Build your own UBUNTU from scratch • Install Oracle JDK Version 8

# ODROID Magazine

Year One
Issue #4
Apr 2014

# DO IT YOURSELF EDITION

## Off-Roading with an ODROID Truck PC

## GET STEP BY STEP TUTORIALS ON HOW TO CREATE:

## A MINECRAFT SERVER

## A HEAVY DUTY TABLET

## CUSTOM ANDROID BUILD

# What we stand for.

We strive to symbolize the edge technology,
future, youth, humanity, and engineering.

Our philosophy is based on Developers.
And our efforts to keep close relationships with
developers around the world.

For that, you can always count on having the quality
and sophistication that is the hallmark of our products.

Simple, modern and distinctive.
So you can have the best to accomplish
everything you can dream of.

HARDKERNEL

# EDITORIAL

Linaro 12.11, which is reaching its end of life this month, was the last version of Ubuntu to offer the Unity 2D desktop environment, which is popular with both beginners and experts because of its friendly icons, unique desktop customization options, and easy-to-use interface. However, its predecessor and close relative, Linaro 12.04, is still alive and well, and comes with Unity 2D. 12.04 is the most recent Long Term Service (LTS) release, and will be supported for 3 more years, until April 2017. If you're looking for an extremely stable version of Ubuntu, Linaro 12.04 is your best bet.

However, the 12.04 version of Ubuntu isn't available as a pre-built from Hardkernel. Why? Because, as the ODROID box says, you can Do It Yourself! The ODROID family of computers are primarily intended for developers, who love to build everything from scratch for two reasons: 1) they usually get paid by the hour, and 2) they spend days constructing long, intensive build scripts that take hours to finish, so that they can go make sandwiches and drink coffee while they wait for the build to be done!

This month, Mauro shows us how to build a custom Ubuntu image from scratch, so you can amaze your friends at your next party, and show that you are a true Linux hacker, worthy of their adoration and free jelly donuts.

We also are very proud to present an emerging trend in the automobile world: a fully functional computer installed in your car's dashboard! Known as Car PCs, several large computer companies have recently contracted with major car manufacturers to include their hardware as high-priced options in certain high end models.

But who says that Car PCs have to be expensive? Our feature article, the Truck PC, is a guide to building your own onboard computer, as an affordable alternative to supergluing an iPad to your dashboard. Requiring less than 5W of power, the ODROID CarPC and its battery can be charged straight from your electrical system, or by using a small solar panel mounted on the roof. The future of truly mobile computing is here today, and ODROID line of micro-computers are once again proven to be ahead of their time.

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.

HARDKERNEL

# 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

Went a little crazier than usual while studing again how a color blind person sees, and missed having a colorblind coworker as when he worked on EGM Brazil in his old gaming magazine editing days.

**News from Art Editor Bruno:**

You will notice a few changes in this issue. The first is that we are now using a color-coded system at the top of each article to show the level of technical detail. This expands our color palette, and as a bonus, we are mixing our content and including shorter articles on subjects such as Linux Tips and Android Gaming.

We are also changing the format of some of the more technical articles, by including text in two columns when necessary.

Why? well it was a bum-
-mer when we need-
-ed to type a longer str-
-ing of code and it was cut.

This means that, in some cases, we are changing the style from three columns, and my art editing teacher would remind me to keep the layout consistent. However, the articles will be much more comprehensive for you to follow the code, which is the main point!.

From now on, we are going to leave some space at the end of the technical articles, so that any future revisions will have room to grow. Once an article is published, we do a lot of tweaking based on user feedback, and if the technical articles are too tight, the additions can be difficult to adjust.

Finally, we now have a Table of Contents, which you can see on the next page! Cool, huh? Now you won't be caught in my little in-jokes like on the last edition, not that I won't try to put some humor here and there!

# INDEX

# BUILD ANDROID ON ODROID-U3

## FROM SCRATCH TO SMASH, TAKE TOTAL CONTROL OF YOUR ANDROID SYSTEM

by Nanik Tolaram and Fabien Robert

In this tutorial, I will discuss how to build the Android operating system for the ODROID-U3 from source, including the kernel. The Android build system is robust, but also a bit complicated if you haven't used it before. There are steps that need to be done properly in order to have a workable and repeatable build system. By the end of this article, you will hopefully have sufficient knowledge and understanding of how it all works.

### Build Hardware and Environment

I won't go into details in terms of setting up your build server for building Android since Google's own Android page has lots of information at `http://source.android.com/source/initializing.html`. If you are having problems installing JDK 6, follow the steps in this link: `http://askubuntu.com/questions/67909/how-do-i-install-oracle-jdk-6`.

Building the Android source code is a big task and requires a powerful machine. To give you an idea, my computer has the following specification:

**32GB RAM**
**i5 Intel Processor**
**2 x 256GB SSD Drive**

Android build systems do a lot of writing and reading, and this in turns requires constant I/O operations. Even with an SSD drive, you still have to spend a good 25-35 minutes waiting time for the build to complete, and this can be very time-consuming if you have to constantly work with Android on a daily basis. Make sure to have as much free disk space as possible, with the minimum requirement around 100GB. There is another trick to speed up the building process and this is the use of ccache project. In the next section I will outlined on how to use it.

If your hardware is not as powerful as i5 or i7 and you are using a normal hard drive than make sure you have your coffee ready !

### Download Source

The Android source code for version 4.1.2 (JellyBean) that I used in this article can be downloaded from droid.com website at `http://dn.odroid.com/4412/Android/4.1.2_Jan-15-2014/BSP/`. There are a couple files that you need to download from that link, as shown in below:

Download the 2 files android.tgz and kernel.tgz, and extract them to a directory in your local drive. Put the

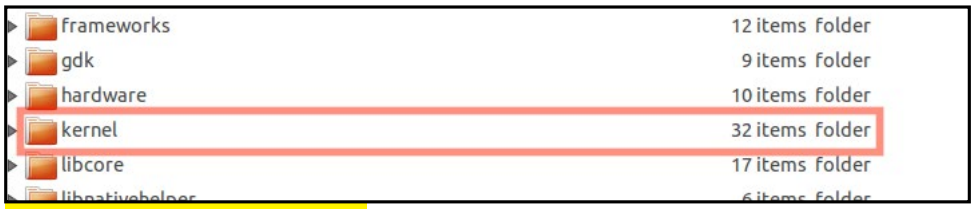
Android and Kernel source code

dn.odroid.com/4412/Android/4.1.2_Jan-15-2014/BSP/

**Index of /4412/Android/4.1.2_Jan-15-2014/BSP**

| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory | | - | |
| android.tgz | 15-Jan-2014 17:30 | 2.4G | |
| android.tgz.md5sum | 15-Jan-2014 17:30 | 46 | |
| kernel.tgz | 15-Jan-2014 17:30 | 104M | |
| kernel.tgz.md5sum | 15-Jan-2014 17:30 | 45 | |
| uboot.tgz | 15-Jan-2014 11:46 | 22M | |
| uboot.tgz.md5sum | 15-Jan-2014 11:46 | 44 | |

*Apache/2.2.22 (Ubuntu) Server at dn.odroid.com Port 80*




**Kernel directory inside Android**

kernel files inside the kernel/ directory under the Android root directory as shown above.

The main reason to place the kernel inside the Android directory is to facilitate the creation of the build script, since the build system revolves around files inside the main Android source directory.

I created a set of patch files for this article on GitHub at `https://github.com/nanikjava/odroid-u-patch`. This patch allows you to build Android and the kernel at the same time. Run the command:

```
git apply --stat ./odroid-u-patch/fix-build-odroid-u3.patch
```

and you will see the output as shown at the bottom of the page.

