label of the current root file system should also be changed from RootFS to RootFS.org, so that two EXT file systems do not end up with the same label.

Prepare the USB drive

To avoid losing data, it is recommended to work with a blank USB drive. Plug it into the ODROID in order to create a new partition. In this example, a 12 GB partition was created using fdisk as the first partition on the drive.

In these screenshots, the USB drive has been assigned to `/dev/sda1`. To verify the applicable device name on your local system, run the dmesg command and inspect the output.

```
root@odroid:~# dmesg |grep "Attached SCSI"
[    7.314564] sd 0:0:0:0: [sda] Attached SCSI removable disk
root@odroid:~# dmesg | grep sda:
[    7.310706]  sda:
[  402.286178]  sda: sda1
root@odroid:~#
```

Next, create an EXT file system on the USB drive.

```
root@odroid:~# mkfs.ext4 /dev/sda1
mke2fs 1.42.8 (20-Jun-2013)
Filesystem label=
OS type: Linux
( … )
Writing superblocks and filesystem accounting information: done
root@odroid:~#
```

Change the label of the USB drive partition to RootFS, then mount the partition as /dst, as shown below.

```
root@odroid:~# tune2fs -L RootFS.org /dev/mmcblk0p2
tune2fs 1.42.8 (20-Jun-2013)
root@odroid:~# e2label /dev/mmcblk0p2
RootFS.org
```

Preparing the source root file system

The image file xubuntu-13.10-desktop-armhf_odroidu_20140107.img will be used to extract the root file system. Because of space considerations, it was first copied over to the USB partition created in the previous step.

```
root@odroid:~# cd /dst
root@odroid:/dst# ls
lost+found  xubuntu-13.10-desktop-armhf_odroidu_20140107.img.xz
root@odroid:/dst# xz -d xubuntu-13.10-desktop-armhf_odroidu_20140107.img.xz
root@odroid:/dst# ls
lost+found  xubuntu-13.10-desktop-armhf_odroidu_20140107.img
```

To write the image file to the partition, use the utility kpartx, which may need to be installed with the command apt-get install kpartx. For ArchLinuxArm users, it should be built from source, located at `http://christophe.varoqui.free.fr/`.

```
root@odroid:/dst# kpartx -av xubuntu-13.10-desktop-armhf_odroidu_20140107.img
add map loop0p1 (254:0): 0 262144 linear /dev/loop0 4096
add map loop0p2 (254:1): 0 9123840 linear /dev/loop0 266240
```

For this Xubuntu example, loop0p1 is the VFAT boot partition, and loop0p2 is the root file system that needs to be copied over to the USB partition.

Mounting and copying the source root file system to the USB partition

```
root@odroid:/dst# cd
root@odroid:~# mkdir /src
root@odroid:~# mount /dev/mapper/loop0p2 /src
root@odroid:~# cp -pr /src/* /dst
```

Changing the label of the current root file system

The current file system needs to be called something other than RootFS, since that label will be reserved for the bootable partition. RootFS.org is a good alternative label.

```
root@odroid:~# tune2fs -L RootFS /dev/sda1
tune2fs 1.42.8 (20-Jun-2013)
root@odroid:~# e2label /dev/sda1
RootFS
root@odroid:~# mkdir /dst
root@odroid:~# mount /dev/sda1 /dst
```

Cleaning up

```
root@odroid:~# umount /src
root@odroid:~# cd /dst
root@odroid:/dst# kpartx -d xubuntu-13.10-desktop-armhf_odroidu_20140107.img
loop deleted : /dev/loop0
root@odroid:/dst# cd
root@odroid:~# umount /dst
```

Reboot

After a power cycle, the ODROID will boot into the root file system located on the USB drive. If it does not, go back and check to see if any of the steps were overlooked. It may take a few attempts before getting everything to work together perfectly.

Verifying the changes

Once the ODROID has finished booting, there are two things to be verified:

Did the boot.scr pass the correct root= parameter to the Linux kernel?

Is the USB drive really being used as the root filesystem?

Guidelines

Any ODROID distribution developer who wishes to facilitate storage of the root file system on an external USB drive should use this quick checklist:

```
root@odroid:~# cat /proc/cmdline | grep root=LABEL=RootFS
console=tty1 console=ttySAC1,115200 mem=2047M fbcon=map:1111111111111111111111
1111 console=tty1 console=ttySAC1,115200n8 root=LABEL=RootFS rootwait ro  mem=20
47M
root@odroid:~# blkid -o list
device      fs_type label    mount point    UUID
------------------------------------------------------------------------
/dev/mmcblk0p1
            vfat    BOOT     /media/boot    F335-39E7
/dev/mmcblk0p2
            ext4    RootFS.org (not mounted) e139ce78-9841-40fe-8823-96a394a09850
/dev/sda1   ext4    RootFS   /              a2f8e79b-772f-4b6a-bf83-17c40fec7720
root@odroid:~#
```

Use root=LABEL= in the boot.scr script.

Build a kernel with USB storage support built in.

If the kernel does not include built in USB storage support, bundle the init ramdisk with the USB storage related modules.

Now enjoy using your Linux in whichever external USB disk you need.

USE YOUR ODROID AS A PLAYSTATION 3 MEDIA SERVER

CHILL OUT IN STYLE BETWEEN GAMING SESSIONS

by Bruno Doiche

Don't be fooled by their misleading project name! The Playstation media server will allow your ODROID to support lots of devices.

At this point, many ODROIDians have already put two or more of these amazing computers to good use, probably with one ODROID at the home office and at least one more plugged into a TV set.

But what if you are a single ODROID owner, happily using it on your home office, but also hoping to access your multimedia content from your living room?

The ODROID's portability is a great advantage, so you could just pack it up and get your setup running on your TV. It's just a matter to get it plugged into the HDMI port, turning it on, and launching XBMC, right?

But what do you do if you are too lazy or busy to disconnect your USB drives, disassemble your robotics project and take them all the way to the TV set (mine is a daunting 5 yards away)?

Additionally, you may already have your TV connected to a Playstation 3 or another device such as:

- Microsoft XBOX 360
- Sony Bravia
- Google Android
- Freebox HD
- Freecom MusicPal
- Pioneer Kuro
- Philips Aurea
- Philips Net TV
- Popcorn Hour
- Asus O!Play
- Xtreamer
- AC Ryan PlayOn!HD
- Brite-view CinemaTube
- Samsung TVs
- Philips Streamium
- Western Digital WD TV Live
- XBMC Media Center
- Boxee

The answer is that you can make your ODROID work as an incredible media server, overcoming the insurmountable distance from your home office to your living room. By following my guide, I'll show you how to do exactly that!

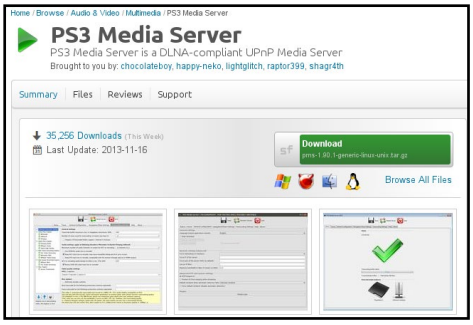
The right tool for the right job

Although the setup and implementation of a PS3 Media Server is far from rocket science, transcoding and sending the files over your network can sometimes be problematic. So to keep things working smoothly, I suggest the following rules of thumb:

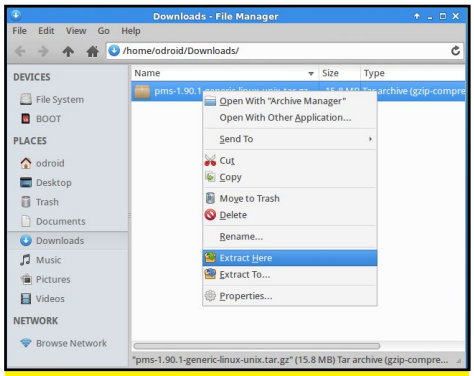
For any quad-core ODROID (X2, U2, U3), you can transcode videos up to 720p using cabled ethernet, but you may experience video stuttering if you try to use wireless ethernet, or upgrade the resolution to 1080p; For any octa-core ODROID (XU, XULite) you can set the resolution up to 1080p using the gigabit ethernet adapter, or use a wireless setup as described in the "XU Wireless Router" article in the Feb 2014 issue of ODROID magazine.

By keeping these rules in mind, I always experience great movies and video playback on my ODROID.

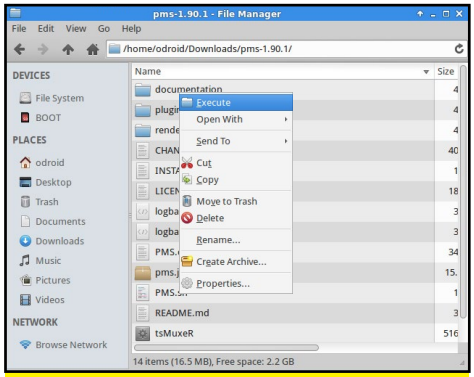
Go to www.ps3mediaserver.org/



The download and saving process is pretty straightforward but beware, if you download from your windows PC or your Mac, check that you are really getting the Linux version!



Extract your .tar.gz file by right-clicking and selecting a folder for uncompression.



Locate the PMS.sh file, and right-click it to run the script.

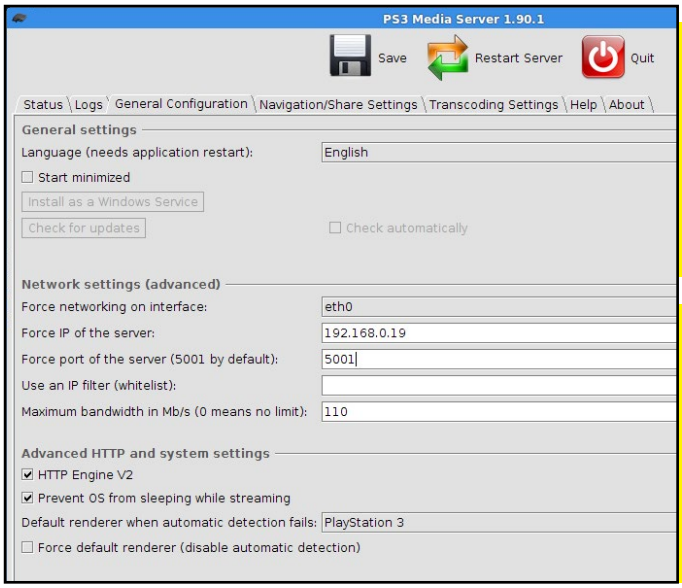
And that's it! Now that you have your Playstation Media Server running, it still needs to connect to the network. Doing so is a simple matter of matching the configuration options shown in these screenshots.

The 3 possible status

Scanning: the server is probing the network to find a device for you, may take a while if your odroid is firewalled.

No deal: check your network, firewall rules, and specially if your client device is on!

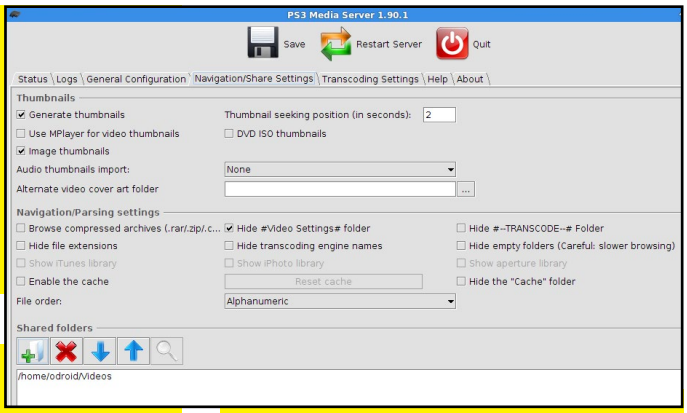
Success: hurry, get the popcorn and the soda!



The default configurations on the ODROID Linux-based builds is the lxcbr interface. Set the default network to the cabled network interface (in this case, eth0) instead.

The IP whitelist can be used in case you need to limit the number of machines connected to your media server. This guarantees that no one will be piggybacking on your server while you are using it.

The server defaults video sharing to use the root directory. To avoid this, change your shared folder to the location of your videos and songs to save yourself from having to navigate to deep directories.



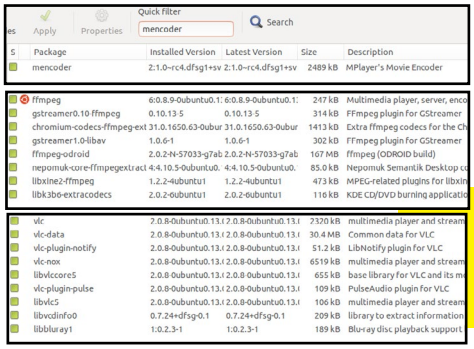
Whenever you change the configuration options, remember to save your settings, then press the "Restart Server" button. It seems obvious, but this step is easily overlooked.

Never hide the #TRANSCODE folder, since this may hinder your ability to choose from different transcoding engines on the PS3 in order to select specific subtitles or audio.

Don't forget to edit your hostname to avoid the sight of seeing your beloved media server going by the drab name of "localhost.

Getting another transcoding engine

It never hurts to have options, and sometimes a specific transcoding engine is more effective at handling a certain video or audio file than another. To guarantee that everything goes smoothly, do yourself (and your ODROID) a favor, and use your package manager to install the most recent version of Mencoder, VLC and ffmpeg. To play a media file using a different encoding engine on the PS3, go to the #TRANSCODE folder, and select another option.



Media decoder engines galore! Although Mencoder does the job, FFMPEG and VLC tend to save the day.

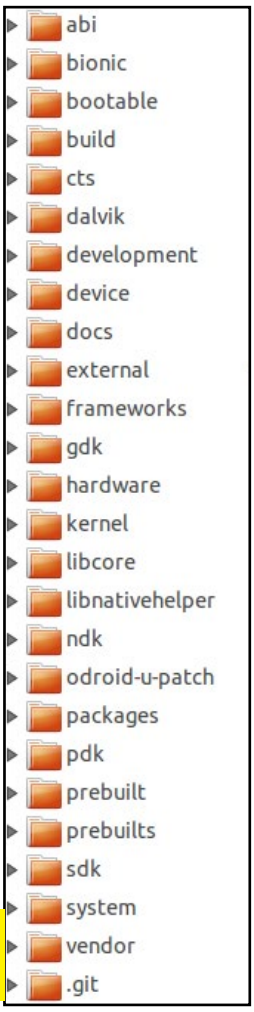
ANDROID DEVELOPMENT

INSPECTING THE ANDROID SOURCE CODE UNDER A MICROSCOPE

by Nanik Tolaram

When discussing Android development, the first thing usually mentioned is the sheer size of the code base, and how difficult it can be to navigate through the different parts of the Android source. To give you an idea of how big Android is, the current size on my local drive for the ODROID-specific version of Android 4.1.2 (excluding Linux kernel) is 8.6GB. When confronted with a massive code base like Android, the easiest approach is to break it down into smaller pieces for ease of understanding. Android is like a gigantic jigsaw puzzle that can be packed and unpacked again as needed.

This article presents an overview of the different directories located inside the source code, describes what projects each directory contains, and what kind of useful things you will find in each area. It is surprising how much can be learned and extracted simply by browsing the source code.



High Level Directory structure of Android source code

A Bird's Eye View

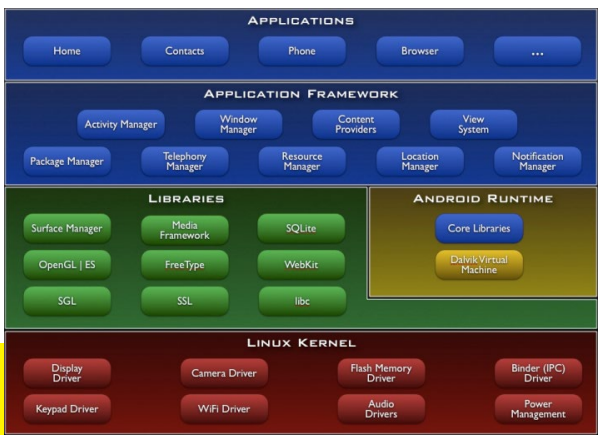
Here is the top level directory structure of the Android source code.