There are 3 new file and 2 modifications for this patch. Make sure you are inside your Android directory and apply the patch by running the following command:

```
git apply ./odroid-u-patch/fix-build-odroid-u3.patch
```

You will get the following messages which can be safely ignored:

```
./odroid-u-patch/fix-build-odroid-u3.patch:171: trailing whitespace.
./odroid-u-patch/fix-build-odroid-u3.patch:173: trailing whitespace.
    ccache =
warning: 2 lines add whitespace errors.
```

**Git stat of the patch**

```
build/core/tasks/kernel.mk                 |  199 ++++++++++++++++++++++++++++++
buildOdroid.sh                             |    5 +
build_android.sh                           |    5 +
device/hardkernel/boards/vendorsetup.sh    |    1
device/hardkernel/odroidu/BoardConfig.mk   |    3
5 files changed, 212 insertions(+), 1 deletion(-)
```

One additional file required for the build process is Makefile, which should be copied to the kernel/drivers/media/video/samsung/tvout directory.

## ccache and script modification

I mentioned using ccache to speed up the compilation process, and now I will walk you through setting it up. First, you must remember that ccache require some free disk space, and in this case, we are going to set it up to use only 10GB, which will be more that sufficient.

Create a directory anywhere in your drive, then initialize the environment variable and run both of the following ccache commands to initialize it:

```
export CCACHE_DIR=\
<your_ccache_directory>

<your__Android_directory>/
prebuilts/misc/linux-x86/
ccache/ccache -M 10G
```

You can verify whether ccache has been successfully initialized by inspecting the cache directory as shown at the upper right on the page.

The last step is to modify the buildOdroid.sh script to change the ccache directory to point it to you local directory like the following:

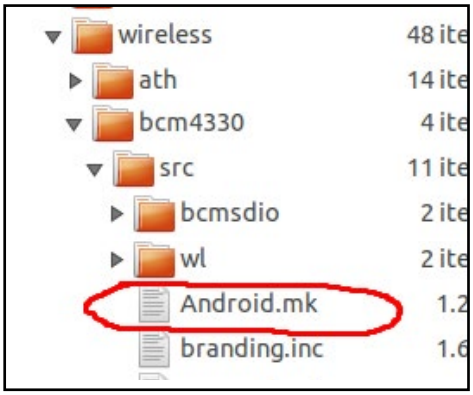
```
drwxrwxr-x 18 nanik nanik 4096 Jan 27 22:39 ./
drwxrwxr-x 28 nanik nanik 4096 Jan 27 22:36 ../
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 0/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 1/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 2/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 3/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 4/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 5/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 6/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 7/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 8/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 9/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 a/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 b/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 c/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 d/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 e/
drwxrwxr-x  2 nanik nanik 4096 Jan 27 22:39 f/
```

**The cache directory and its subdirectories, numbered in Hex from 0 to F**

```
source build/envsetup.sh
lunch odroidu-eng
export USE_CCACHE=1
export CCACHE_DIR=\
<your_ccache_directory>
/usr/bin/time -f "\n%E
elapsed,\n%U user,
\n%S system,\n%M memory,\n%x
status" make -j8
```

## Kernel Modification

There is an unnecessary file that needs to be removed from the `kernel/ directory`, which has to do with building the Broadcom 4330, which is not needed for the ODROID-U. Delete the file `Android.mk` inside the `kernel/drivers/net/wireless/bcm4330/src/` directory as shown below.




## Ready... Set... Go!

Once you have finished the above steps, you are done with the initial prebuild setup. Navigate to the Android source directory, and follow these steps to start building Android:

Run `source build/envsetup.sh`. You will get output as shown next.

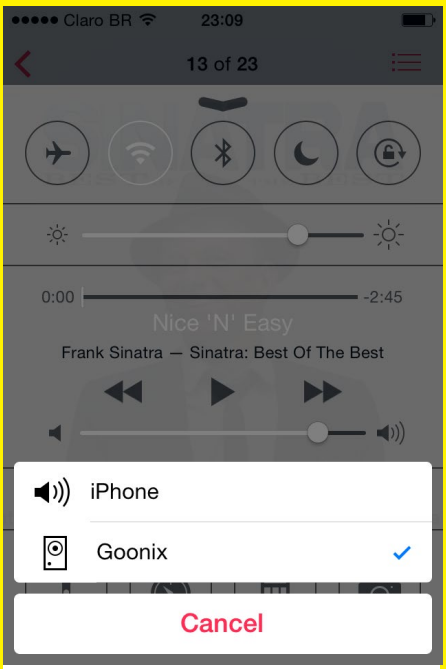
# SHAIRPORT

## TURN YOUR ODROID TO AN ITUNES AIRPORT AUDIO STATION

by Bruno Doiche

Got an old stereo with amazing speakers in need of a little mp3 action? Plug a regular Stereo Male to 2-RCA from your odroid to the AUX ports on the Stereo and do the following on your Linux terminal:

```
sudo apt-get install\
avahi-utils libmodule-build-perl\
libio-socket-inet6-perl libao-dev\
libssl-dev libcrypt-openssl-rsa-perl\
libwww-perl pkg-config
git clone https://github.com/
njh/perl-net-sdp.git perl-net-sdp
cd perl-net-sdp/
perl Build.PL
sudo ./Build
sudo ./Build test
sudo ./Build install
cd ..
git clone https://github.com/
hendrikw82/shairport.git
cd shairport
make
./shairport.pl -a name
```




Goonix, what a cute hostname for a machine... But wait!? Isn't that the same computer used to do the PS3 media server in the last issue? Such versatility!

```
including device/asus/grouper/vendorsetup.sh
including device/generic/armv7-a-neon/vendorsetup.sh
including device/generic/armv7-a/vendorsetup.sh
including device/hardkernel/boards/vendorsetup.sh
including device/moto/wingray/vendorsetup.sh
including device/ti/panda/vendorsetup.sh
including sdk/bash_completion/adb.bash
```

Console output of envsetup.sh

Run lunch and you wil be shown selection as shown below. Select the 7th option.

```
You're building on Linux

Lunch menu... pick a combo:
     1. full-eng
     2. full_x86-eng
     3. vbox_x86-eng
     4. full_grouper-userdebug
     5. mini_armv7a_neon-userdebug
     6. mini_armv7a-userdebug
     7. odroidu-eng
     8. full_wingray-userdebug
     9. full_panda-userdebug

Which would you like? [full-eng]
```

**You shouldn't need to think that 7 is a number to pick just to get lucky, it is in fact the correct option here.**

Inform ccache of the directory that it will use to cache the compiled file, and also an environment variable to inform the build process that we want to use ccache.

```
export USE_CCACHE=1
export CCACHE_DIR=<your_cca-
che_directory>
```

The final step is to execute the build process by typing make -j4.

I've created a script called buildOdroid.sh that you can use to build Android, but it's good if you can go through the above steps to get a flavour of the build steps.

After completing all the above steps, you will see the build process run, which will take some time to build. On my machine, with caching enabled, it took on average 18 minutes. At bottom we show what you should be seeing when you get successful compilation.

## Image files

After the build is completed, the resulting binaries and image files (.img) are found inside the directory `out/target/product/odroidu/` .

```
-rw-rw-r--  1 nanik nanik    167626 Jan 27 21:36 ramdisk.img
-rw-rw-r--  1 nanik nanik   1955584 Jan 27 21:36 ramdisk-recovery.cpio
-rw-rw-r--  1 nanik nanik   1388672 Jan 27 21:36 ramdisk-recovery.img
-rw-rw-r--  1 nanik nanik    167690 Jan 27 21:36 ramdisk-uboot.img
-rw-rw-r--  1 nanik nanik       518 Jan 27 21:31 README.txt
drwxrwxr-x  3 nanik nanik      4096 Jan 27 21:36 recovery/
-rw-r--r--  1 nanik nanik   4526080 Jan 27 21:36 recovery.img
drwxrwxr-x  8 nanik nanik      4096 Jan 27 21:36 root/
drwxrwxr-x  6 nanik nanik      4096 Jan 27 21:36 symbols/
drwxrwxr-x 14 nanik nanik      4096 Jan 27 21:35 system/
-rw-r--r--  1 nanik nanik 234864788 Jan 27 21:46 system.img
drwxrwxr-x  3 nanik nanik      4096 Jan 27 21:33 test/
-rw-r--r--  1 nanik nanik  36463328 Jan 27 21:37 userdata.img
drwxrwxr-x  2 nanik nanik      4096 Jan 27 21:36 utilities/
```

There are several different .img files that you will need for your ODROID-U:

**boot.img • this file is the main bootloader image that contains the u-boot, kernel and ramdisk files**

**recovery.img • this file is used to create the recovery partition**

**system.img • this is the main image file that contains the full Android system and applications**

## Cleaning up

If you run into a problem during the compilation, make sure to "clean" the directories using the make clobber command before restarting the build process. This will delete all of the binary/compiled objects that are produced during the compiling/linking stage.

```
make_ext4fs -s -l 1073741824 -a system out/target/product/odroidu/obj/PACKAGING/systemimage_intermediates/s
+ make_ext4fs -s -l 1073741824 -a system out/target/product/odroidu/obj/PACKAGING/systemimage_intermediates
Creating filesystem with parameters:
    Size: 1073741824
    Block size: 4096
    Blocks per group: 32768
    Inodes per group: 8192
    Inode size: 256
    Journal blocks: 4096
    Label:
    Blocks: 262144
    Block groups: 8
    Reserved block group size: 63
Created filesystem with 1277/65536 inodes and 61173/262144 blocks
+ '[' 0 -ne 0 ']'
Install system fs image: out/target/product/odroidu/system.img
out/target/product/odroidu/system.img+ maxsize=1096220928 blocksize=4224 total=234864788 reserve=11075328

32:26.38 elapsed,
5206.56 user,
346.32 system,
3510740 memory
0 status
```

Android has been built successfully! Good job!

# PORTABLE IMAGE BACKUP

## CREATING A RECOVERY FILE FOR YOUR FAVORITE OPERATING SYSTEM

**by Rob Roy, Chief Editor**

**Good backup habits will keep you safe from bad luck, evil pets, and especially your own hubris**

Once you've got your ODROID set up the way that you like, it's important to make sure that you can restore your system quickly and easily. If you enjoy experimenting with Linux or Android, need to install your OS on several ODROIDs, or want to keep a backup in case of disk failure, you can do so by making an image of your SD card or eMMC module. An image file is an exact bit-for-bit copy of the original disk, complete with bootloader, kernel, root file system, and user files.

To begin, power down the ODROID and remove the eMMC or SD card that you'd like to backup. Using another Linux host computer with an SD card to USB adapter, plug the SD card or eMMC adapter into the USB port. If using an eMMC module, attach the SD card adapter that came with your ODROID before inserting it into the SD card slot. Depending on which operating system your host is running, the procedure for backing up your disk to an image file will be somewhat different.

### Windows

Ugh.. Windows

Hardkernel publishes an improved version of Win32 DiskImager that automatically fills the disk with zeroes before writing the image. It's available for free download at `http://bit.ly/1lYQ7MF`, and is very easy to use. Simply select the USB drive in the dropdown, choose the image file destination using the folder button, and press "Read".

Depending on the size of your SD card or eMMC module, the backup process may take anywhere from 15 - 60 minutes. The resulting .img file will end up being the exact size of the disk that was copied, so make sure to have enough disk space available first. Note that the image backup should be done on an NTFS partition, since DiskImager will be unable to write a file larger than 4GB to a FAT32 disk.

After the image has completed, we can make it more portable by compressing the file using the xz utility, which has the advantage of very high compression ratios. If xzip is not already installed, download and unzip the prebuilt Windows binaries at `http://tukaani.org/xz/`, then copy the appropriate version of xz.exe to the same directory as your backup file. Type the following command into a Windows command prompt, after navigating to the correct directory:

```
xz -z mybackup.img
```

This step will also take some time to complete. After the compression is done, a file called mybackup.img.xz will replace the original .img file. This can shrink the file up to 80%, depending on the amount of data stored on the original operating system. Make backups of your backup by storing several copies on different disks, in order to ensure that you won't lose your valuable data.

When it's time to recover the backup image by writing it back to an SD card or eMMC module, use the xz command again to decompress the backup file:

```
xz -dk mybackup.img.xz
```

This will recreate the original .img file by reversing the compression algorithm. Note that the -k option preserves the original .img.xz file, so that it may be reused later to do another recovery.

Finally, go back to Win32DiskIm-

# RENAME
## YOUR FILES FROM UPPERCASE TO LOWERCASE IN ONE COMMAND LINE

by Bruno Doiche

Ever needed to organize the files in your directories, but have a bunch of misfits that need to be renamed to comply to your so dreamed orderly database of files? Sure, when they are few, you just issue the mv command and resolve. But what if they come in hundreds?
Issue the following syntax on your Terminal:

```
for i in *; do mv $i $(echo $i | tr [:upper:] [:lower:]); done
```

It's that easy!

# PROTECT
## YOURSELF FROM SUPERUSER ACCIDENTS

Whenever you are editing system files on your text editor, do you go to superuser mode using `sudo` or `su` ? Break this dangerous habit of exposing yourself to an accidental file deletion, move or reboot by creating a script that will keep your environment safe. Let's call it `autosudo.sh`

```
#!/bin/bash
   FILE=$1
# Check Write Permission
   if [ -w $FILE ]
then
    /usr/bin/vim $FILE
else
# Sudo If We Dont Have Write Permissions
    sudo /usr/bin/vim $FILE
fi
```

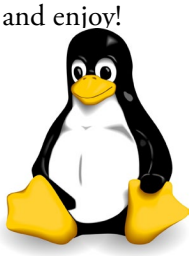
Give it executable pemissions with `chmod +x` , copy it to `/bin` run systemwide and then edit like this:
`autosudo.sh yourfile_to_edit`





ager and select the destination disk for writing the image from the dropdown, choose the .img file with the file explorer, and press "Write". Note that it must be at least the same size or larger than the original disk. After the process completes, the selected disk will be an exact copy of the your original operating system. Insert the new disk into your ODROID, power it on and enjoy!

## Linux



Yeah Linux baby!

In true Linux fashion, image backups are done entirely from the command line. If the xz binaries are not yet available on your system, type `sudo apt-get install xz-utils` to install them. Then, mount the SD card or eMMC module by double-clicking on the USB adapter's desktop icon. Type `df -h` in the Terminal window and make note of the device name, which will be in the format `/dev/sdX`.

Navigate to the directory where the image file is to be stored, then type the following command, substituting the device name of the USB adapter noted in the previous step for `/dev/sdX`:

```
sudo dd if=/dev/sdX bs=1M of=./mybackup.img
```

Just like Windows, after the Read operation is completed, xz is used to compress and decompress the image file for portability:

**Compress an image file using xz**

```
xz -z mybackup.img
```

**Decompress a zipped image file using xz**

```
xz -dk mybackup.img.xz
```

When writing the decompressed image to a new card, use the same dd command as the Read operation with the input file (if) and output file (of) options reversed:

```
sudo dd of=/dev/sdX bs=1M if=./mybackup.img
```

## Mac OSX

We may not talk often about Macs, but the magazine art is made on OSX

The procedure for creating an image file using OSX is similar to Linux, with three small differences. First, instead of using apt-get to install xz, download the xz-utils package from the same website mentioned in the Windows instructions above, making sure to select the OSX binaries (`http://tukaani.org/xz/`). The other differences are that the block size (bs) parameter for the dd command is in lowercase, and the USB adapter's device name is in the format `/dev/diskX`:

**Read from the original disk to an image file using OSX**

```
sudo dd if=/dev/diskX bs=1m of=./mybackup.img
```

**Write from an image file to a new disk using OSX**

```
sudo dd of=/dev/diskX bs=1m if=./mybackup.img
```

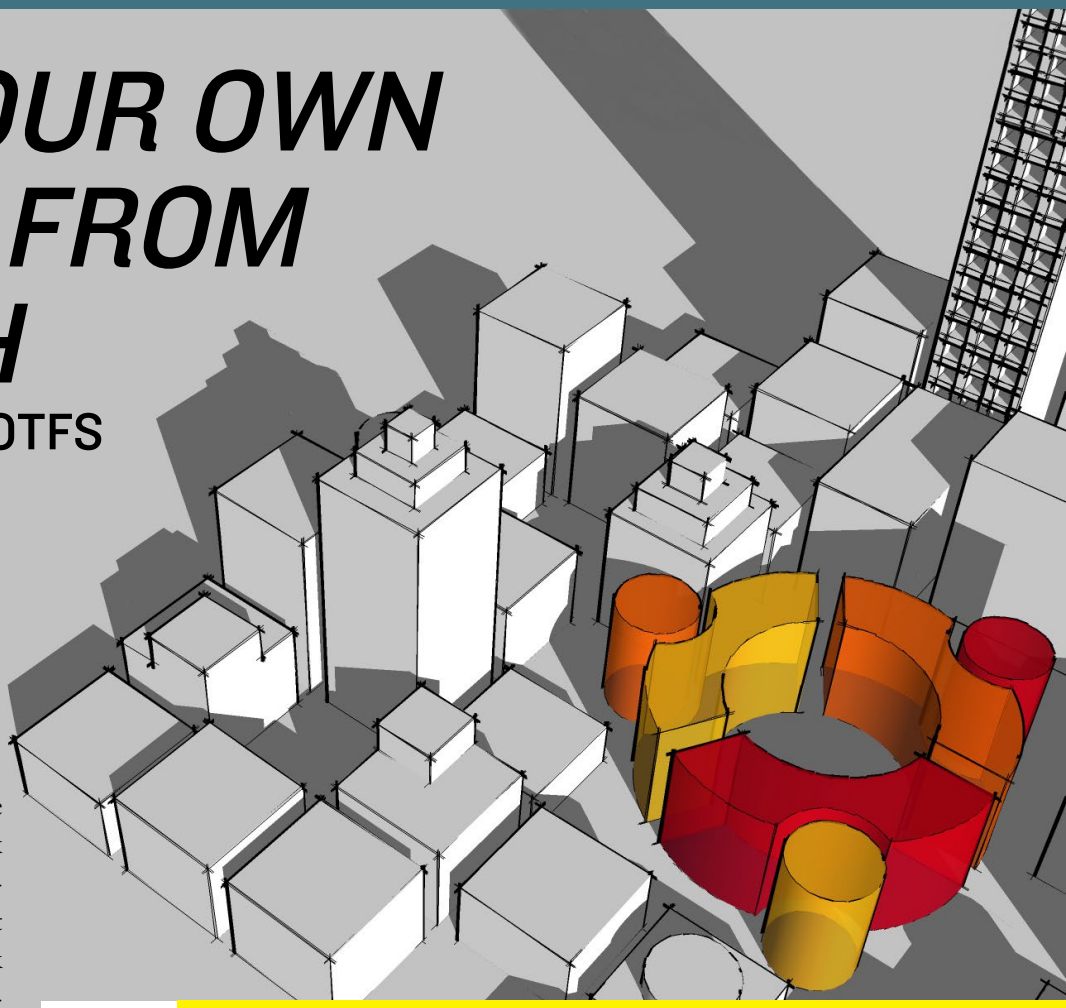
It's a good idea to make a backup of your system before a major upgrade is attempted, a challenging configuration is completed, or a large set of software packages have been installed. In case the original disk becomes corrupted, a compressed image backup will also get you back on track quickly, without needing to take the time to reinstall and reconfigure the entire system.

If your data is important enough, It's also a good idea to keep a backup archive as well as some offsite copies for safekeeping. You can never have too many backups!

# BUILD YOUR OWN UBUNTU FROM SCRATCH

## USING LINARO'S ROOTFS TO COMPILE LINUX LIKE A PRO!

by Mauro Ribeiro



Soon, Ubuntu stores will be the favorite hangout spot for a future ODROIDian society.

A major advantage of open-source operating systems such as Linux is having the option to download the source code and compile it yourself. You can add patches, tweak the code, and inspect it for bugs without needing to wait for an official release or update. The ODROID platform can run many different operating systems, and some of them are not available as pre-compiled ARM images. Taking the time to learn how to build your system from scratch enables you to download newly developed operating systems and try them out. In this example, Linaro's version of Ubuntu will be used to demonstrate how easy it is to take control of your OS at the most basic level.

### General Notes

**• This guide was tested on a host computer running Ubuntu 13.10 64-Bit with ia32 libs installed.**

**• Free up at least ~10GB of disk space on your host computer.**

**• Set aside some spare time.**

**• If something goes wrong, start over.**

**• The default user and password is "linaro".**

**• TTY1 will auto-login as root.**

## Setting up the environment

```
$ cd ~
$ mkdir ubuntu-guide
$ cd ubuntu-guide
$ export GUIDE=`pwd`
$ export SDCARD=/dev/sdX
```

Make sure to replace X with the correct letter of your SDCard.

## Downloading all the necessary components

**• Pre-built bootloaders**

This article won't cover bootloader building because nothing changes over the pre-built bootloader provided on the Hardkernel developer website.

```
$ wget odroid.in/guides/\
  ubuntu-lfs/boot.tar.gz
```

**• Kernel Sources**

```
$ git clone --depth 1 \
  https://github.com/\
  hardkernel/linux.git -b \
  odroid-3.8.y odroid-3.8.y
```

**• Toolchain for Crossbuild**

In this guide, I'm using GCC 4.7.2 from Archlinux ARM as my toolchain. I like this toolchain given the known stability of this version.

```
$ wget odroid.in/guides/\
  ubuntu-lfs/arm-unknown-\
  linux-gnueabi.tar.xz
```

**• Linaro's rootfs**

I chose to use Linaro's rootfs because it comes easily packed as a .tgz file, and will work very well for this guide. At the moment of this writing, Linaro's 13.12 is what was available, and any other rootfs should work just fine.

```
$ wget http://releases.\
  linaro.org/13.12/ubuntu/\
  arndale/linaro-saucy-\
  server-20131216-586.tar.gz
```

#### 5. Compilation tools

U-Boot tools comes with a tool called mkimage we need that to create a boot.scr.

```
$ sudo apt-get install
u-boot-tools build-essential
libqt4-dev perl python git
pkg-config ncurses-dev uuid-
runtime lib32z1 ia32-libs
lib32ncurses5 lib32bz2-1.0
```

## Building the Image

#### 1. Emptying your card.

I like to start with a clean SD card.

```
$ sudo dd if=/dev/zero \
  of=$SDCARD bs=1M
```

#### 2. Installing bootloaders

```
$ tar zxvf boot.tar.gz
$ cd boot
$ chmod +x sd_fusing.sh
$ sudo ./sd_fusing.sh \
  $SDCARD
```

#### 3. Create Partitions

We use two partitions, one for kernel+initrd(if used) and one for rootfs. The kernel+initrd partition is a FAT32 type, and the rootfs is a ext4 partition with no journal and no "atime as mount" option.