Altogether, there are 23 main Android directories, however, this number changes from version to version.

Architecture Details

In order to understand what role the different source code pieces play in the overall Android architecture, we will map each role shown to you below to a directory.

The source code will be associated with this diagram by using the labels that differentiate each layer, such as Applications, Application Framework, Libraries, and Android Runtime.



Android Architecture

Under the hood

Let's start putting the source code under the microscope by looking at what each directory contains.

abi/ (Libraries)

This directory stores Gabi++, a new minimalistic C++ runtime that provides the same headers as the system one, with the addition of RunTime Type Information (RTTI) support. The content of this directory is useful if you are planning to write applications using the C++ language.

bionic/ (Libraries)

If you have done any C programming in Linux, then you are familiar with the BSD C library. However, because the BSD library is big in terms of size, Android uses a port of that library called Bionic. Bionic is a stripped down version of the original BSD C library, and supports x86 and ARM architecture instruction sets. This is the core library upon which all Android code depends.

bootable/ (Booting)

Most Linux users are familiar with the GRUB bootloader that runs on an x86 PC, and in the ARM embedded world, a similar bootloader, with a smaller footprint, is used. This directory normally stores the bootloader code for use with your embedded device, such as u-boot or one of its derivatives. Android devices contain a special partition called "recovery", which is technically a self-contained application that includes a Linux kernel with which a user can performance maintenance, troubleshooting or upgrades to Android. This "recovery" app is inside the subdirectory called `recovery/` .

Inside /recovery

build/ (Build)

The complex nature of Android warrants its own build system. This directory contains all of the scripts (Shell, Python and Makefile) that are needed to build the source code from its various directories and package them together into a single set of image files. On completion of the build process, Android is reduced to several .img files (for ARM) and a single .iso (for x86). Figure 3 shows the different relevant sub-directories inside the recovery area.

Inside /build

cts/ (Test)

This directory contains the compatibility test suites (CTS) that allow device manufacturers to test whether a particular device is Android compatible. Test cases are written in a language called Junit, which provides direct access to the Android testing APIs. To learn more about the CTS, please refer to the Android CTS website at `http://source.android.com/compatibility/cts-intro.html`.

dalvik/ (Android Runtime)

This directory contains the complete source code for the Dalvik virtual machine. Besides Dalvik, it also contains several useful tools that are related to profiling, tracedump and many more. The Dalvik core itself lives inside the `vm/` subdirectory.

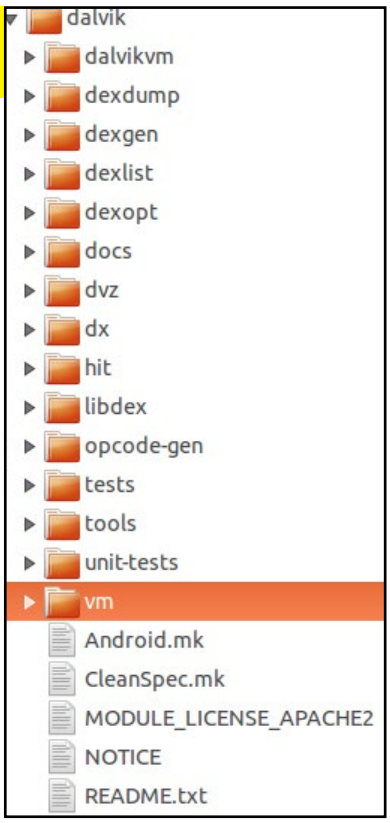
development/

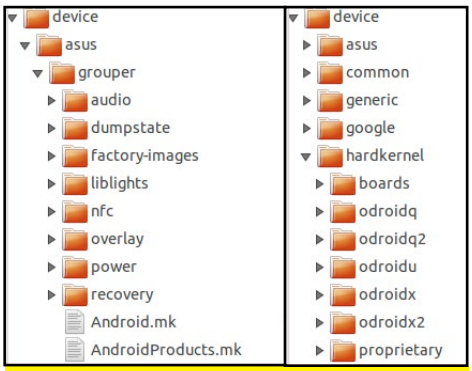
This directory is of interest to Android application developers, since it contains a number of sample applications that can be re-used or extended. There are also several useful tools inside this directory such as the Android APK checker, an HTTP server for testing, and many others. App developers will want to take a look at `apps/` and `samples/` directory for example projects.

device/ (Build)

Android runs on hundreds of devices, and each device has a unique configuration in terms of hardware and peripherals. Device configurations and scripts that are specific to a particular device are stored inside this directory, and contains files for different hardware. For example, below look at the differences between the files required for Nexus 7 devices (left) and the files required for Hardkernel devices (right).

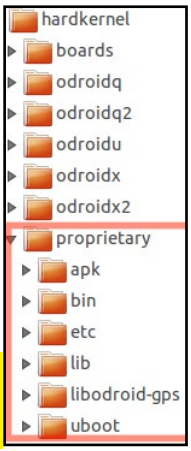
Inside /dalvik





Inside the /device directory

If you look inside the `proprietary/` folder, you will find a number of binary drivers that are used specifically for the ODROID platform.



hardkernel proprietary/

Docs/ (Document)

This directory contains documentation for the Android framework and API. The files are in raw format that are converted to HTML during the build process.

external/ (Libraries)

After the kernel, this directory is the most complex in terms of its source code. It contains all the different open source projects that Android relies on for its existence. All applications running in Android, either directly or indirectly, use some of the libraries inside the "external" directory.

frameworks/ (Application Framework)

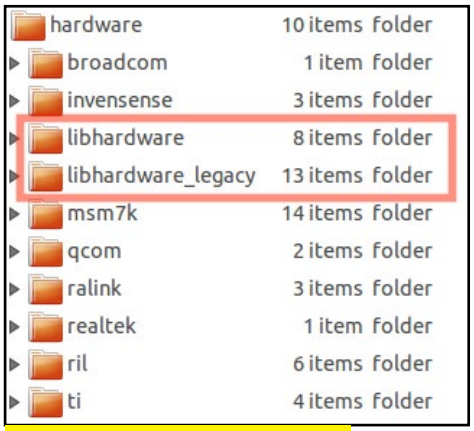
The "frameworks" directory is the the heart of Android system, and contains a combination of applications, SDK, APIs, utilities and much more. The level of code complexity is similar to the `external/` directory. If you ever need to customize Android, or want to learn how the whole application stack works, this is the place to look. Most of the user space applications reside here.

gdk/

This is an experimental directory that was introduced in Android 4.1.2, but does not exist after version 4.1.2. It contains llvm and CLANG code, which is not currently being used inside the Android framework. Due to the experimental nature of this directory, it can be safely ignored.

hardware/ (Build)

This directory contains the Hardware Abstraction Layer (HAL). The HAL permits vendors that do not provide open source drivers to supply their own pre-compiled binary drivers. The two main directories that provide the Android HAL are shown in Figure 8. The rest of the directories contain source code for the user space HAL that is made available in the Android repository.



HAL layer source code directory

libcore/ (Android Runtime)

The main focus of this directory is to house the core library used by the framework, as well as header files that are used by native code when using the Java Native Interface (JNI). Other subdirectories contain libraries such as json, luni ("`lang util net io`") and dalvik system utilities (dexfile and vmruntime).

libnativehelper/ (Libraries)

Android provide the flexibility of writing apps in Java and interfacing with native code with the help of the Java Native Interface (JNI). This library contains the module called libnativehelper that is used internally by Android to interface between Java and the native world. The integration layer is a simple JNI abstraction tool to make integration easier.

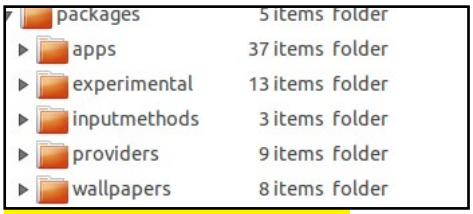
ndk/ (Libraries)

The Native Development Kit is normally used to develop Android applications using native code, and this directory contains the NDK source code. It includes tools needed for building NDK applications, including template makefile for building the native code on different platform such as ARM, MIPS, and x86. A few additional tools such as make, sed and toolbox can be also found inside this directory.

packages/ (Applications)

All of Android's prebuilt applications such as Calculator, Launcher and Settings are found here. This directory is a goldmine for application developers who wants to understand how applications are interacting with the system services such as the network, phone, sms, and accelerometer. The `apps/` subdirectory contains most of the applications, while the `experimental/` directory contains experimental applications which do not get included in the final image file.

The `inputmethods/` subdirectory contains input applications such as keyboard, mouse, touch, etc. As expected, the `wallpapers/` subdirectory contains wallpaper applications and resources.



Inside packages/

prebuilt/ (Build)

The content of this directory is slightly different than `prebuilts/`, but does not exist after version 4.1.2. However, it does contain the GCC 4.4.3 toolchain and some jar files that are used during the build process.

prebuilts/ (Build)

This directory contains the binaries for the toolchain suite that is used to compile Android source code to an image file. Due to licensing issues, the JDK is not part of this directory, but instead contains the toolchain that is included with the GNU compiler. The compiler supports both ARM and x86 architecture, and the main toolchain resides in the `gcc/linux-x86/toolchain` directory for Linux and `gcc/darwin-x86/toolchain` for Mac. Additionally, the directory contains the prebuilt kernel for the qemu emulators inside the qemu-kernel directory.

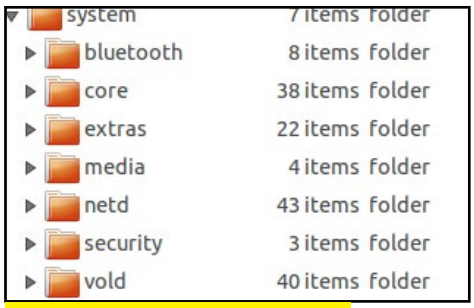
sdk/ (Tools/Build)

The Android development kit not only consist of libraries and API that a developer can use, but also includes

a number of tools, apps and scripts. This directory contains many auxiliary programs such as the SDK launcher, Traceview, and more.

system/ (Application Framework)

This directory contains the libraries and applications that form part of the core Android framework. Different global services that are made available to all applications, such as bluetooth, volume mounting, security and vold are located here.



Inside system/

vendor/ (Build)

Vendor related hardware drivers that are provided as binary files are stored here. Normally, the directory contains a subdirectory indicating which hardware is supported. All the necessary binary objects, including configuration files, are available in the "vendor" directory.

As an Android developer, you will most likely need to learn about more than one aspect of Android development, and thus you will be looking into the various directories for support, examples and documentation. If you like to design applications, you will mostly be interested in the packages directory to understand how the internal built-in application uses the API, or to discover if there are any hidden APIs that you can leverage. However, if you are a platform developer who would like to customize Android for a particular vertical market, then the framework and system directory will be of interest.

Further Reading

Regardless what you are trying to achieve with Android, there many educational benefits from studying the source code (both native and Java) and understanding how the whole stack works together. To explore more in depth about the Android source code components, please visit my eLinux wiki page, which details the different subdirectories inside the Android source code: `http://elinux.org/Android_Source_Code_Description`.

KEEPING YOUR ODROIDS UP TO DATE

DON'T MISS THE CHANCE TO BE RUNNING THE LATEST AND GREATEST KERNEL RELEASE

by Rob Roy, Chief Editor

Did you know that Hardkernel publishes nightly builds of its custom ODROID kernels, so that you can easily keep your personal system updated with the latest software improvements? The kernel packages are built on an ODROID-XU directly from the GitHub source code, and then uploaded to the Hardkernel website for your convenience.

In order to update your image with the latest Linux kernel, directly from the desks of the Hardkernel developers, download the kernel installation script from `http://builder.mdrjr.net/tools/kernel-update.sh`, and initiate the update from any Terminal window by typing:

```
$ wget builder.mdrjr.net/\
  tools/kernel-update.sh
$ sudo sh kernel-update.sh
```

Once the script has completed, reboot the ODROID so that the new kernel may take effect. The main supported operating systems include Ubuntu, Fedora, OpenSUSE, Debian, and Ubuntu Server, but the script can be easily modified for any distribution.

The script also automatically detects the ODROID platform (U, X, or XU) and installs the appropriate kernel version. Should you experience any issues with the upgrade, a backup of the kernel files is stored as a .tgz file in the `/root/` directory for quick recovery.

In addition to updating the kernel, it's also important to update the packages included on your Ubuntu or Debian distro.

To do so, connect to the official software repository and download the latest package updates using the following command:

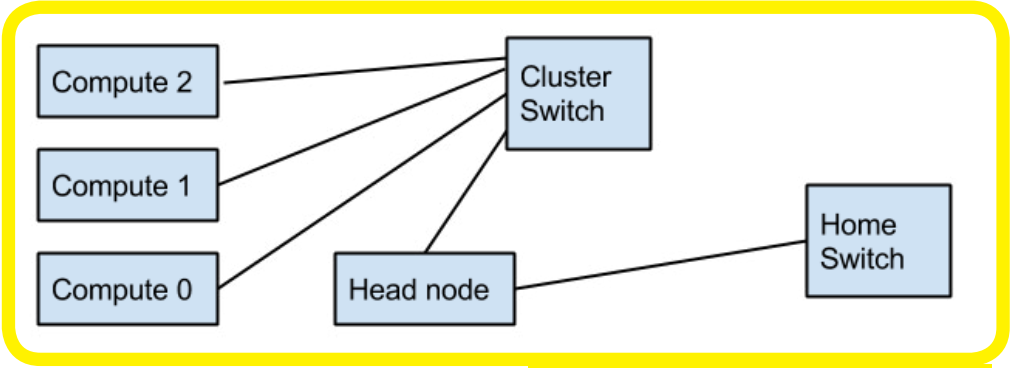
```
$ sudo apt-get update \
&& sudo apt-get dist-upgrade\
&& sudo apt-get autoremove
```

Ubuntu updates are released daily, and it's a good idea to update your kernel and software as often as possible.

HIGH PERFORMANCE COMPUTING AT HOME

SETTING UP AN HPC HEAD NODE

by Cooper Filby and Anthony Skjellum - Runtime Computing Solutions LLC



Looks like a simple thing to do right? Well, after reading this article, it will be even simpler for you.

In the February 2014 issue of ODROID Magazine, we started our series on efficient and inexpensive High Performance Computing achievable from the comfort of your own home. We began by describing the process of setting up a headless cluster for running MPI-based parallel programs. In this article, we detail the networking configuration required to set up a dedicated head node for our cluster using iptables to configure NAT, and DNSMasq to configure DHCP and DNS services for our cluster.

What you'll need for this article:

- **2x ODROIDs - in our examples, we will be using XU+Es running Ubuntu 13.09 server. More ODROIDs can easily be included as well, to create a bigger cluster.**
- **1x Ethernet Switch (preferably Gigabit Ethernet, also called 1000-BaseT)**
- **3x Ethernet Cables (plus 1 cable for each additional ODROID)**
- **1x USB Gigabit Ethernet Adapter (ideally 1 for each ODROID)**

Cluster Overview

Up above, we illustrate a sample network connection for our ODROID cluster in order to diagram how the listed components will be used. In our finished set up, all of our compute nodes will be connected to our dedicated cluster switch, either using onboard or USB ethernet.

The head node will use its USB ethernet to connect to the cluster switch, and use its onboard ethernet to connect to the home network switch. Compute nodes will be able to communicate with the home network and outside world through the head node.