It's also important on this step that the first partition starts at least 3072 sectors later, since this is the bootloader space.

```
$ sudo fdisk $SDCARD
n
p
1
3072
+64M
n
p
2
134114
<just press entere here>
t
1
c
w
```

This can be slightly cryptographic for some users, but it's quite simple:

**n = new**
**p = partition**
**1 is the number of the partition that we are creating**
**3072 is the start address of this partition**
**+64M is the size of this partition, this is the FAT32 partition, so doesn't have to be big**
**n creates a new partition**
**134114 is the start of partition 2, which is right after partition 1**
**We don't tell the size to fdisk and leave it empty so it can use the rest of the sdcard**
**t = type**
**1 is our partition number**
**c is the type for Fat32 partition**
**w = write**

After all this, call partprobe to get the new partitions recognized by the kernel:

```
$ sudo partprobe
```

#### 4. Format and mount the partition

We need to format the partitions and change the UUID so later on you can use the kernel-update script:

```
$ mkfs.vfat -n boot \
  $SDCARD"1"
$ mkfs.ext4 -L rootfs \
  $SDCARD"2"
```

Now that the partitions are formatted, let's change the UUID of the ext4 partition:

```
$ tune2fs $SDCARD"2" -U \
  e139ce78-9841-40fe-8823-
96a304a09859
```

Then, disable journaling to prevent excessive wearing of the card:

```
$ tune2fs -O ^has_journal \
  $SDCARD"2"
```

Then, mount the partitions:

```
$ mkdir rootfs && mkdir boot
$ sudo mount $SDCARD"1" boot
$ sudo mount $SDCARD"2" \
  rootfs
```

#### 5. Install the rootfs on our sdcard

Decompressing the rootfs and copying it to the card is very simple:

```
$ sudo tar -zxf linaro-\
  saucy-server-*.tar.gz
$ sudo mv binary/* rootfs
```

#### 6. Building the kernel

This a guide on how to cross-compile the kernel for your board too.

First, decompress the toolchain:

```
$ tar -Jxf arm-unknown-\
  linux-gnueabi.tar.xz
```

We already have the kernel sources that we downloaded earlier.

```
$ cd odroid-3.8.y
$ export ARCH=arm
$ export CROSS_COMPILE=../\
 arm-unknown-linux-gnueabi/\
 bin/arm-unknown-linux-\
 gnueabi-
$ make ARCH=arm \
  odroidu_defconfig
```

The last line is for the U3, and if you are doing this for the X2, just replace it with odroidx2_defconfig.

Building the kernel will take a while depending on your machine.

```
$ make -j8
```

I use -j8 because my computer is a quad-core with hyperthreading, so 8 threads are available. You should configure the number to match your computer's processor.

### 7. Install the kernel and modules that we just built

First, install just the kernel image.

```
$ sudo cp arch/arm/boot/\
  zImage ../boot
```

Next, install the modules:

```
$ sudo make ARCH=arm \
INSTALL_MOD_PATH=../rootfs \
modules_install
$ cd ..
```

Once the modules are installed, the kernel is ready!

### 8. Create an initial Boot Script for the first boot

```
$ cd boot
$ cat << __EOF__ | \
  sudo tee boot.txt
setenv initrd_high "0xffffff"
setenv fdt_high "0xffffff"
setenv bootcmd "fatload mmc
0:1 0x40008000 zImage; bootm
0x40008000"
setenv boot-
args "console=tty1
console=ttySAC1,115200n8
root=/dev/mmcblk0p2 rootwait
rw mem=2047M"
boot
__EOF__
```

```
$ sudo mkimage -A arm -T \
 script -C none -n boot -d \
 ./boot.txt boot.scr
$ cd ..
```

This creates the boot.txt file, and the sudo mkimage line creates the boot.scr.

### 9. Unmount and clean-up

```
sudo umount boot
sudo umount rootfs
sync
```

## First Boot

Now, we are ready to do our first boot. Remove the card from your computer and connect to your board.

### 1. Configuring your network card.

```
$ cd /etc/network/\
  interfaces.d
cat << __EOF__ >> eth0
auto eth0
iface eth0 inet dhcp
__EOF__
$ reboot
```

### 2. Configuring FSTAB

```
$ mount -t devtmpfs \
  devtmpfs /dev
cat << __EOF__ >> /etc/fstab
UUID=e139ce78-9841-40fe-
8823-96a304a09859 / ext4
errors=remount-ro,noatime 0
1
/dev/mmcblk0p1 /media/boot
vfat defaults,rw,owner,flush,
umask=000 0 0
tmpfs /tmp tmpfs
nodev,nosuid,mode=1777 0 0
__EOF__
```

```
$ mkdir -p /media/boot
$ mount /media/boot
```

### 3. Running the kernel update script

```
$ apt-get install \
  u-boot-tools
$ wget builder.mdrjr.net/\
  tools/kernel-update.sh
$ chmod +x kernel-update.sh
$ sudo ./kernel-update.sh
```

Running this step is important to create a uinitrd as well add all the other boot.scr files for different monitors and resolutions.

Everything below this is just regular Linux usage that you can find on Google and Linux Forums, and is intended only for those who want a Graphical environment.

## Install Xubuntu

Before starting the downloading, make sure that you have at least 450MB of disk space available.

```
$ sudo apt-get install \
  xubuntu-desktop
```

### 1. Installing Mali Drivers

```
$ cd ~
$ mkdir mali
$ cd mali
```

### 2. Downloading the Mali dependencies

```
$ wget http://builder.mdrjr.
net/tools/mali.txz
$ wget http://malideveloper.
arm.com/downloads/drivers/
DX910/r3p2-01rel4/DX910-SW-
99003-r3p2-01rel4.tgz
$ apt-get build-dep xserver-
xorg-video-armsoc
$ apt-get install mesa-utils
mesa-utils-extra libgles2-
mesa-dev libgles2-mesa
libgles1-mesa-dev libgles1-
mesa libegl1-mesa libegl1-
mesa-dev
```

### 3. Installing Blobs and Headers

```
$ tar zxf DX910-SW-99003-\
  r3p2-01rel4.tgz
$ tar Jxf mali.txz
$ mv /usr/lib/arm-linux-\
  gnueabihf/mesa-egl ~
$ cp -aR blobs/* /usr/lib
$ cp -aR include/* \
  /usr/include
$ ldconfig
```

### 4. Building and Installing the XII Driver

```
$ cd DX910-SW-99003-r3p2-\
  01rel4/x11/xf86-video-\
  mali-0.0.1
$ ./autogen.sh
$ cd src
$ rm -rf compat-api.h
$ wget \
http://cgit.freedesktop.\
org/~cooperyuan/compat-api/\
plain/compat-api.h
$ cd ..
$ make -j4
$ make install
$ mv /usr/local/lib/xorg/\
  modules/drivers/mali* \
  /usr/lib/xorg/modules/\
  drivers
```

### 5. Configuring Xorg.conf to use Mali

```
$ cat << __EOF__ >> \
  /etc/X11/xorg.conf
Section "Device"
Identifier "Mali-Fbdev"
 Driver "mali"
 Option "fbdev"            "/
dev/fb1"
 Option "DRI2"
"true"
 Option "DRI2_PAGE_FLIP"
"true"
 Option "DRI2_WAIT_VSYNC"
"true"
 Option "UMP_CACHED"
"true"
 Option "UMP_LOCK"
"false"
EndSection

Section "Screen"
 Identifier "Mali-Screen"
 Device "Mali-Fbdev"
 DefaultDepth 24
EndSection

Section "DRI"
 Mode 0666
EndSection
__EOF__
```

### 6. Create a udev rule to change mali permission in order for a regular user to use it

```
$ cat << __EOF__ >> /etc/\
  udev/rules.d/10-mali.rules
KERNEL=="mali",SUBSYSTEM=="m
isc",MODE="0777"
KERNEL=="ump",SUBSYSTEM=="um
p",MODE="0777"
__EOF__
```

Congratulations.. You made it!

# *HOW TO INSTALL THE ORACLE JAVA DEVELOPMENT KIT (JDK) VERSION 8*

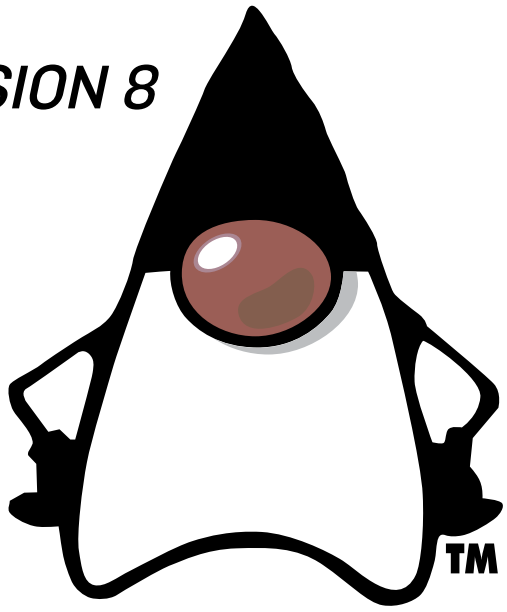
## SAVE TIME WITH JAVA'S "CODE ONCE, RUN ANYWHERE" ARCHITECTURE

**by Robert Raehm, Edited by Venkat Bommakanti**

Java is one of the most popular programming languages for both application and web development. It has the advantage of true cross-platform compatibility, which means that code written in Java will run on any Java Virtual Machine regardless of the processor, computer, operating system, or other hardware. Oracle publishes a free Development Kit, which is also available as an ARMHF binary, which means that the ODROID family can easily run the vast library of Java software. The latest version available as of April 2014 is JDK8, which can be installed alongside previous versions of Java, and provides a rich platform for development, including significant speed improvements over previous versions.

### Requirements

- **An ODROID from the X, U or XU series**
- **An 8+ GB eMMC or Class 10+ MicroSD**
- **A custom Ubuntu, Debian or similar image (13.04 or higher), available from the ODROID Forums (http://forum.odroid.com)**




**Even though this issue is designed to highlight several DIY projects, you could also call it the "Cute Mascot" edition!**

### Download the tarball

To begin, backup your personal files from your Ubuntu installation if necessary. On the Ubuntu desktop, create a folder to receive the downloaded package.

The examples in this article use the March 13th, 2014 version of Oracle JDK. You can download the latest version by visiting the Oracle website at `https://jdk8.java.net/download.html` by clicking the link for the most recent package labelled Linux ARMv6/7 VFP, HardFP ABI. As the time of this writing, the latest version available was `jdk-8-fcs-b132-linux-arm-vfp-hflt-03_mar_2014.tar.gz`.

After agreeing to the Terms and Conditions and downloading the file, it is good practice to check the md5-checksum of the package to make sure that it was transferred correctly. This is done by using the md5sum utility:

```
$ md5sum jdk-8-fcs-b132-
linux-arm-vfp-hflt-03_
mar_2014.tar.gz
```

The result should be compared to the contents of the checksum file located in the same directory as the package download. For this example, the md5sum file was located at `http://www.java.net/download/jdk8/archive/b132/binaries/jdk-8-fcs-b132-linux-arm-vfp-hflt-03_mar_2014.md5`.

My downloaded file had the checksum of c17b5194214b8ea9ad8e6fc302fe078. If the file that you downloaded has a different checksum than the one located on the server, discard it, restart the download and compare the checksums again.

## Unpack the tarball

In the terminal window, change directories (cd) to the designated download folder and unpack the file:

```
$ tar -zxvf jdk-8-fcs-
b132-linux-arm-vfp-hflt-03_
mar_2014.tar.gz
```

This creates a new subdirectory called jdk1.8.0 in the download directory.

## Mount the Java installation

On Linux systems, Java is typically installed in the system directory at `/usr/lib/jvm`

when using an automatic installer. However, since we are manually installing the package, the uncompressed files will need to be moved to the correct directory from the Terminal window.

```
$ sudo mv jdk1.8.0 \
/usr/lib/jvm
```

## Update the PATH environment variable

Your original Linux installation may have come with a prepackaged version of the Java Development Kit, and the location of that installation will most likely be specified in the PATH environment variable. The PATH variable specifies certain directories to search when a command is typed into the Terminal window, so that packages may be invoked from any directory.

After installing JDK 1.8.0 with the above steps, we need to ensure that the 1.8 version is used as the default virtual machine going forward. To do so, update the PATH environment variable to include the new version:

```
$ export PATH=/usr/lib/jvm/
jdk1.8.0/bin:$PATH
```

The $PATH at the end of the command appends the current PATH environment variable to the new one. Since the $PATH string is searched for the first occurrence of a program, once a match is found, the system ignores the rest of the $PATH string, thereby bypassing any previous Java installs that may also be included.

## Complete the installation

Typically, when programs are installed in Linux using installation utilities, certain symbolic links are created. We will need to manually update those symlinks using the following 4 commands:

```
sudo update-alternatives
--install /usr/bin/javac\
javac /usr/lib/jvm/jdk1.8.0/
bin/javac 1

sudo update-alternatives
--install /usr/bin/java\
java /usr/lib/jvm/jdk1.8.0/
bin/java 1

sudo update-alternatives
--config javac

sudo update-alternatives
--config java
```

## Verify the installation

As a final step, we need to ensure that JDK8 was installed properly, and that the appropriate components are being used. To do so, run the java binary using the version parameter to report the current default version:

```
$ java -version
```

The output should look similar to this, indicating that JDK8 is the default:

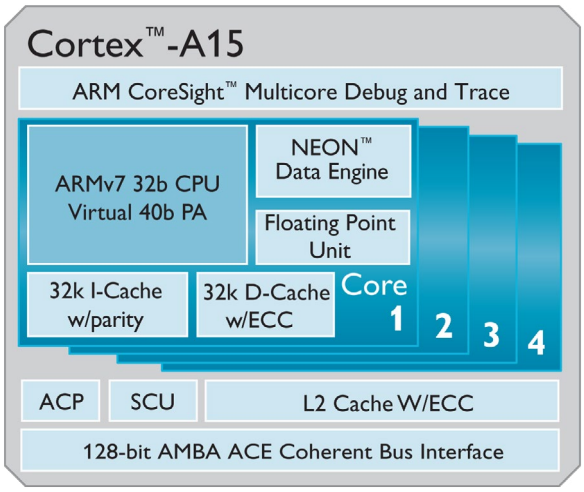
```
java version "1.8.0"
Java(TM) SE Runtime Environ-
ment (build 1.8.0-b132)
Java HotSpot(TM) 32-Bit
Server VM (build 25.0-b70,
mixed mode)
```

For additional information or questions, please visit the original forum thread at `http://forum.odroid.com/viewtopic.php?f=52&t=204`.

# USING ODROIDS IN HIGH PERFORMANCE COMPUTING

## WHAT A DIFFERENCE A KERNEL MAKES

by Kurt Keville, MIT




**The A15 is designed with advanced power reduction techniques, and powers our flagship XU, so get the most of it!**

We have been comparing different kernels and their respective performance on the XU (see `http://tinyurl.com/XUBench1` and `http://tinyurl.com/XUBench2`). It was interesting to see the differences between the kernels you get with Rob Roy's Particle and Whisper images (3.4.67 was the last update we did before we ran the benchmarks) versus the 3.13 kernel you get with the experimental Linaro 14.02 distro.

For those of you who worked with the Calxeda Highbank or Midway architectures, which are reflected in the openbenchmarking URLs referenced above and at `http://tinyurl.com/ApacheOnARM`, you will not be surprised at the Quad-Core ARM Cortex-A9 performance when delivering pages via httpd. Indeed, most traditional ISPs don't need to do much math to serve up lots of web pages, so an A9 class processor with a shorter NEON extension than the Cortex-A15 should work just fine.

It is interesting to see how the XU outperforms the quad-core A9 using the Apache benchmark, and also surprising how much better the 3.13 kernel is on the same benchmark as the 3.4 kernel on the Exynos 5410. The XU wins the race, likely because the A15 cores were being fully utilized and the A7 cores were quiesced, giving us the ideal power vs. performance ratio for that benchmark.