Connecting the Head Node

As outlined above, we need to wire our head node to be multihomed, that is, connected to two networks, the home network and our cluster network. On the node you wish to use as your head node (in this case the ODROID XU+E we have named odroid-server0 as in part 1 of our series), connect the onboard ethernet device to your home network, and then connect the USB ethernet dongle to the switch for your cluster. Both of our network interfaces on our head node should be configured to allow communication with both the compute nodes and the outside world.

Editing `/etc/network/interfaces`, we need to set up eth0 to use the home network's DHCP, and eth2, the USB Ethernet Adapter, to communicate with the rest of the cluster. For eth0, your entry should look as follows:

```
auto eth0
iface eth0 inet dhcp
hwaddress ether
1a:e7:ed:f2:ff:73
```

Where the MAC address (in this case `1a:e7:ed:f2:ff:73`) can be of your choosing. For eth2, we will want to designate its static networking information, since it will be hosting a our DHCP and DNS service for the rest of our cluster. The exact information we use may vary by user, but for the purposes of this article we're going to assume that our cluster network is `192.168.128.0` through `192.168.128.24`.

Thus, we will configure eth2 as follows:

```
auto eth2
iface eth2 inet static
address 192.168.128.254
netmask 255.255.255.0
```

Once we run sudo service networking restart, eth0 and eth2 will be configured to connect and communicate with our two separate networks; eth0 will be assigned an IP address from your home router, while eth2 will get the IP specified in `/etc/network/interfaces` (which is `192.168.128.254` in this example). Next, we will configure `/etc/hosts` with our static IP as follows:

```
127.0.0.1 localhost
192.168.128.254 odroid-server0.ocluster odroid-server0
```

Where ocluster is the chosen name of our cluster domain and odroid-server0 is the chosen name of our head node. You can pick a different hostname for your head node as long as you update `/etc/hosts` accordingly.

Configuring NAT

A good next networking question to ask is this: What is Network Address Translation (NAT), and why is it important to our cluster? When it comes understanding NAT, it's essential to know how communication occurs between networks and hosts, and the two main types of IP addresses: public and private. Public IP addresses are assigned by a central authority and delegated to you, such as by your Internet Service Provider (ISP). They can be used to send traffic externally between networks or hosts with a public IP on the Internet. On the other hand, private IP addresses are for internal usage within a local area network (such as your home network).

Private IP addresses don't make sense on the public Internet; they can only be directly used to send traffic to other hosts on the private network on which they're defined. Furthermore, hosts on a private network are hidden from the rest of the world, as external hosts have no direct way to send traffic to a host without a public IP. So, the question becomes this: if private network IP's are for internal traffic only, how can home networks communicate with the rest of the world? The answer is Network Address Translation (NAT). WIth NAT, your home router will "virtualize" the Source IP address on your outgoing packet with that of your Public IP (like the one assigned by your ISP), allowing remote hosts to respond to the request of an internal machine. In the context of our cluster, we will create a new private network that will allow us to further isolate cluster traffic from our home network, and use the head node to gain access to our compute nodes.

The translation that happens in your router on the way out (private to public) and on the way in (public to private) is the main feature of NAT. One public IP address belonging to your router can practically manage thousands of private IPs using the standard protocol.

With NAT in mind, we will need to set up our head node for NAT by enabling IP forwarding and configuring the firewall with iptables. In this instance, we are going to use eth0 for our external network, and eth2 (implemented using the USB dongle) to communicate with the internal network to allow fast communication between nodes. First, we will need to edit `/etc/sysctl.conf`, and remove the # from the line containing `net.ipv4.ip_forward = 1` (and make sure it equals 1 and not 0).

To enable this setting, we can then `run sudo sysctl -p /etc/sysctl.conf` to apply our changes. Alternatively, if you just want to test this setting without applying it permanently, you can run `sudo sysctl -w net.ipv4.`ip_forward=1. With IP Forwarding enabled, we now just need to install iptables with `sudo apt-get install iptables` and run three commands to setup NAT on the head node:

```
$ sudo iptables -t nat -A
POSTROUTING -o eth0 \
-j MASQUERADE

$ sudo iptables -A FORWARD
 -i eth0 -o eth2 -m state
 --state RELATED,ESTABLISHED
 -j ACCEPT

$ sudo iptables -A FORWARD
 -i eth2 -o eth0 -j ACCEPT
```

These commands may look a bit confusing, but fortunately we can describe what they're doing without much difficulty. In essence, we're enabling NAT and telling the head node to forward all traffic coming to eth2 (the internal network) through eth0, and to allow external traffic to pass to internal network if it's part of an established connection. As of now, these settings are temporary, and will be cleared upon reboot of the head node unless we explicitly save them and set them to load on boot.

Unfortunately, testing these settings without a DNSMasq service setup requires a bit more work, such as using a machine with a GUI and manually assigning an IP address and routing rules to test external communication. Instead, we're going to push forward bravely and configure these settings to save and load on reboot, then go about configuring DNSMasq so we can test everything at once.

To save the current firewall rules, we can run `sudo iptables-save > iptables.up.rules ; sudo mv iptables.up.rules /etc/.` Alternatively, if you're running as root, or have invoked sudo -i, you can simply run `sudo iptables-save > /etc/iptables.up.rules`. Finally, we'll need to create a startup script to load our iptables rules on boot by editing `/`

`etc/network/if-pre-up.d/iptables`, and writing the following lines to it:

```
#!/bin/bash
/sbin/iptables-restore < /
etc/iptables.up.rules
```

Finally, run `sudo chmod +x /etc/network/if-pre-up.d/iptables`, so the script can be executed on start.

Configuring DNSMasq

Our next question: What's DNSMasq, and why do we need it for our cluster? DNSMasq provides us with a lightweight DHCP server and a DNS server, both of which make our cluster more flexible for a variable number of nodes. DHCP is the Dynamic Host Configuration Protocol, and in essence it assigns IP addresses and networking information to hosts that request it.

On the other hand, DNS is the Domain Name System, and it allows us to refer to individual machines by a hostname, rather than by it's IP address. However, before we install and configure dnsmasq it's worth noting that misconfiguring your DNS server or DHCP server could cause all sorts of chaos on your home network. Fortunately, should this happen, unplugging the head node should rectify these issues should that happen.

To get started, we're going to run sudo apt-get install dnsmasq to install the server, and then stop it with `sudo /etc/init.d/dnsmasq stop`. To configure DNSmasq, use a text editor to modify `/etc/dnsmasq.conf` (as sudo). You may notice that there are a large number of commented out lines showing various configuration options for DNS and DHCP. In this case, we're going to specify a few options for our head node at the end of the file, instead of uncommenting the individual lines throughout the file that we need, as this should make it easier for us to make changes later on since all our settings will be in one place. For our cluster, we added the following lines:

```
interface=eth2
domain=ocluster
dhcp-range=eth2,192.168.128
.1,192.168.128.254,255.255.
255.0
dhcp-host=00:13:3b:99:92:b1,
192.168.128.254
```

So what do all these configuration lines mean? Interface specifies which interface the DHCP and DNS server should listen for requests on, while domain specifies the domain of our cluster.

DHCP-range specifies the range of IPs that can be allocated to hosts on the specified interface, in this case IP's between `192.168.128.1-192-.168.128.254` on our internal network, eth2. The final line is a static IP lease for our head node, where `00:13:3b:99:92:b1` is the MAC address of our head node on eth2, and `192.168.128.254` is the static IP. With these changes made, we need to configure odroid-server0 to use itself as it's primary DNS server so it can resolve the names of internal hosts.

To do this, we can modify `/etc/dhcp/dhclient.conf` and remove the # from the line containing `#prepend domain-name-servers 127.0.0.1`; and run `sudo dhclient` so the changes take effect. Finally, we found that dnsmasq would not work correctly for internal nodes when invoked by its startup script unless we modified `/etc/default/dnsmasq` and removed the # from the line, `#IGNORE_RESOLVECONF=yes`, although your mileage may vary.

Once this is all configured, we can run `sudo /etc/init.d/dnsmasq` restart to start the service. Alternatively, we could go ahead and just configure all of the nodes of cluster with static IP addressess and modify the `/etc/hosts` file on each node to include the IP addresses and hostnames of all the nodes on the cluster. While this approach can be simpler in some regards, it is much less suitable for a larger cluster, and thus will not be discussed here.

Connecting Compute Nodes

If you have your compute nodes configured correctly so there are no MAC address or hostname conflicts, then connecting the compute nodes is as simple as hooking them up and powering them on (assuming the ethernet devices are configured to use dhcp). From the head node you should be able to use ssh to connect to your compute nodes, e.g. `ssh odroid-server1`. Once connected, verify DNS and NAT are working by pinging your head node by hostname and an external website such as google.com.

On the off chance you run into configuration issues, there are a few steps you can take to determine whether DHCP/DNS or NAT is causing issues. You can use the tool nmap to scan and see if your compute nodes are being assigned an IP with `nmap -sP 192.168.128.0/24`.

If you can connect to your compute nodes but get a message about an unknown host when pinging other hosts by hostname, then there is likely a problem with DNS. If you have issues connecting to external hosts from your compute node try running `ping 8.8.8.8` to see if NAT is working correctly and allowing traffic through the head node.

A Simple MPI Example

Now that we've done all the set up, here's an example of parallel programming using MPI. You can run a "canned MPI application," ours is simple, with more complex programs coming in future installments. This

script, which we've called `simple1.sh`, just reports networking and host information from each of the compute nodes.

```
simple1.sh

#!/bin/bash
A=`echo -n "Hello from: "`
H1=`hostname `
H2=`hostname -i `
H3=`hostname -I `
H4=`hostname -f `
echo $PMI_RANK $A name=$H1
shortlist=$H2 longlist=$H3
FQHN=$H4
```

Type the following to run the script:

```
$ mpirun -np 4 -hosts
odroid-server1,odroid-
server2,odroid-
server3,odroid-server4 ~/
scripts/simple1.sh
```

Which gives the following output:

```
0 Hello from:
name=odroid-server1
shortlist=192.168.128.1
longlist=192.168.128.1
FQHN=odroid-server1.ocluster

1 Hello from:
name=odroid-server2
shortlist=192.168.128.2
longlist=192.168.128.2
FQHN=odroid-server2.ocluster

2 Hello from:
name=odroid-server3
shortlist=192.168.128.3
longlist=192.168.128.3
FQHN=odroid-server3.ocluster

3 Hello from:
name=odroid-server4
shortlist=192.168.128.4
longlist=192.168.128.4
FQHN=odroid-server4.ocluster
```

In this case, we used 4 ODROIDs systems to run our simple script, with 1 cores each. We used MPICH2 with an enumerated host list, and you can install MPI on Ubuntu with `sudo apt-get install mpich2` (installs MPICH2 and dependencies).

Next Steps

From here, we can now easily add more nodes to our cluster once they have a basic networking configuration in place. So where do we go from here? Thus far, we have just been using the ODROID user on each node, but this is not ideal if you want to have several users on your cluster.

Furthermore, copying files around can be a hassle with the current setup. Therefore, we will continue this series by detailing an AutoFS and LDAP server setup on the head node to allow us to share files and authenticate users across our cluster.

Additionally, we will look into running more complex MPI jobs using C/C++ programs we write as examples rather than the simple shell script we showed here to better demonstrate our cluster's capabilities. Real MPI programs transfer data with message passing and and harness parallel processing by working together, rather than our script, which ran independently on each ODROID.

Additional Reading

The MPICH Programming Language is a high performance and widely portable implementation of the Message Passing Interface (MPI) standard. MPICH and its derivatives form the most widely used implementations of MPI in the world. They are used exclusively on nine of the top 10 supercomputers (as of November 2013), including the world's fastest supercomputer: Tianhe-2. Learn more by visiting `http://www.mpich.org`, which offers many tutorials, publications, and other documents for developers.

FLAPPY BIRD
INSTALLING THE ORIGINAL GAME

by Ronaldo Andrade

Game enthusiasts all over the world were surprised when Vietnamese developer Dong Nguyen decided to remove his masterpiece from the Google Play and Apple stores. But this does not mean you can't get Flappy Bird anymore!

All you have to do to join the fun is to download the APK to your ODROID, install and play. It's a free application that doesn't require any additional licensing, and you no longer have to be jealous of everyone who downloaded it before it became scarce.

`http://apkandroid.blogspot.com.br/2014/02/flappy-bird-13-apk.html`

Enjoy, and don't break your screen!

Ronaldo has the bragging rights of a monk-like patience, and the highest score among the magazine's team

HOW TO KNOW WHEN YOUR CAT IS NAPPING

A GUIDE FOR ATTACHING SENSORS TO THE ODROID-XU

by Marian Mihailescu

Of course this article made us feel obliged to use as many cat memes as the art editor would deem appropriate.

One of the advantages of having an ODROID board as a computing platform is flexibility. It can be used as a computer, a research tool, a game console, or a media center. In this article we will explore a new way in which the ODROID can be used: for monitoring your home. More exactly, we will discuss how to attach a couple of sensors to the ODROID-XU which will allow us to detect motion and to monitor the room temperature.

Motion and temperature sensors are some of the easiest to connect, and have made the Arduino and the Raspberry Pi platforms very popular, as you can attach them easily to the GPIO (General Purpose Input/Output) pins exposed on the board. Of the Hardkernel products, the ODROID XU series is better suited to connecting sensors, as it includes a 30-pin expansion port that can be used for several types of connections, such as SPI (Serial Peripheral Interface) , I2C (Inter-Integrated Circuit) and GPIO. For the ODROID U series, there is an IO board accessory that can be connected to the USB port, providing similar capabilities.

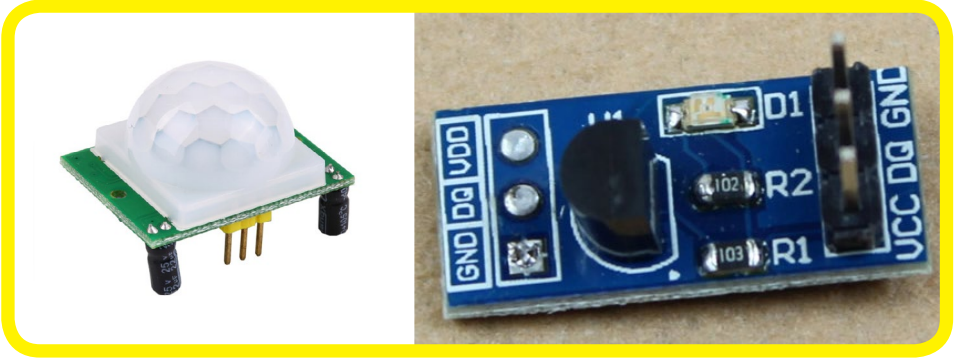
The sensors used for the purpose of the article, shown right, are quite common and inexpensive: the DS18B20 digital temperature sensor and the HC-SR501 Pyroelectric Infrared (PIR) motion sensor detector, each available for around US$2 on eBay.

Both sensors connect using 3 pins: Power, Ground, and Data. However, after inspecting the datasheets, we see two major problems. The sensors need 3.3V or 5V power, and output on the Data pin needs a similar voltage. The motion sensor outputs a logical value (0V = no motion; 3.5V/5V = motion), while the temperature sensor outputs the temperature using the 1-wire bus protocol. The first problem is that all the ODROID expansion headers are 1.8V (except the PWRON signal), so connecting the sensor's Data output directly to the board is dangerous. The second problem is that the 1-wire bus protocol is not enabled on the ODROID boards.

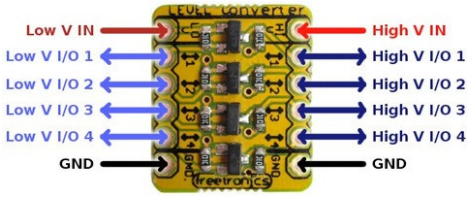
Connecting the sensors

In order to be able to connect the sensors, we need to adjust their output to the 1.8V voltage compatible with the ODROID boards. This is done using

Left: HC-SR501 PIR Motion Sensor
Right: DS18B20 Digital Temperature Sensor



a level converter, such as the Freetronics Logic Level Converter Module. Although it is not explicitly stated that it is compatible with 1.8V, the level converter works by having high power and low power references that are used to provide a bidirectional interface between different devices operating at these voltages. The connections for the logic level converter are shown in Figure 2.



Freetronics Logic Level Converter Module with 4 IO Ports

To provide the low voltage reference or 1.8V, we connect another GPIO pin of the ODROID board, which we configure to always output the logical value "1".

The entire connection schematic for the motion sensor is depicted in Figure 3. Pin 1 (5V0) of the ODROID board is connected (using the red wire) to the motion sensor power pin (VCC) and to the logic level converter High V IN input. Next, Pin 2 (GND) of the board is connected (using the black wire) to the motion sensor ground pin (GND) and the logic converter GND pins. We are using the GPIO pin 16 (GPX1.0) to provide the reference 1.8V power to the level converter Low V IN input (the green wire), using the following linux commands:

```
root@odroid:/ # echo 304 > /
sys/class/gpio/export
root@odroid:/ # echo out > /
sys/class/gpio/gpio304/di-
rection
root@odroid:/ # echo 1 > /
sys/class/gpio/gpio304/value
```

The first line is used to select GPIO pin 16 (GPX1.0). From the documentation provided by ODROID, 304 is the "base" address of GPX1 chip, to which we add the desired GPX pin address (in this case, 0). The second line is used to configure the pin for logical output, while the last line will set the output to logical 1, resulting 1.8V on pin 16. The motion sensor data output (OUT), which will be 5V when motion is detected, is connected (using the yellow wire) to the logic level converter High V I/O 1 pin, with the corresponding Low V I/O 1 pin (which is transformed to 1.8V) being connected to the board GPIO pin 13 (GPX1.5) using the blue wire. To pin uses the address 309 (304+5) and is enabled using the following command:

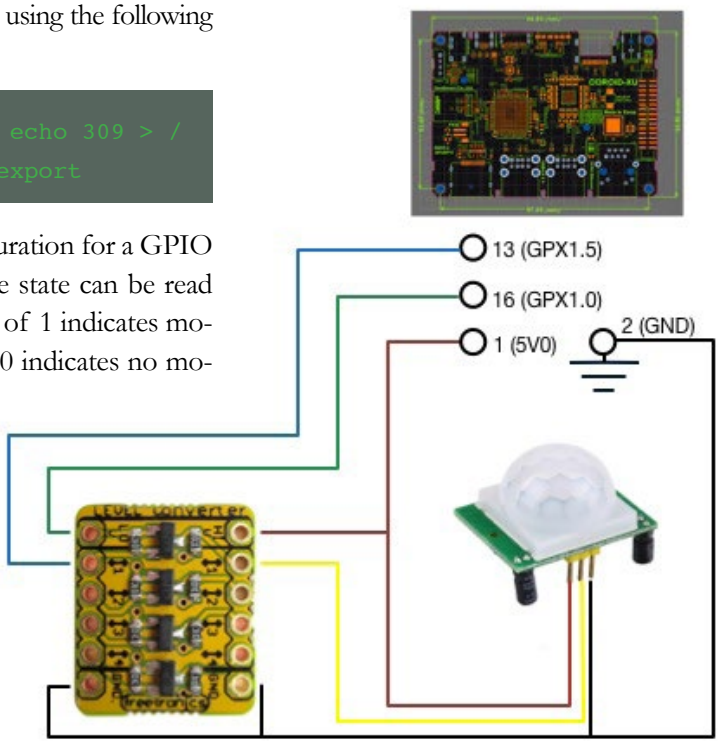
```
root@odroid:/ # echo 309 > /
sys/class/gpio/export
```

The default configuration for a GPIO pin is "input", and the state can be read from "value". A value of 1 indicates motion, while a value of 0 indicates no motion detected. The sensor also has two resistors which can configure the sensitivity (Sx) and the time which the output is set to "1" when motion is detected - from a few seconds to minutes (Tx).

Cats can sleep up to 16 hours a day, enough time for them to dream up new ways to be cute and cuddly and still hold down a full-time job

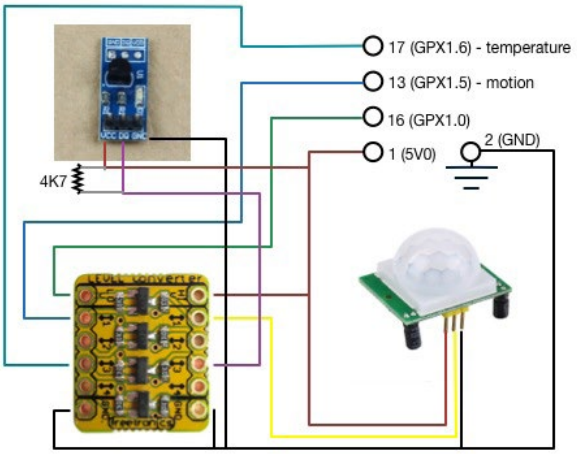
```
root@odroid:/ # cat /sys/
class/gpio/gpio309/value
1
```

Connect the Motion Sensor as show in this schematic, remember that this is for the ODROID-XU



Enabling the 1-wire protocol

The digital temperature sensor is connected in a similar way as the motion sensor (see Figure 4). The sensor power (VCC) is connected to the board's 5V source (pin 1), the sensor ground (GND) to the board's pin 2 (GND), and the sensor data (DQ) is connected to the level converter High V I/O 3 input (using the magenta wire). The level converter Low V I/O 3 is connected to the board's GPIO pin 17 (GPX1.6) using the cyan wire. The only difference is the presence of a 4.7KOhm pull-up resistor between the VCC and GND pins of the temperature sensor, used to keep the data transfer stable.



Connect the Temperature Sensor

The DS18B20 is using the 1-wire protocol for communication. The linux kernel has a driver for the 1-wire protocol (w1), and all that needs to be done on the ODROID devices is to specify what pin header we want to use for this protocol. For the ODROID XU, the modifications are done in the file `arch/arm/mach-exynos/board-ODROIDxu-ioboard.c` and include the following lines:

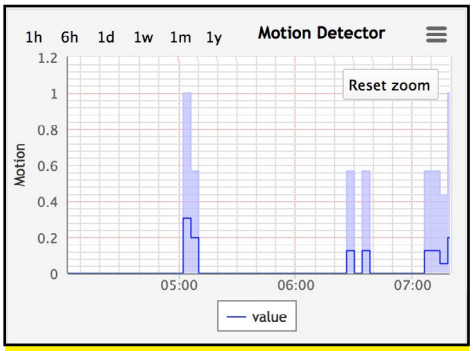
```
#if defined(CONFIG_W1_MAS-
TER_GPIO) || defined(CONFIG_
W1_MASTER_GPIO_MODULE)
static struct w1_gpio_plat-
form_data w1_gpio_pdata = {
.pin = EXYNOS5410_GPX1(6),
.is_open_drain= 0,
};

static struct platform_de-
vice ODROIDxu_w1_device = {
    .name  = "w1-gpio",
    .id    = -1,
    .dev.platform_data  =
&w1_gpio_pdata;
};
```

This will allow pin 17 (GPX1.6) to be used for the 1-wire protocol if it is enabled in the kernel configuration. After the new kernel is compiled, the sensor is activated and the temperature (which needs to be divided by 1000 to get Celsius degrees) is read with the following commands:

```
root@odroid:/ # mod-
probe w1-gpio
root@odroid:/ # mod-
probe w1-therm
root@odroid:/ # cd /
sys/bus/w1/devices;
ls
lrwxrwxrwx 1 root root 0
Feb  1 12:24 28-000004bc791d
-> ../../../devices/w1_bus_
master1/28-000004bc791d
lrwxrwxrwx 1 root root 0 Feb
1 12:24 w1_bus_master1 ->
../../../devices/w1_bus_mas-
ter1
root@odroid:/ # cat 28-
000004bc791d/w1_slave
28625
```

This article provides a basis for connecting various sensors to your ODROID board. There is a small step from reading and visualising the sensor values, to making active decisions based on them (e.g. sending an email when motion is detected) and then home automation.



The ODROID-XU sensors capture my cat roaming around the house at 5AM and 6:30AM

If you'd like to get more detailed information about the XU sensors, you can use the magic of Google Search, or visit some of the recommended links listed below:

DS18B20 datasheet: http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
HC-SR501 datasheet: http://www.mpja.com/download/31227sc.pdf
ODROID XU connectors: http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu#expansion_connectors
kernel patch to enable w1 protocol: https://github.com/hardkernel/linux/commit/6ffdec4496b7fcb2504423ab3827993ff341696d

That was definitely fun to do, I hope sometime soon we do something with another pet, parrots maybe...

UBUNTU 14.04 TRUSTY TAHR

NOW AVAILABLE FOR THE ODROID PLATFORM!

by Rob Roy, Chief Editor

What does the Tahr say? "Dude, if you have early adopter blood, go and upgrade!"

Although Ubuntu 14.04 is not set for official release until April 17th, 2014, ODROID owners can perform an early upgrade to Trusty Tahr using the "update-manager" application. By synchronizing your OS version with the latest Canonical release, you can keep your favorite Ubuntu image running indefinitely, without needing to reinstall the entire operating system whenever a new version is announced.

ODROID computers that are already running an Ubuntu 13.04 or 13.10 image only need to enter a few commands in order to upgrade the Ubuntu operating system to the latest 14.04 version. To begin, close all open applications, launch a Terminal window, and type:

```
$ sudo apt-get update \
&& sudo apt-get upgrade \
&& sudo apt-get dist-upgrade\
&& sudo apt-get autoremove \
&& sudo apt-get clean
```

The `update-manager` command applies the latest updates to the current operating system, in preparation for the upgrade. Confirm all questions until the bash prompt re-appears, then type:

```
$ sudo update-manager –d
```

Press the "Upgrade" button, click "Continue" a few times, and go make a sandwich! After determining which packages will be required for the upgrade, your system will ask for a reboot. Before doing so, it's important to replace the default video configuration file, so that the ODROID video settings in `/etc/X11/xorg.conf` take effect and produce an HDMI signal.

If you've already rebooted, don't worry, you can perform the patch via SSH. Otherwise, open a second Terminal window and enter the next two commands to restore the ODROID video drivers:

```
$ cd /etc/X11/xorg.conf.d
$ sudo mv exynos.conf \
 exynos.conf.original
```

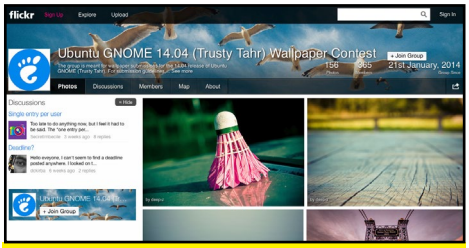
If you're running Lubuntu Whisper, you'll also need to move the XFCE desktop service into the LXDE configuration file to retain desktop compatibility with 14.04:

```
$ cd ~/.config/autostart
$ rm xfdesktop.desktop
$ cd ~/.config/lxsession/LXDE
$ echo "@xfdesktop \
 --replace" >> autostart
$ sync && sudo reboot
```

After the ODROID has rebooted, verify that your new operating system is installed by typing the following command in a Terminal window:

```
$ lsb_release -a
```

It's that easy! Don't forget to get your new Trusty Tahr 14.04 desktop wallpaper from Flickr at `http://www.flickr.com/groups/2484760@N20/`. Please note that some earlier versions of Ubuntu may also require the additional step of recompiling the Mali drivers for use with Xorg Server 1.14, which is detailed in the May issue under the title "Recompiling Mali".



Enrich your desktop experience, there is as many wallpapers as you would ever need.

LEARN REBOL

WRITING MORE USEFUL PROGRAMS WITH AMAZINGLY SMALL AND EASY-TO-UNDERSTAND CODE

By Nick Antonaccio and Bohdan Lechnowsky

Rebol (Relative Expression Based Object Language) is a revolutionary advancement in programming language emerging from over thirty years of language research. It offers enormous flexibility and power, with a focus on intuitive

the first installment of Learn Rebol last month, we discussed the motivation behind Rebol and learned how easy it is to create a GUI-based program in Rebol on Android.

In this installment, we'll briefly revisit how to install Rebol on Android, but we'll also show how to install it for ODROID's Ubuntu environment. In addition, we'll show how to create a text editor with which you can write Rebol programs (or anything else for that matter), calculators, bar charts, data input grids, and even a Web photo viewer – all from scratch! Not only will these programs run on your Android or Ubuntu installation, but they will also run on your Windows, Linux and MacOS X laptop or desktop without any modifications!

And yes, you can run any app you create in Rebol 3 on your Android-powered phone or tablet as well!

Installation

Android:

Open a web browser and navigate to `http://development.saphirion.com/experimental/builds/android/`

Download `r3-droid.apk` (amazingly smaller than 2MB).

When finished, double-click on the download icon (usually by the clock) and grant permissions to install.

Go to the apps list and click the icon for R3/Droid.

Ubuntu:

Open a web browser and download `http://atronixengineering.com/r3/downloads/r3-arm-view-linux-2014-02-19-715e14`

Perform the following commands in the terminal emulator in the directory where you downloaded r3 (as sudo):

```
mv r3-arm-view-linux-2014-02-19-715e14 r3
chmod +x r3
./r3
```

Note: Not only does Rebol 3 work great on ODROID devices, but Rebol 3 for Linux-ARM is being developed on ODROID devices.

Pretty simple, eh?

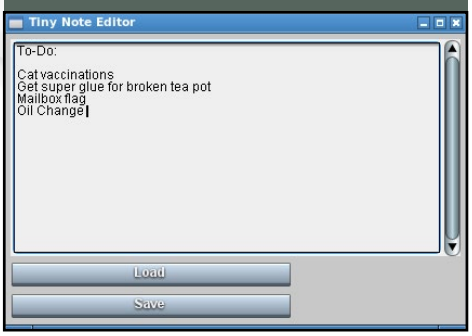
Getting to work

As mentioned in the last installment, Rebol has multiple GUI toolkits. In Rebol 3, the most popular is called R3-GUI. The GUI toolkit of choice needs to be loaded if the program is going to make use of it. This can be done dynamically from the web (for always automatically using the latest version), or a copy of the GUI toolkit can be stored on your computer (for quicker access).

R3-GUI can be loaded from the web by simply using the command: load-gui

Here's the code for a little note pad app that allows you to create, load, edit, and save a text file. You could use it to store an editable to-do list, shopping list, notes and reminders, or other free form data. The first two lines are stock code that you'll see at the beginning of every example in this section. There's a text area widget and two buttons which load and save the notes.txt file. It's pretty easy to follow the code, even without any formal introduction to Rebol:

```
REBOL [title: "Tiny Note Editor" file: %tiny-note-editor.r]
load-gui
view [
    a: area
    button "Load" on-action [attempt [set-face a read/string %notes.txt]]
    button "Save" on-action [write %notes.txt get-face a alert "Saved"]
]
```



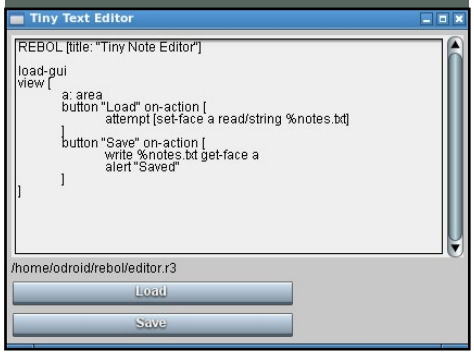
If you don't feel like typing in the code, you can simply enter the following to load it from the web where I have saved a copy of the script:

```
do http://respectech.com/
odroid/learnrebol/tiny-note-
editor.r
```

Likewise, the following examples can be accessed the same way, but just change "tiny-note-editor.r" to the filename specified in the REBOL header on each script.