Future ARM clusters like SpiNNaker will have hundreds of thousands of cores, so every minor power efficiency improvement will be important. Many performance improvements also quite virtuously represent power reductions. For instance, if you remove much of your local media, in the form of SD cards or SATA drives, you can use the various tricks associated with tftp or PXE booting and ramdisks to speed up operations and reduce the list of devices that you are powering. Netbooting and NFSroot are high on the list of power reduction techniques.

The path to energy-efficiency in the ODROID-centric Datacenter can be facilitated through simple kernel and user space fixes. They don't make a great difference on their own but they add up. These are some of the more fruitful examples.

**The operation not performed is the most energy-efficient.**

In application code you can take full advantage of the capabilities of your chip. Using a fused multiply-add gets you those 2 operations for the same clock cycles of running those operations separately.

**Implement HPC maintainer / user behavior modifications.**

By this we mean queueing. If you use something like PowerNap or PowerWake, you can save considerable power over the lifetime of your gear. This functionality was described in my article in Issue 2 (February 2014) of ODROID Magazine.

**Categorize and maximize things that lend themselves to consolidation and distribution to leverage hybrid architectures.**

Put your writeable directories on the NFS shares so you don't need journaling filesystems or checkpointing on your (read-only) directories on the client nodes. It saves time and energy.

**Find a way to effectively utilize idle cycles for computation.**

We used a profiling tool to calculate the ideal communication vs. computation overlap strategy to grab the appropriate amount of data for an operation, so that we never get into

a data-starved or CPU-starved situation. If a data-starved environment is unavoidable we can go to a lower ACPI power state to dial to power back while we are waiting for the transfer to complete.

**Compile code locally to maximize resource usage.**

The package GCC 4.8 on the XU seems to give us the best, smallest binary.

**Use the most numerically efficient approach.**

Here again, this has most to do with application code, since you can often represent your floating point numbers in a number of levels of precision.

**Give the big problems their due emphasis, but also solve the lots of little problems.**

There are quite a few little fixes that we recommend. It will come as no surprise to the ODROID kernel hackers out there, that there is considerable flexibility in what resources you can exercise and emphasize in your production application, and in what you can turn off in the kernel with little to no adverse effect.

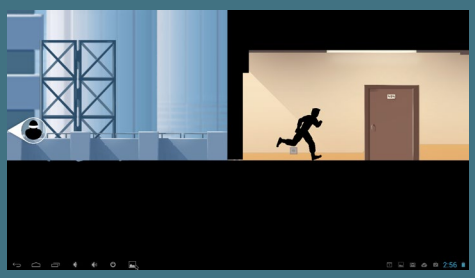
## Conclusion

To get the best performance out of your ODROID, you can turn on exactly what you want and turn off everything else. Make sure you are just running the single app you want to run (in our case a benchmark). You can't use 100% of your processor in your production app if you are busy responding to interrupts, so kill off (or don't start) unnecessary daemons.

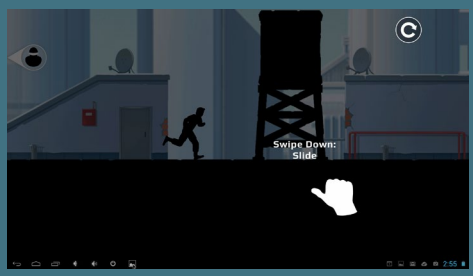
You can drop down to single-user mode (init 2) if you wanted to be sure you were not losing resources to unwanted apps, including anything you didn't turn off in the kernel, like USB and video. There are a few additional tips and tricks, such as the tickless kernel described at `http://tinyurl.com/XULess-Watts`. Enjoy the journey!

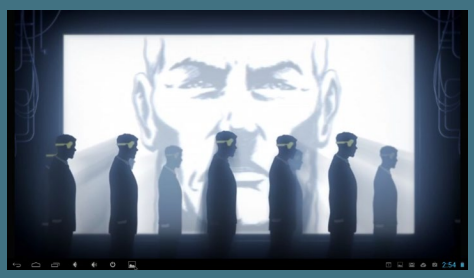
# VECTOR
## PARKOUR PACKED ACTION

**by Ronaldo Andrade**



Vector is an exciting, arcade-style game featuring you as the exceptional free runner who won't be held down by the system. The game opens with a view into a totalitarian world where freedom and individually is nothing more than a distant dream. But the heart of a freerunner is strong, and you soon break free. Run, vault, slide and climb using extraordinary techniques based on the urban ninja sport of Parkour all while being chased by "Big Brother" whose sole purpose is to capture you and bring you back.



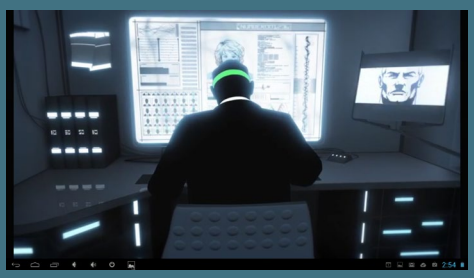



Inspired by the practice and principles of Parkour, Vector's intuitive controls accommodate players of all levels, and sophisticated level designs challenge the most demanding players with fast-paced timing puzzles as the traceur "flows" over the dystopian rooftops.

Overall, this is an awesome game from Nekki, a Russian development company. If you ever watched Parkour and found it interesting, you will love this game. The action is fluid and the commands simple, making it fun to play. But don't let these words fool you, the challenge the game presents is above average. There are three different stages which you can play on the full version, each more beautifully constructed and challenging than the last.

The main objective here is to escape from the guards that are after you, but in order to get three stars, you will have to collect holo-cubes and perform every trick, there is also some bonus money scattered through the levels, but they are not necessary to get 3 stars.

On the ODROID, you can use your keyboard and your mouse to control the player, or a joystick as well.



### GENERAL TIPS:

**Collect every cube and perform every trick -- this will grant you three stars at the end of the level.**

**Look ahead, check the scenario to see what stunt you will need to pass the obstacle using the least effort, consuming less time.**

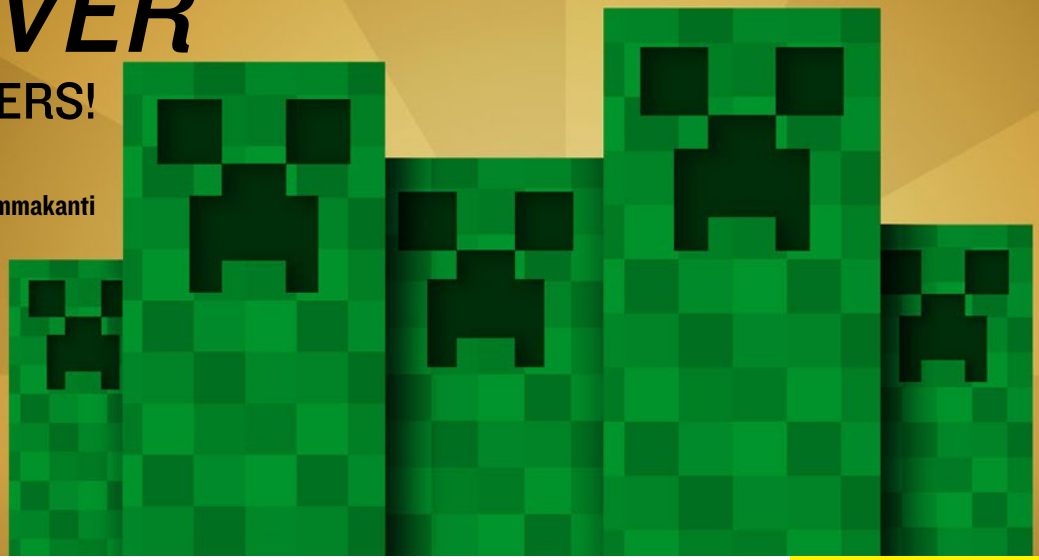
**At the store, you can use the money you receive after each stage to buy gadgets that can be useful in some situations.**

# HOW TO SETUP A MINECRAFT SERVER

## CREEEEPERS!

by @qkpham

Edited by Venkat Bommakanti



Almost everyone loves playing games, especially Minecraft! It's been enjoyed by over 14 million people worldwide for its addictive gameplay and customizable maps. Although the official package from Mojang Software is closed-source, several open-source Java versions of Minecraft Server are also available for the ODROID platform. Programming a virtual world using a free Minecraft Server package such as Spigot, Bukkit or BungeeCord is also a great way to learn Java while having fun too!