Now, we'll modify it so we can specify the file to open and the filename to save:

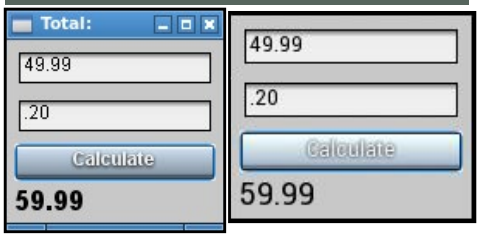
```
REBOL [title: "Tiny Text
Editor" file: %tiny-text-
editor.r]
load-gui
view [
    a: area
    filename: text file 200
    button "Load" on-action
[
        set-face filename
request-file
        set-face a read/
string to-file get-face file-
name
    ]
    button "Save" on-action
[
        write file: request-
file/file to-file get-face file-
name get-face a
        set-face filename
form probe file
        alert "Saved"
    ]
]
```



As you can see, Rebol's code structure is pretty free-form. The Tiny Text Editor is laid out differently than the Tiny Note Editor, but mostly because the buttons' on-action blocks are split into lines based on the actions being specified.

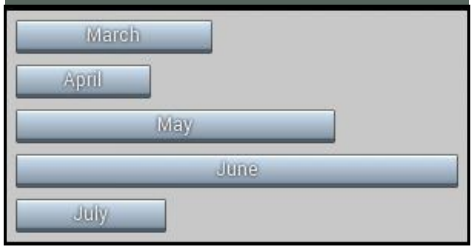
Here's a short program to calculate restaurant tips. Like every other app here, it can be run instantly on any Android phone, tablet, or device, or on any desktop/laptop PC, using the exact same code:

```
REBOL [title: "Tip Calcula-
tor" file: %tip-calculator.r]
load-gui
view [
    f: field "49.99"
    t: field ".20"
    button "Calculate" on-
action [
        set-face x round/to
(to-decimal get-face f) * (1
+ (to-decimal get-face t))
.01
    ]
    x: title "Total: "
]
```



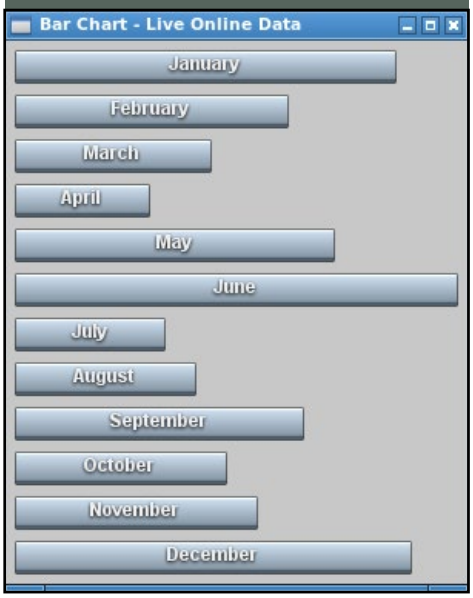
Here's an example that displays a block of data in graphic bar chart format. It consists of 2 stock header lines, 1 line of data to display, and 1 short line of actual code. Like every other R3 example in this text, all you need to create and run this application on any platform is a text editor and the tiny R3 interpreter. This app, along with the R3 interpreters for every popular OS platform, could be quickly emailed to friends or co-workers, and opened instantly on any device each user happens to have available:

```
REBOL [title: "Bar Chart"
file: %bar-chart.r]
load-gui
d: ["March" 13 "April" 9
"May" 21 "June" 29 "July"
10]
g: [] foreach [m v] d [ap-
pend g reduce ['button m v *
10]] view g
```



Network access and web protocols are built into R3 natively, to provide easy access to all types of online data. Here's a variation of the above program, which displays a chart of live data read directly from a web URL (`http://learnrebol.com/chartdata`). Change the data at the web URL, run the app, and the bar chart displays the appropriate graphic adjustments:

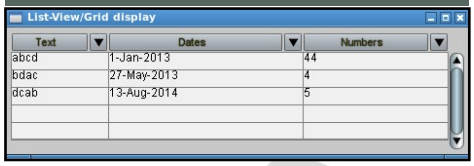
```
REBOL [title: "Bar Chart -
Live Online Data" file: %bar-
chart-live-data.r]
load-gui
d: load http://learnrebol.
com/chartdata
g: [] foreach [m v] d [ap-
pend g reduce ['button m v *
10]] view g
```



Here's a grid display, typical of any app that involves managing tables of text, numbers, or other data. Rows can be added or removed, cells can be edited manually by the user, and the values are sortable and filterable by clicking column headers (months are automatically arranged chronologically, text alphabetically, numbers ordinally, etc.). Most of this example is just the data to be displayed. You don't really even need to understand anything about programming to follow this code:

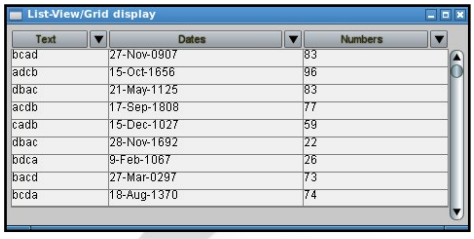
```
REBOL [title: "List-View/
Grid display" file: %grid-
display.r]
load-gui
```

```
view [
    text-table [“Text” 100
“Dates” 200 “Numbers”] [
        [“abcd”  1-jan-2013
44]
        [“bdac”  27-may-2013
4 ]
        [“dcab”  13-aug-2014
5 ]
    ]
]
```



R3 can be used to create full featured web applications. Here's a variation of the above program, which reads data created by an R3 web app running at `http://learnrebol.com/grid-data.cgi`. Run it several times to see the updated data generated each time by the web app:

```
REBOL [title: “List-View/
Grid display” file: %grid-
display-live-data.r]
load-gui
webdata: load to-string read
http://learnrebol.com/grid-
data.cgi
view [text-table [“Text” 100
“Dates” 200 “Numbers”] (web-
data)]
```

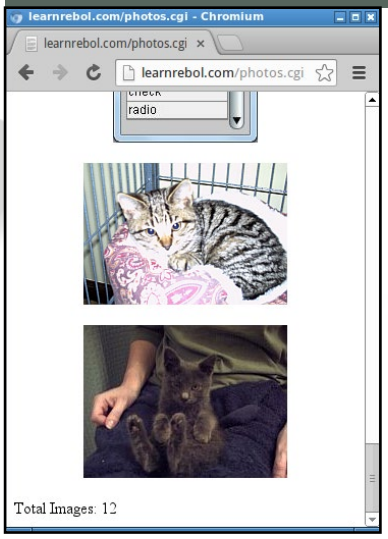


Here's the code for the app running on the web server that creates the random data displayed in the GUI grid above:

```
#!./rebol3 -cs
REBOL [ ]
random/seed now/time
print {content-type: text/
html^/}
data: copy {}
loop 100 [
    append data rejoin [
        “[“ mold random
“abcd” “ ” random now/date “
” random 100 “]”
    ]
]
print data
```

Here's another simple web app. It centers and displays all photos found in a given folder on a web server, along with the total count of all displayed images. A demo of this script is available at `http://learnrebol.com/photos.cgi`. Web apps like this can run on any computing device that has an Internet connection, even if R3 isn't installed on the device. All you need is a web browser (the code runs on the web server, and pushes out results for the browser to see):

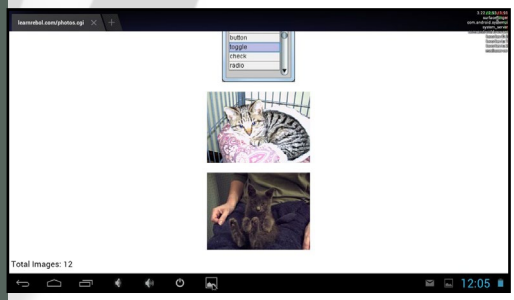
```
#!./rebol3 -cs
REBOL [title: “Web Photo
Viewer”]
print {content-type: text/
html^/}
folder: read %./
count: 0
foreach file folder [
    foreach ext [“.jpg”
“.gif” “.png” “.bmp”] [
        if find file ext [
            print rejoin
[{<BR><CENTER><img src=”}
file {”></CENTER>}]
            count: count + 1
        ]
    ]
]
print join {<BR>Total
Images: } count
```

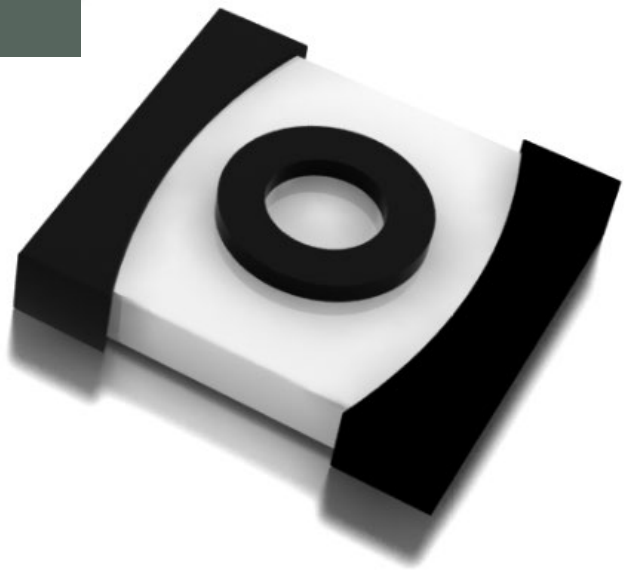


Conclusion

As you might conclude from the preceding examples, creating powerful and useful apps in Rebol 3 is about as simple as it can get. But, it can get even easier! In a future article in this series, we'll discuss using Rebol 3 to create DSLs (Domain-specific Languages) to make programming applications as simple as you can imagine.

If you want to get in on the discussion with the Rebol/Red community, there are two main forums where you can interact with other programmers and developers in real-time: The [Rebol and Red] chat room on StackOverflow.com, and the Rebol-powered darknet “AltME Rebol4 World”. To join the AltME world, send an email to user bo at the domain respectech.com and/or henrikmk at the domain gmail.com asking to be invited. We are a closed community to avoid spam. Don't be shy, the Rebol/Red community is known as the friendliest software development community on the planet!

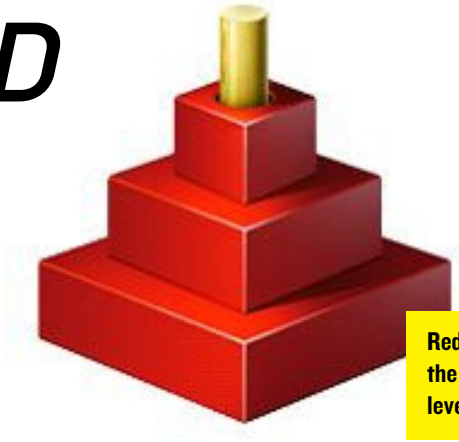




LEARN RED

THE NEXT EVOLUTION OF REBOL: PART I

By Gregory Pecheret



Red's logo depicts a full stack language able to cover the full development spectrum from low-level to high-level programming. Red is a full-stack language!

Question: Which programming language has ALL of the following advantages?

cross-platform
cross-compiling
open source
embeddable
Unicode-compliant
Lua/Scala/Rebol-inspired
paradigm neutral (functional, imperative, symbolic, prototype-based objects)
full stack (machine level to Meta Domain-Specific Language, and everything in-between)
simple/compact/fast/ubiquitous/portable/flexible/green

The answer is Red!

Red is a modern programming language that re-uses most of Rebol's syntax and semantics. While Rebol is an interpreted language, Red can run either as an interpreted language, or be statically compiled to native code. The Red interpreter and compiler is written in Rebol 2, so at most a Rebol 2 interpreter is required to compile a Red program through Red. Red also has a standalone executable that can be used to interpret or compile programs. Beginning with version 2, the Red engine will be ported from Rebol 2 to Red. This will make Red a self-hosted language, with two stacks called Red and Red/System.

Red/System is the low-level component of the Red programming language and provides Red's runtime library, a linker to produce executables and a low-level system programming language similar to a C-level language suitable for device driver development, native library usage, and more. Red is a flexible mid to high level scripting language similar to Rebol, suitable for complete applications, user interfaces, data modeling, domain-specific language creation and scripting.

Red's runtime uses a hybrid approach by compiling what can be solved statically, using a just-in-time compiler for other cases, and finally interpreting when none of those approaches satisfies.

Getting Started

We will first use the Red executable (available since version 0.4.0) to compile the Red console (available since version 0.3.2), and then we'll use this console to learn some Red language basics.

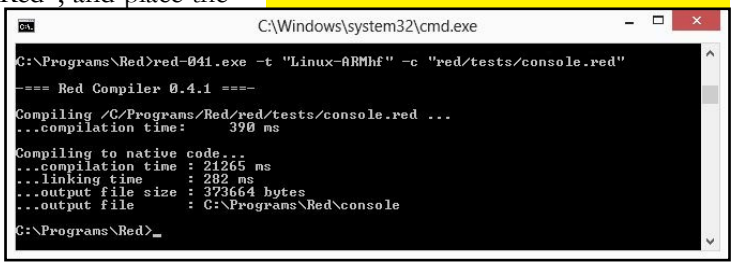
Red is built on the earlier Rebol 2, which was never ported to Linux ARM (like Ubuntu on ODROID). The current version of Rebol 3 has been ported to Linux ARM, but is not compatible with Red at this time. Therefore, we will need to cross-compile our Linux ARM programs on a Linux, Windows or Mac OS X desktop machine until Red becomes self-hosted.

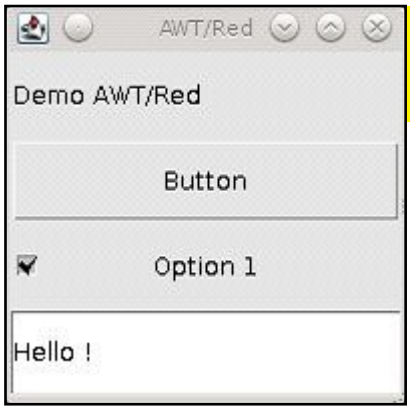
From the download section of `http://red-lang.org`, download the Red executable for your favorite desktop platform. Create a directory on your system called "Red", and place the Red executable in that directory. This will be your Red root directory. Run it straight from the file since no installation is required, except chmod +x red-041 on systems that require this. The Red executable is both a Red compiler and a Red interpreter, all rolled into one file. If you run it without any command line options, it will launch as a Red interpreter (Read Eval Print Loop). The first time it is launched, it will compile the interpreter automatically. Since Red compiles its own REPL, we'll need to download the Red source code from `github.com/red/red` to perform this compilation. If you're not familiar with GitHub, go to `https://github.com/red/red/tags` and click the .zip (Windows) or .tar.gz (Linux) file under the most recent version (v0.4.1 as of this writing). Now, let's compile the REPL console for ODROID:

The console executable for Linux-ARMhf ("Linux-ARMhf" for hard-float vs "Linux-ARMsf" for soft-float operating systems) is now available in the red-master folder. Ubuntu on ODROID is a hard-float operating system.

The following example illustrates both capabilities by calling a Red library embedded in Java to build a basic user interface through the Java AWT. This is based on the code available in your Red root directory at `red/bridges/java/hello.red`.

Compiling Red Console on MS Windows





Since version 0.3.3, it is possible to both compile a Red program as a native library and to embed a Red library in Java thru JNI.

Here you have to build hello.red as a library and bridge.java with a JDK.

To build hello.red, use the Rebol 2 executable you downloaded earlier on one of the supported platforms and run this Rebol command from the Rebol console:

```
cd %path/to/your/red/root/
directory
do/args %red.r "-dlib -o
%red/bridges/java/hello
%red/bridges/java/hello.red"
```

Then compile bridge.java using Java's compiler (which you must already have installed, and which is beyond the scope of this tutorial):

```
javac bridge.java
```

And run it with Java (again, which must already be installed):

```
Java –Djava.library.path=.
bridge
```

To avoid potential issues, make sure your Java runtime and compiler matches (javac –version, java –version).

Since Red v0.4.1, Red features a parsing engine, a well-known method in Rebol to write dialects (sometimes referred to as DSLs – Domain-Specific Languages). A typical case of parse usage would be rewrite the code shown next column in a more elegant way to wrap the AWT calls. This is part of the Red roadmap.

Using a dialect, then it could be compressed to something simple with the flexibility of the following AWT examples:

```
main: function [][
    frame: java-new [java.
awt.Frame "AWT/Red"]
    layout: java-new [
        java.awt.GridLayout
4 1
    ]
    button1: java-new [
        java.awt.Button
"Button"
    ]
    label: java-new [
        java.awt.Label
        "Demo AWT/Red"
        label.CENTER
    ]
    checkbox1: java-new [
        java.awt.Checkbox
"Option 1"
    ]
    textfield: java-new [
        java.awt.TextField
"Hello !"
    ]
    java-do [frame/setLayout
layout]
    java-do [frame/add la-
bel]
    java-do [frame/add but-
ton1]
    java-do [frame/add
checkbox1]
    java-do [frame/add tex-
tfield]
    java-do [frame/setSize
200 200]
    java-do [frame/setVis-
ible yes]

    events: java-new
[events]
    java-do [frame/addWin-
dowListener events]
]
```

```
view [
    set "AWT/Red" 4 1
200x200
    button1: button []
    label1: label "Demo AWT/
Red" center
    checkbox1: check "Option
1"
    textfield: field "Hello !"
]
```

Just imagine how much more productive software developers could be by using the power of DSLs like this. The view dialect example above uses less than ¼ the amount of code than the Java example it replaces, and is much more readable and easy to debug and change.

Red's Past and Future

The author of Red, Nenad Rakocevic, first announced his intent to build the Red language on February 26th, 2011 at the Rebol/Boron conference in the Netherlands. He has since been working full-time on his endeavor. Red is still a work in progress, but Red foundations have been set and it is already possible to cross-compile and/or embed Red in Java, for example.

Android is clearly a focus, and Red is able to produce APK binaries as the proof of concept "eval" progam demonstrates (download it from `http://static.red-lang.org/eval.apk`). Red is becoming an alternative to allow development of Android applications that remain independent of complicated IDEs and development environments and is extremely lightweight. Upcoming releases will include simple I/O support of files, full object support, Visual Interface Dialect for GUI creation, and more.

To view Nenad's recent talk on "What is Red" at the ReCode Conference 2013, visit `http://www.youtube.com/watch?v=H4kMlOkN894`. *The talk was recorded with a camera programmed and developed using Red and Rebol 3.*

WEB DEVELOPMENT WITH CODE MONKEY AND QUIET GIANT:

USING ODROIDS TO BUILD A SUCCESSFUL BUSINESS

by Rob Roy, Chief Editor

The Code Monkey desktop's default wallpaper reminds you that you can do more with less.

ODROID computers are extremely versatile, and can be used for nearly every type of computing application, including gaming, robotics, desktop publishing, web browsing, audio production, media server and playback, and much more. Many ODROID owners use them exclusively for fun, learning and home entertainment. However, because of their powerful quad-core processors, generous 2GB of RAM, and low cost, you can also run a successful web development business with an ARM-based ODROID cluster from your home!

The high cost of a typical software development computer can be prohibitive for home enthusiasts who wish to offer professional web development services, especially considering the cost of installing a typical Windows or OSX operating system, purchasing expensive laptops, hard drives, memory, processors and cooling systems, licensing a development studio package such as Visual Studio, and renting a dedicated development server from a data center.

In contrast, the startup cost of building an ODROID-based development studio is approximately 90% less expensive than a home-built x86 machine, while leveraging the latest free, open-source operating systems and community-supported software to provide a robust platform for web developers.

I have been running a successful web development business since 2012 with my ODROID-X2 and ODROID-U2, using them exclusively to produce and maintain modern HTML5 websites that are viewed by hundreds of thousands of visitors each month. Almost any web software that can be written using an expensive Windows- or OSX-based system can also be developed using an ODROID, including HTML5, responsive layouts, content management, cross-browser compatible code, and web applications written in popular modern languages such as jQuery, AngularJS, PHP, Java, and JavaScript.

For those aspiring to augment their income with web development, the community images Code Monkey and Quiet Giant provide a virtual 8-core system that operates in parallel to provide a sandbox web environment. A sandbox environment is where a website is prototyped on a local network server before being published to an expensive public internet server. Code Monkey includes several great Interactive Development Environments (IDEs) including Bluefish Web Editor, which will be used in this article to illustrate how to get a basic Wordpress site running in an inexpensive sandbox environment. Both images used in this article are available for free download from the ODROID forums at `http://forum.odroid.com`.

Choosing the Development Environment

The development stack used on many internet servers is known as LAMP, which stands for Linux, Apache, MySql, and PHP. Code Monkey runs a version of Linux called Ubuntu, and Quiet Giant runs a server-specific version of Ubuntu, published by Linaro, which is optimized for high traffic server usage. Nginx is a popular alternative to Apache, and other Linux distribu-

tions such as Debian are also viable for use as the server OS. MySql is the standard database package for storing persistent user data, layout and other web information relevant to the appearance and data storage of the site.

Quiet Giant comes with MySql and Apache pre-installed, along with several other services that can be used to mirror a production environment, such as DNS, Tomcat and Mail. It is not recommended to expose the sandbox website to the internet for security reasons, so it is very important to use a router with a firewall in order to protect the local system from intrusion, injection or spying. As a general rule, never keep sensitive data on a public-facing internet server unless proper security and firewall software is in place to prevent hacking.

The advantages to LAMP development are the wide availability of quality low- and no-cost software, a stable OS environment that requires minimal maintenance, and an ability to host the website on any machine, regardless of the server's performance profile. For the purposes of demonstrating sandbox development, the Quiet Giant image will run the same software that would be installed on a high-end internet server (Apache, PHP and MySql) so that, when it comes time to push the site to the data center, there will be no compatibility issues, since the code was developed using the same tools and software packages that are running in the production environment.

Gathering the Equipment

To illustrate the concepts of ODROID web development, I chose Wordpress 3.8 as an example Content Management System (CMS) platform, which will be installed onto the Quiet Giant server, and then customized using Bluefish Web Editor from the second ODROID running Code Monkey.

Any two ODROIDs from the X, U or XU series can be used, with the XU being the best option for the Quiet Giant server due to its USB 3.0 ports and high-performance A15 cores, and the X or U machine serving as the ideal development computer due to their low cost and portability. If an XU is not available, another X or U computer can be substituted for the server, since Quiet Giant has already been ported to every ODROID hardware platform.

Setting up the Network

Three protocols are used to share resources between the two machines, creating a robust development environment paired with powerful server capabilities:

Samba file-sharing protocol enables a remote server's shared directory to be mounted as if it were a local hard drive. Quiet Giant includes a pre-configured Samba server that automatically permits sharing of the Apache web directory (www), making it visible as a mountable drive to any client machine located on the local network.

SSH allows remote commands to be sent to the Quiet Giant server from the Code Monkey development ODROID over an encrypted, password-protected connection. It mimics the use of the Terminal window on the server by creating a remote BASH shell that can be used to start and stop services. In this example, SSH will be used to configure the MySql server before Wordpress is installed.

HTTP is the standard web protocol that runs on port 80, and allows Apache to listen for incoming web traffic. Whenever the web server's URL is typed into a browser on the client machine, a request for port 80 is sent to that server, which then notifies the web software, such as Apache, that a visitor is requesting a copy of the site.

Before installing WordPress, a static IP address for the server needs to be established so that the client and server machines can find each other. To establish the static IP, boot up the Quiet Giant image, plug in the ethernet cable, and log into the router admin panel from any other computer on the network. The ODROID running Quiet Giant can be identified in the router's client list by its host name of "odroid-server".

After configuring the DHCP reservation, it may be necessary to reboot the Quiet Giant server in order for the new address to be assigned. For more details on creating a static IP, please refer the specific router's instruction manual.

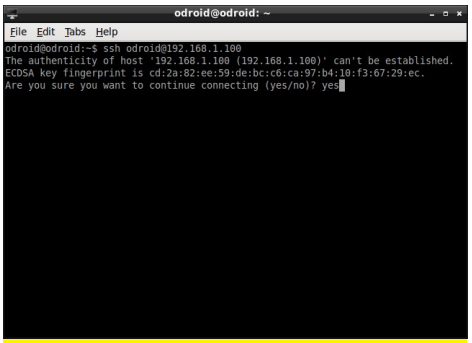
Although it may be tempting to use a wireless dongle as part of the server's hardware configuration, a wired ethernet connection will give the best performance, and reduce the amount of waiting required when updating files, due to the higher throughput of the LAN connection which is up to 30% faster than wireless.

Configuring the Database Server

For the Wordpress installation to function properly, MySql needs to reserve space in the database for Wordpress content and configuration files. To configure the MySql installation, log into the Quiet Giant server using the SSH protocol by booting up Code Monkey on the client ODROID and launching a Terminal session. The default SSH username and password of "odroid", and a server address of 192.168.1.100, will be used in the following examples.

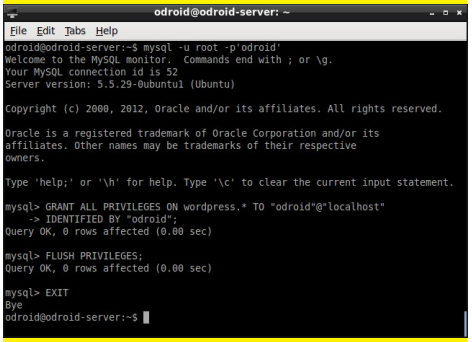
```
ssh odroid@192.168.1.100
```

Start the MySql admin panel after the command prompt appears with the following command. The default MySql



Establishing an SSH connection to our server

Checking the MySql root user privileges



Configuring the MySql server for use by Wordpress

username is "root" and the password is "odroid":

```
mysql -u root -p'odroid'
```

Once the MySql admin panel is launched, the Wordpress database can be configured:

```
mysql> create database word-
press;
mysql> grant all privi-
leges on wordpress.* to
"odroid"@"localhost" identi-
fied by "odroid";
mysql> flush privileges;
mysql> exit
```

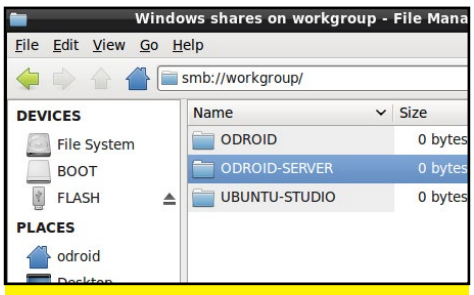
Now that the MySql database is ready to accept the Wordpress content, start the File Manager program by double-clicking its shortcut on the Code Monkey desktop. The rest of the Wordpress configuration and installation will be performed using the development machine, and the SSH session to the Quiet Giant server is no longer needed.

Copying the Client's Website

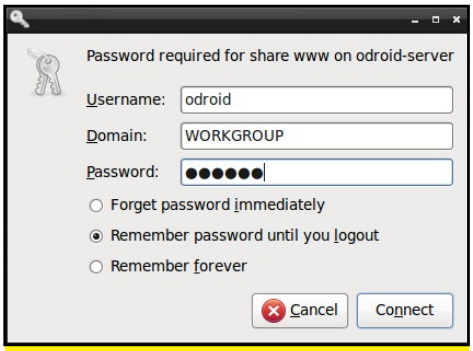
Wordpress, one of the world's most popular and well-supported Content Management Systems (CMS), is open-source and available for free at `https://wordpress.org/download/`. It lets developers build professional websites quickly, while also providing an interface for non-technical people to update website content. Wordpress is very simple to install, use, and customize, while also offering hundreds of useful add-ons that easily perform complex tasks such ad rotation, social media integration, image slideshows, video carousels, responsive layouts and much more.

Download the Wordpress .tar.gz package and save it to the `~/Downloads/` directory on the Code Monkey client machine. Decompress the package by right-clicking it in the Thunar File Manager and selecting "Extract Here".

To copy Wordpress to the Quiet Giant server, first connect to the server's shared Samba "www" directory by selecting the "Browse Network" option on the left side of the Thunar window. Navigate to the "Windows Network" -> "WORKGROUP" -> "ODROID-SERVER" directory, and click on the "www" share. The Samba share may also be accessed directly by typing "`smb://odroid-server/www`" into the Thunar address bar. Use the default username and password of "odroid" to complete the connection.



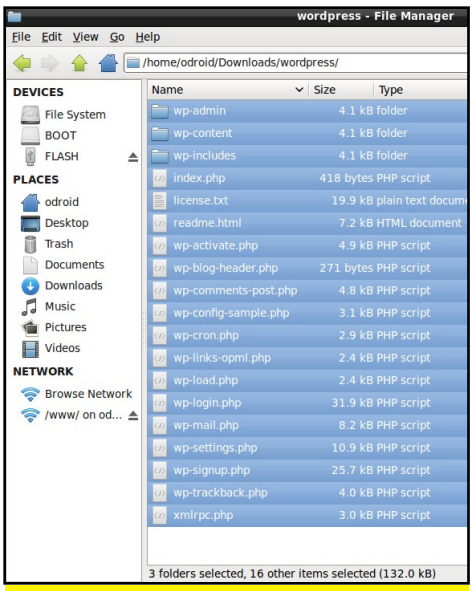
Locating the Samba share



Validating the Samba credentials

The Samba file-sharing protocol facilitates the installation of Wordpress files on the server by mounting the Apache "www" directory as if it were a local hard drive on the client ODROID. After the website is installed, the same Samba share allows customization of the layout, styles and other Wordpress code via Bluefish Web Editor.

Using Thunar, copy the files from inside the newly extracted wordpress directory, and paste them into the www shared directory (`smb://odroid-server/www/`).

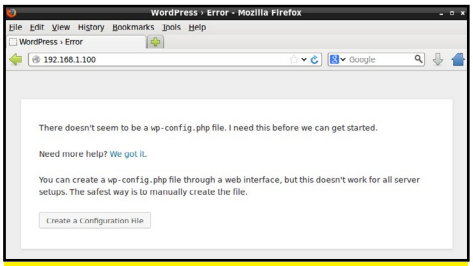


Copying the Wordpress files to the server

Configuring the Wordpress Site

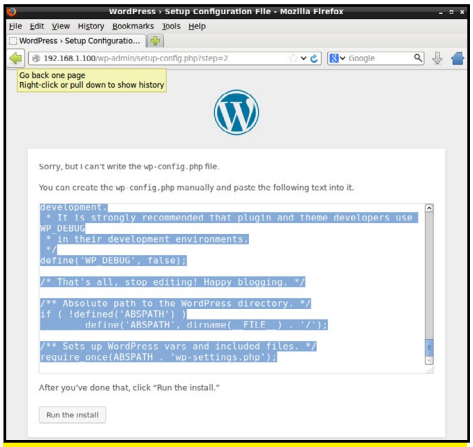
Once the files have been copied, launch any web browser from the Code Monkey desktop, and type the static IP address of the Quiet Giant machine as the URL (for example, `http://192.168.1.100/`). The Wordpress welcome screen will ap-

pear, with a notification that one of the configuration file needs to be created. So far, so good!

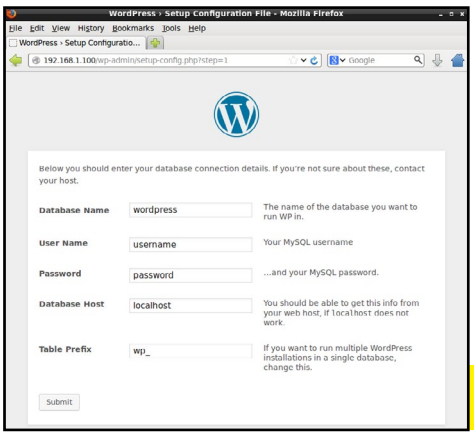


Initial setup of Wordpress

To create the necessary wp-config.php configuration file, open the pre-built example from the Wordpress web page by clicking on the "We Got It" link, and copying the file contents using Ctrl-C. Launch the Gedit Text Editor from the Code Monkey desktop, and paste the file into a new document using Ctrl-V. Save the file as "wp-config.php" by pressing Ctrl-S, clicking on the "www on odroid-server" link in the left side of the Save File dialog, and pressing "Save".

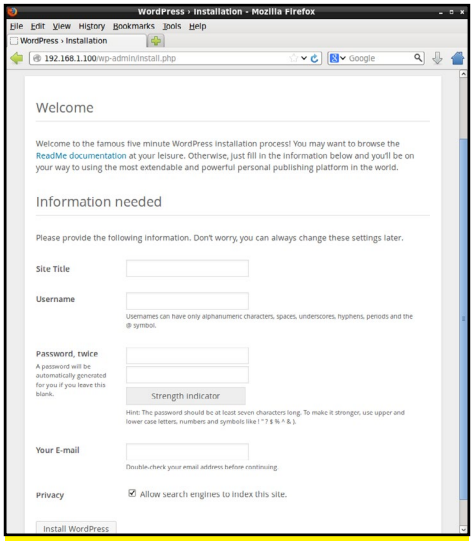


Copying the example wp-config.php Wordpress configuration file



Database setup of Wordpress

Switch back to the browser and click on "Run the Install" to access the Database Setup screen. Replace both the username and password with "odroid", and click "Submit".



Final setup screen of Wordpress

Customizing the Wordpress Site

At this point, the Wordpress sandbox development environment is ready for use. Many administrative and configuration tasks, such as adding themes and plugins, can be performed via the wp-admin interface located at `http://192.168.1.100/wp-admin`.

Advanced developers may want to customize the site PHP files, located in `smb://odroid-server/www/`, by using the Bluefish Web Editor application to open the shared Samba directory. Extensive tutorials on customizing Wordpress, downloading themes and plugins, and modifying the source code are available at the official Wordpress site `https://codex.wordpress.org/Getting_Started_with_WordPress`.

The Sandbox Advantage

The ODROID web development multi-boxing setup solves the productivity issue of constantly using FTP to publish incremental changes to a remote web server. Making local changes, pushing them to the internet server, then constantly hitting Refresh in the browser, wastes valuable time that is better spent customizing the site.

By setting up a multi-boxed sandbox environment, the Quiet Giant machine resides behind the local network, and all development can be done directly on its hard drive using the Samba file sharing protocol to edit the files from the Code Monkey development machine. Since Apache and MySql are used in the ODROID sandbox environment, compatibility with any production environment that offers LAMP is nearly guaranteed. File are copied to the server using the File Manager instead of FTP, resulting in enormous time savings because of instantaneous file access.

Additionally, the low cost of the all-ODROID development studio means that a beginning website business can make money on its first project, without having to first recover high startup equipment costs. ODROIDs also make it very easy to test websites on both Desktop and Mobile without the need to purchase additional QA tablets, since the ODROID is capable of running both Linux and Android, and supporting a wide selection of browsers on both operating systems.

Migrating the Website to a Public Server

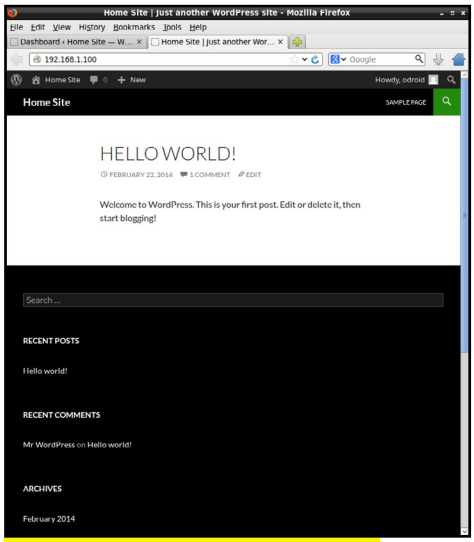
Once the content of the Wordpress website is ready for publication, the site files can be uploaded to the public server either via FTP, or through a standard cPanel application, depending on the hosting service. Filezilla is available on the Code Monkey image for hosting companies that provide access to the website's FTP server. If using the cPanel interface to transfer the website files instead, they may be uploaded directly from the Samba share at `smb://odroid-server/www/`.

Once the web files have been cop-

ied to the public internet server, it's also necessary to migrate the Wordpress database, which contains the site's content and custom settings. Connect to the Quiet Giant image via SSH and type the following command:

```
mysqldump -u odroid
-p'odroid' wordpress > ~/
Public/www/wordpress-data-
base.sql
```

This exports a copy of the database to the file "wordpress-database.sql" on the server's local hard drive. This file may then be imported to the public MySql server directly from Apache's "www" directory, which is already available as a shared drive to the Code Monkey machine. After the database import is completed, the new Wordpress website will be identical to the one created in the sandbox environment.



Working Wordpress Site

Other Sandbox Applications: Hosting a Minecraft Server

Quiet Giant is a powerful OS that also comes with an optimized Minecraft server build called Spigot. Spigot is written in Java, and allows complete customization of any aspect of the Minecraft server. The Spigot files are automatically shared on the local network at `smb://odroid-server/spigot/`, and can also be modified from Code Monkey using any available IDE in the same way that the Wordpress site was edited.

To run Spigot, establish an SSH session to Quiet Giant, then type the following command:

```
cd ~/Public/spigot
sh ./spigot.sh
```

Anyone on the local network may then play Minecraft on the Spigot server, and test out the server's world by using Quiet Giant's static IP address as the server name. Complete support for customizing and administering Spigot is available at `http://www.spigotmc.org`.

Once the Minecraft server has been customized for public use, follow the same procedure for uploading as with the Wordpress site, using either Filezilla or cPanel to send the files from the sandbox server to the public site. Using addons, Minecraft players can submit donations and purchases directly to a PayPal account, which can cover the cost of the public server, and perhaps bring in some extra money on the side.

ODROIDs for Profit

Who knew that a $59 computer could be a core part of your home web development business? Replacing expensive laptop and desktop computers with ODROIDs can save you thousands of dollars in equipment costs. Clients who wish to maintain security while developing their websites can be assured that their product will not be available on the Internet until it's ready, thanks to the privacy of the sandbox environment. The portability of ODROIDs make it easy to write software on the go, making it an ideal solution for a lucrative mobile software development business.

REBOOT YOUR UBUNTU AFTER INSTALLING CPUFREQ

After installing your CPU governor from Issue 2, have you noticed that your Ubuntu distro hangs when issuing a restart? If so, check your current cpufreq policy by typing:

```
sudo cpufreq-info
```

If your policy is set to anything other than "performance", you may have to power cycle your ODROID to reboot. But don't worry, you can also just issue the following command prior to a restart in order to get the job done:

```
sudo cpufreq-set -g performance
```

and bam! Your ODROID is able to restart now.

RESIZE YOUR PARTITION

Did you install a new image on your eMMC card or SD card and forget to allocate the free space on your main partition? You don't need to go through the hassle of disconnecting the card or module and taking it all the way back to GParted on another copy of Linux. Mauro, one of the Hardkernel developers, created a handy script for our convenience:

```
http://forum.odroid.com/
download/file.php?id=502&sid
=842ba747c84c171245591847c5
53b7af
```

Make it executable:

```
chmod +x resize.sh
```

run it as sudo

```
sudo ./resize.sh
```

Then, reboot your ODROID. Once the boot is completed, run the command again to allocate the free space, and you are all set!

MEET AN ODROIDIAN

ROB ROY: CHIEF EDITOR OF ODROID MAGAZINE

by Robert Hall

Rob Roy, a frequent contributor to the ODROID forums, was recently selected by Hardkernel to publish ODROID Magazine due to his passion for ARM technology. Currently living in San Francisco, California, Rob has worked in the software engineering industry for over 20 years, with many stories to tell about the early days of computing. He's been recognized for his innovative contributions by many high-profile clients such as PNC Bank, Cleveland Indians, BP, Chevron, PPG, Hyundai, Dolby Technologies, Hi5, and VEVO.

Heavenly ski trip in South Lake Tahoe 2014

How did you get started with computers?

When I was about 10 years old, my parents let me visit with my uncle Jack and aunt Eydie in Pittsburgh, Pennsylvania for a few weeks, as a summer vacation. Jack was a mechanical engineer, and used an Apple][+ and an Apple][e in his job to perform stress calculations. I was very interested in math, and so he gave me a couple lessons in how to use the computer, along with a typing program called Letter Invaders.

That weekend, he took me to a local Apple Bytes enthusiast club, where I got to meet a lot of really smart and fun people who shared a common interest of exploring the possibilities of these revolutionary new home computers. It was absolutely amazing what could be done with 64KB of RAM and 113KB 5-¼" floppy disks back then. After the meeting, I played the game Ultima 3 non-stop for a few days, then started learning BASIC from my uncle, since the language was built into the Apple hardware.

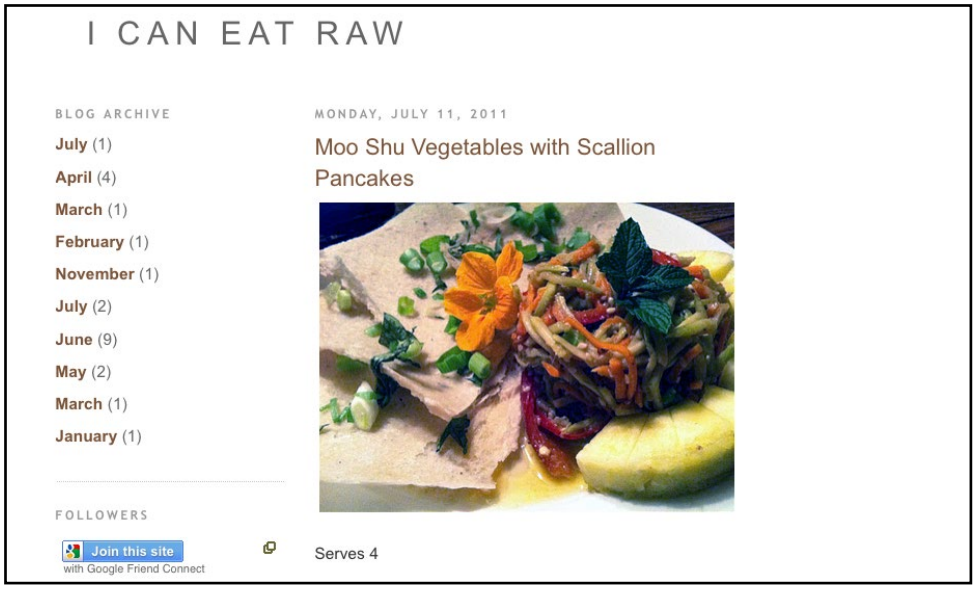
When I got home from my vacation, I told my parents exactly what I wanted for Christmas: an Apple //c. My brother and I played games on it, did our homework, and even wrote little programs to do simple things like bouncing a pixel around the screen, or keeping track of the shopping list for the week. We later got an Amiga 2000, which was a very advanced computer for its time. I attended Carnegie Mellon University, where I had access to a Cray YMP through my on-campus job, and got to explore the early internet, when it was only used by a handful of universities, before HTTP was invented.

What do you like most about the ODROID community?

It reminds me of that first Apple Bytes meeting with my uncle, where people from all walks of life and backgrounds got together to speak the same language. ODROIDs provide an opportunity for everyone to learn and experiment with ARM technology in a supportive environment. The depth of knowledge available in the forums, and the worldwide appeal of ODROIDs, bring back that feeling of something exciting about to happen. I think that ODROIDs herald a new revolution of affordable home computing, which is a long-awaited breath of fresh air for the technology world.

What other interests and hobbies do you enjoy?

I have been a vegetarian for almost 20



years, and follow a living foods diet. For those who are curious about what a living foods diet consists of, I have many free and original gourmet recipes available at `http://icaneatraw.blogspot.com`. I also enjoy almost any type of exercise, including running, skiing, swimming, biking and martial arts. I earned a second-degree black belt in Hapkido after college, and just recently came back from a ski trip with my girlfriend at Heavenly in South Lake Tahoe. I do a hot springs retreat at least once a month, to relax and enjoy the beautiful California Redwoods.

What motivated you to produce the OS images available on the forums?

I run a web development company from my home office, and I was looking to replace my trusty 2008 HP laptop with something more modern and portable. I read an article on Wired about the Raspberry Pi, and remembered using RISC-based machines in college. That led me to research and buy the most powerful ARM board that I could find, which happened to be the ODROID-X2. After seeing what it could do, I set a goal for myself to completely replace my Windows XP machine with my new ODROID, while still doing everything that I used to do in Windows, by instead using Ubuntu and Android on my X2.

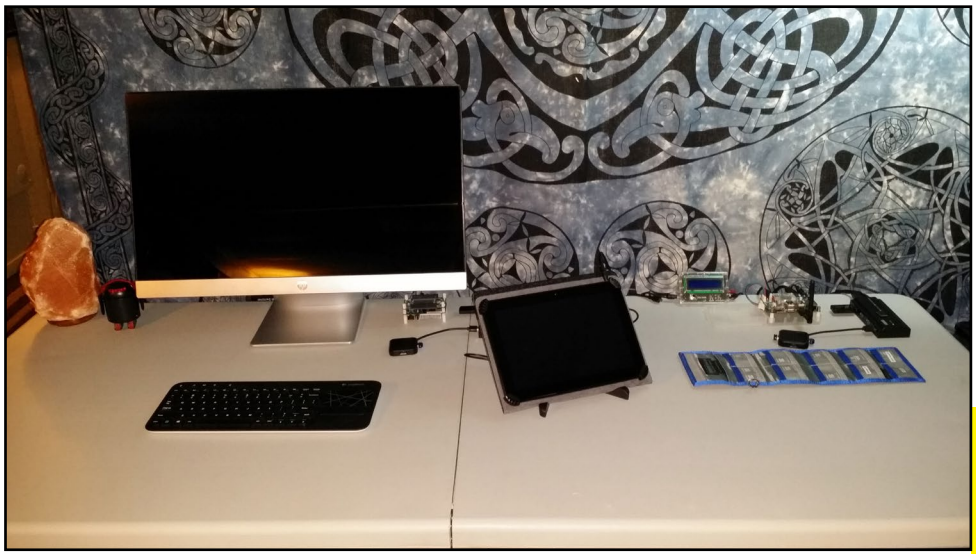
When I got my first ODROID, I knew next to nothing about Linux, coming from a Microsoft and Apple background, so I kept reading and asking questions on the ODROID forums. I wanted to tweak the Linux OS to be more like my old Windows XP environment. I tried an ODROID port of Slackware, which was graphically fast, but very challenging for a beginner to customize. The forums were supportive of experimentation, and other ODROID contributors inspired me to want to learn more.

For my web business, I decided to use the official Linaro 12.04 image which came with the standard Unity desktop. I started hacking away on the command line until I got rid of all of the warnings and errors in the kernel log. Then, I installed Synaptic and configured every desktop environment that I could find, and downloaded all of the equivalent Linux programs to those that I had used in Windows. Throughout the process, there were many software obstacles that required research, questions, and perseverance in order to achieve a completely stable environment. I really learned a lot about Linux from that experience.

After learning that I could copy an image using the "dd" command, I published my first pre-built OS called Fully Loaded, which included several desktop environments such as Unity, KDE Plasma, LXDE, Gnome and Xubuntu. After that, I was encouraged to create a few more images for my own personal use such as Pocket Rocket and Whisper, and realized how easy it was to share what I had done with others.

I currently offer more than a dozen images to the ODROID community, and receive PayPal donations from many community members who appreciate the time and effort that I put into making them. I very much enjoy interacting with people from around the world, and by learning more about Linux, I've been able to help others find new ways to use ODROIDs to fit their needs.

Rob Roy's ODROID software contributions may be downloaded for free from `http://oph.mdrjr.net/robroyhall/`.



The Ultimate ODROID Setup: U3, XU, Q2 and a BERO bluetooth robot
`http://www.betherobot.com`

THANK YOU

THIS MAGAZINE ISSUE ENDED

you can now stop reading....

OKAY, OKAY

SO, BY POPULAR DEMAND

and because you probably skipped ahead to read about how to play all the cool games on the front cover...

LINUX GAMING ON ODROID

THE RIGHT SYSTEM FOR YOUR GAMES (PART TWO)

by Tobias Schaaf

Last month, I presented and discussed many of the different emulators available for the ODROID platform. I compared some of them and showed what the differences are, and how the emulators have evolved over the years. If you are unsure where to find the games you want to play, or choosing what system to pick for your games, this article will help you find some answers.

Choosing the right system for your favorite game genre

There are plenty of genres to choose from, but which system is best for which genre, and what games for your genre exists for which system? If you read my last article, you know that, for example, you won't find adventure games for the MAME or NeoGeo emulators, but where do you find them? What system offers the best racing games, and which has the best strategic games?

Since there are many genres and many systems to play on, this will be once again reflect my personal preferences, is based on my own experiences.

Adventure

If you want to play adventures, you should definitely go for ScummVM!

ScummVM is an awesome piece of work. There are tons of Adventure games and ScummVM just plays them very well on the ODROID.

It runs the famous Monkey Island series:

And other classics such as Day of the Tentacle, The Dig, Beneath a Steel Sky, and Broken Sword:

More information about ScummVM can be found here: `http://scummvm.org`.

ScummVM, a testment to the classic age of point and click adventure games with compelling stories great 16bit art and an ambience that even the most die hard indie developers are still struggling to recapture for gamers worldwide

There are many awesome Adventure games available for the ODROID, and you can play them with either mouse, keyboard, joystick or gamepad. There were some adventures available for console systems, but they are mostly ports of the original PC or Amiga games. So, there is no need to consider the console Adventure ports, since the same games are most likely available for ScummVM, which also supports different versions of the same game. This means that you can even play the original Amiga version of Monkey Island or the Mac OS version of Day of the Tentacle directly on ScummVM.

Action

There are many Action subgenres such as shooter, fighting games, beat 'em up, side and top down scrollers. If you look into the gigantic MAME library and my personal favorite, the NeoGeo, you will find hundreds of Action games.

Need to make your blood boil, sharpen your reflexes and wish to complete your DIY arcade controls? So, look no further than MAME games, with the added benefit of not having to spend bags and bags of quarters to end those darned difficult games.

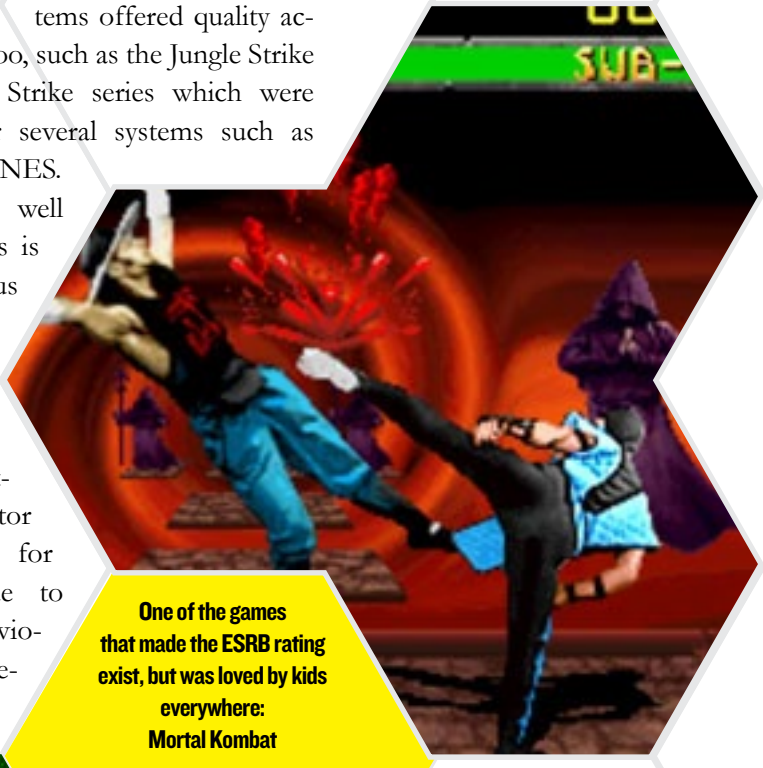
Some notable examples include the Metal Slug Series, 1944 - The Loop Master, Blazing Star, Gun Force 2, King of Fighters Series, and Last Blade 2:

If you really like fast action games, beat 'em up, action scrollers and shooters, these games are what you're looking for, and ODROID does a really good job with this genre. The best part is that most of these games are made for two or more players, so that you can play with a friend on your ODROID and enjoy these kind of games together.

There are many action games available for all of the major consoles, and most of the games that exists for MAME and NeoGeo were subsequently ported to other platforms. They are generally somewhat easier than the arcade originals, so if you want the original gaming experience, I strongly recommend MAME emulation.

Other console systems offered quality action games too, such as the Jungle Strike and Desert Strike series which were available for several systems such as Amiga and SNES.

Another well known series is the infamous Mortal Kombat games, which is a brutal fighting emulator not suitable for children due to its extreme violence and realistic blood effects.



One of the games that made the ESRB rating exist, but was loved by kids everywhere: Mortal Kombat

A little reminder of 90's-era American endeavors to keep a changing world at peace: the "Strike" series.

Jump 'n Run or Platformer

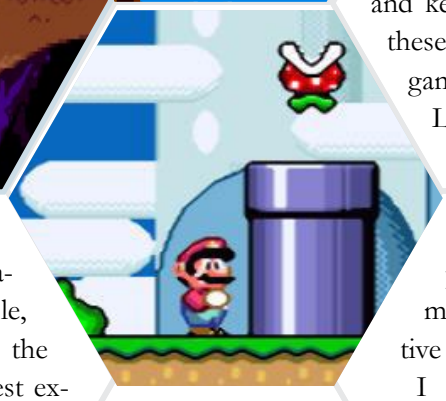
Although it's easier to associate Adventure and Action games with specific emulators and systems, it's more difficult to determine which platform to use when looking for Platformer games to play, as they are available for every system.

Platformers, also known as Jump 'n Run, is an extremely popular and timeless genre. Classics such as Earthworm Jim, Turrican, Aladdin and Asterix are rewritten for each new console as either modernized versions or faithful remakes of the originals.

But how can you find a specific Platformer game that you're looking for?

Nearly every big Disney production has its own Platformer or Jump 'n Run, along with many other cartoons from the 90s, such as Aladdin, Lion King, Asterix, Tiny Toons, Bat-Man, SpiderMan, X-Men, Mickey, Kirby, Donkey Kong, and so on.

Up until the GameBoy Advance (GBA) these types of games were very common, but on newer generations of



consoles, such as the PlayStation 1 or the PlayStation Portable, there are fewer versions of the classic Platformers. So, the best experience for the jump 'n run genre will be found with the NES, SMS, Genesis, SNES, and GBA emulators.

There are some great Action games on the Amiga as well. Back in the day, I really enjoyed a game called Flash-Back which was interesting because it offered lots of different sceneries and tasks. FlashBack is considered a more "mature" platformer, and also exists for other systems such as the SNES. It was even recently remastered for PCs again with a modernized look.

Strategy

Strategy games are not very common around consoles, but there are a few games such as Final Fantasy Tactics or Advanced Wars.

However, those games are nothing compared to the real strategy games found for PC (or some for Amiga as well), and as their name suggests, they are more about tactics than real strategy.

Games such as Dune 2 (the original RTS), Command and Conquer, and round based strategy games such as Battle Isle and Historyline 1914-1918 are hard to find on

The 16-bit console era will always be remembered by its many mascots that shaped the gamer's imagination, and kept us in love with those heroes in updated forms

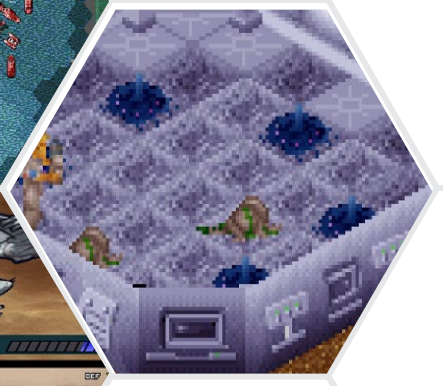
consoles, since they are played best with a mouse and keyboard. For these types of games, a native Linux program instead of an emulator provides the most interactive experience.

I really love those very early strategy games. I spent weeks finishing all the levels of Dune 2 when it first came out, and today there are some really nice remakes such as Dune Legacy.

There is an abundance of Strategy games that run natively on Linux. Some are remakes from old classics, and some are new games that exist for Linux and other systems. Examples of the Strategy genre include Free Heroes 2, Battle for Wesnoth, Zod Engine, Advanced Strategic Command, Crimson Fields, OpenXCom, Jagget Alliance 2, Unknown Horizons and Widelands.

If you're looking for strategy games, it's rather easy to find good games that run directly on Linux, and with an ODROID, you have plenty to choose from. You can experience your old favorites with games such as Zod Engine, OpenXCom, Jagget Alliance 2, or you can try new original games such as Battle for Wesnoth. There are a lot of games in the Debian and Ubuntu software repositories, as well as some that I the and-

Careful strategy, resource control, learning to adapt quickly to changing situations, and thinking about the consequences of your actions. If you have ever played a strategy game, you know what we are talking about



created and posted online in my repository mentioned at the beginning of this article. So, if you like strategic games like I do, you will love playing them on an ODROID in high definition.

Racing

Although it's not my favorite genre, I used to play some of the racing games, and it really is a matter of personal preference. If you like the old 2D classics, such as the OutRun series, or the Lotus series, or the all famous Mario Kart, then the SNES, Amiga and Sega Genesis emulators will offer the most choices. There are also several bird perspective racing games like the old Micro Machines racing games.

You can also play more modern racing games on the PS1 or the PSP, such as the 3D racing games Gran Turismo and Asphalt: Urban GT 2.

The Genesis also has some racing games with good looking graphics and catchy music, and the PS1 and PSP offer several famous series such as Need For Speed and Gran Turismo.

From the technically focused games like Gran Turismo to just plain fun ones like Mario Kart, racing games put our competitive instincts in motion, and make us want to become good enough to finish in Ist place.

Simulation

Simulations such as CrosixTH or OpenTTD can run directly on Linux for the ODROID without an emulator.

I love Corsix TH! It's very fun to see many crazy illnesses you can discover and cure, and the game has some lovely animation. I also really like OpenTTD. I played the original Transport Tycoon on my first DOS-based PC. It was awesome to see all that money flowing in, and I really wished it was that easy in real life. OpenTTD is unique because of the online feature, and the multitude of add-ons available for the game. You even can have old trains from the 19th century.

What other simulation games exist for the ODROID? Some of them run on emulators, such as Theme Park for the Amiga and SNES.

Many simulation games exist as native Linux ports. Examples include Widelands, a clone from the old Settlers series, and Unknown Horizons, which is similar to the Anno series. FreeCiv and FreeCol are clones of the Civilization and the Colonization series. I also enjoy playing the original Colonization on the Amiga because of its awesome music and entertaining graphics.

There are some exceptionally well-done 3D space simulators available on the ODROID. The FreeSpace series, which I loved to play on the PC, runs great on the ODROID. It has stunning graphics, engaging gameplay, and is a really big game! There are huge destroyers in the game that are literally a thousand times bigger then your space fighter.

The best place to look for your favorite simulation games is the Software Center included with your Linux distribution.

Just five more minutes! How many times have you told yourself that when playing an engrossing board or computer game, only to be surprised when the sun comes up?

Role Playing Games (RPG)

Role Playing Games is actually one of my favorite genres, and there are quite a lot of RPGs available, with way too many to mention them all. Some highlights include the now famous Final Fantasy saga, the Tales series and Chrono Trigger. Although NES, Sega and SNES offer a few great RPGs, such as Link of Zelda, Secret of Mana and Earthbound, but RPGs have evolved over the years. As I mentioned in my previous article, newer emulators equals bigger ROM sizes which equals more content, resulting in better looking games.

Irrefutably bounded to Japanese anime, RPG will take you to wonderous journeys that will kept you wanting one more gaming sequel!

Ultimately, I prefer the GBA for RPG games, especially the Summon Night series. I really love its fighting style, which is rather rare for a Role Playing Game. You actually have to fight your enemies in real time with different weapons, and you jump, block and force them back while using a set of spells. It's all very fun to play, especially since the game permits forging special weapons of different types with unique attributes, depending upon the Materials used. All of this adds up to great gameplay, especially for a game from that era.

I also enjoy paying "Riviera – The Promised Land", which has a very deep story and a great fighting style as well. I recently found out that Riviera was remade for the PSP, which has pretty much the same graphics with improved effects and voice acting as an improvement over the GBA version. I also recommend some of the well-done Dragon Ball Saga games for the GBA.

I mainly recommend the GBA, PS1 and PSP emulators on GST if you enjoy RPG games.

First Person Shooter (FPS)

You won't find many, if at all, FPS games on the first generation of video consoles such as the NES and SNES. However, titles such as Battlefield 3, Battlefield 4 and Call of Duty, all of which are available for current Xbox and PlayStation consoles, are proof that people like to play FPS games on console systems.

As a side note, I never understood the attraction of playing an FPS game on a console. It's called "point and shoot", not "swirl around and shoot", and the mouse is a natural pointing device whereas a gamepad is not. Apparently a lot of console gamers do not share my opinion, but that's fine.

Anyway, the ODROID offers some nice FPS titles as well, such as Quake 3 Arena (Open Arena) or World of Padman which offer fast multiplayer action on the ODROID. We even have some very nice exotics such as the Jedi Knight series.

The Jedi Knight games are not pure FPS but as we all know, it's just more fun to slice the enemy up with your light saber!

Conclusion

As you can see, the ODROID offers many options when it comes to games. There are quite a few more sub-genres than those detailed here, such as sport games like soccer and tennis, puzzle games, and other games such as Harvest Moon which are hard to fit into any of the main genres.

With the ability to emulate many different systems on the ODROID, you have the world of gaming at your fingertips, so grab your controller, mouse and keyboard, and see how well you can do on that big TV of yours!

Tobias, a long-time contributor to the ODROID forums and Linux Gaming columnist, produces a popular Gaming OS called ODROID GameStation Turbo with XBMC, available for free download at `http://oph.mdrjr.net/meveric/images/`. *GST offers many gaming console and system emulators for the X and U series, along with a custom build of XBMC designed specifically for gamers. He maintains a repository of many of his favorite classic games at* `http://oph.mdrjr.net/meveric/repository/`.