This article details how to install a basic Minecraft server on your ODROID, so that you can play online games with a few of your friends in a world of your own creation. Using the ODROID as an inexpensive sandbox is also a great way to test out maps, upgrades and modifications before uploading them to a public server.

### Requirements

1. An ODROID from the X, U or XU series
2. An 8+ GB eMMC or Class 10+ MicroSD
3. A custom Ubuntu, Debian or similar image (13.04 or higher), available from the ODROID Forums (`http://forum.odroid.com`)
4. Java version 1.8 (OpenJDK8 or Oracle JDK8)
5. Local Area Network (LAN) connection, including a router with port-forwarding feature

### Install Java

If Java version 1.8 isn't already installed on your system, please refer to the article in this issue of ODROID Magazine called Installing Oracle JDK8. Mojang publishes a Java version of the Minecraft software for compatibility with other operating systems such ARM Linux.

### Install Minecraft

First, download the latest Minecraft Server software from the official site at `https://minecraft.net/download`, making sure to get the Java-based .tar version.

Create a minecraft directory in your home directory for storing the downloaded minecraft_server.jar. Once the tarball is downloaded, type the following commands to start the server:

**When your hear creepers making the Sssss... noise, there's only one thing to do:**

**RUN!**

```
$ cd ~/minecraft
$ java -Xms1536M -Xmx1536M
-jar minecraft_server.jar
nogui
```

The Minecraft server should be up and running now! The final step is to get the server's IP address so that our players can connect to it via their Minecraft clients.

### Obtain the internal IP address

Find out the internal (local) IP address of your server by typing ifconfig in the Terminal window and locating the tag inet addr. On my ODROID, the IP address was listed as 192.168.1.10. Make sure this address has a long lease issued by the local DHCP server or router in order to avoid frequent configuration updates.

## Setup port forwarding

Minecraft uses the TCP port 25565, which should be forwarded to the server's IP address by your local router using port forwarding. Refer to the user manual for assistance with setting up the router to forward port 25565 to the IP address obtained in the previous step.

## Obtain the external IP address

The public IP address that identifies your LAN to the outside world can be discovered by visiting `http://www.whatismyip.com`. The address will be in the form `aaa.bbb.ccc.ddd`, which means that the fully-qualified URL for connecting to the Minecraft Server on your LAN would be `http://aaa.bbb.ccc.ddd:25565`. Note the additionof the relevant TCP port at the and of the URL.

If your external IP is dynamic (typically changed periodically by your ISP), you can use services like No-IP. You can create an account on their website, then download and install the Dynamic DNS Update Client (DUC) at `http://www.noip.com/download`. Detailed instructions on setting up Dynamic DNS can be found at `http://bit.ly/1ggmo2n`. In this case, the fully-qualified Minecraft Server address would be `http://youracctusername.no-ip.com:25565`.

To make sure everything's working, you can test that your server is visible online by going to `http://www.canyouseeme.org`. You can also quickly check its status at `http://dinnerbone.com/minecraft/tools/status/`.

System performance will be acceptable under normal wireless ethernet conditions, but a wired connection will decrease latency and increase game responsiveness.

## Joining the Game

Start your Minecraft client on a Windows or OSX machine by entering the public IP address from the previous step (`http://aaa.bbb.ccc.ddd:25565`) when adding a new server to the client's server list. At the time of this writing, the Minecraft Client software unfortunately does not yet run on the ODROID platform. There is a Minecraft Pocket Edition available for Android, but it is not compatible with the full version of Minecraft Server.


A successful connection to the ODROID Minecraft Server will bring the user into our virtual world as seen above.

## Additional Server Configuration

The server options in Minecraft are configured by editing the server.properties file located at `/home/yourusername/minecraft/server.properties`:

```
#Minecraft server properties
#Mon Dec 24 09:23:18 EST
2012
#
generator-settings=
level-name=world
enable-query=false
allow-flight=false
server-port=25565
level-type=DEFAULT
enable-rcon=false
level-seed=
server-ip=
max-build-height=256
spawn-npcs=true
white-list=false
spawn-animals=true
hardcore=false
texture-pack=
online-mode=true
pvp=true
difficulty=1
gamemode=0
max-players=20
spawn-monsters=true
generate-structures=true
view-distance=10
motd=A Minecraft Server
```

The three settings useful in changing maps and improving performance include:

**level-name**

If you want to add another map or world to your server, just unpack the world file inside your minecraft folder and then change the level-name setting to the name of that folder. For example, if your extracted world folder is odroid then change the level-name value to odroid instead of the default world value.

**view-distance**

Can be reduced to 7 to improve server responsiveness

**max-players**

Performs best when set between 2 and 5

# DOWNLOAD YOUTUBE VIDEOS TO WATCH OFFLINE

by Bruno Doiche

We are now leaving in a connected world, but from time to time, we need to go to places where there is no kind of network connectivity. Well, pack up a survival kit with whichever you like from youtube with youtube-dl!

To install, just type the following at the terminal:

```
sudo pip install --upgrade youtube_dl
```

Now you can download any video that you want from youtube, just do:

```
youtube_dl <youtubevideo_url>
```

What you say? you just want the music from the videos and the audio from the podcasts and want to save space?

Ok, let's create a simple script to solve this then

```
echo "ffmpeg -i $1 -acodec libmp3lame -ac 2 -ab 128 -vn -y $2" > mp3zator.sh
```

Turn it to an executable with:

```
chmod + X mp3zator.sh
```

And execute it like this:

mp3zator <your_video_.mp4> <your_audio.mp3>

Alright, get all you need and get lost without fear of not having your beloved movies, videos and music to consume while you code in a far far away land.





Please note that Minecraft relies heavily on floating point operations. Unlike x86 architecture based CPUs, ARM based SOCs are not optimized for floating point operations, so the server options need to be tuned down to compensate for the heavier load.

If you'd like to further improve performance, several open-source versions of Minecraft Server are available that significantly decrease the server's computations, providing a smoother experience and allowing more players to join the game.

## Craftbukkit

Create a folder for Craftbukkit by typing `mkdir ~/craftbukkit` in a Terminal window, then visit `https://dl.bukkit.org/downloads/craftbukkit/` to download the latest version of Craftbukkit to the newly created directory. Once the download has completed, run the server to build your world.

```
java -Xms1536M -Xmx1536M
-jar craftbukkit.jar
cd ~/craftbukkit/plugins
wget http://dev.bukkit.org/
media/files/674/323/NoLagg.
jar
wget http://dev.bukkit.org/
media/files/665/783/PTweaks.
jar
wget http://dev.bukkit.org/
media/files/586/974/NoSpawnC-
hunks.jar
```

## Spigot

An alternative to Craftbukkit is Spigot, which provides more configuration options and is optimized for performance and speed. Following the same procedure as listed above, downloading the Spigot package instead, found at `http://www.spigotmc.org`.

```
mkdir ~/spigot
cd spigot
wget http://ci.md-5.net/job/
Spigot/lastSuccessfulBuild/
artifact/Spigot/target/spig-
ot.jar
java -Xms1536M -Xmx1536M
-jar spigot.jar
```

Spigot is very stable, and since it is based on Craftbukkit, the Bukkit plugins NoLagg, PTweaks and NoSpawnChunks above will also work with Spigot.

## MineOS

MineOS is a Web-based administrative panel that offers easy management of Minecraft servers. It can handle Vanilla, Bukkit, Tekkit and Canary by default, but you can install any other server system and configure it to automatically download a new version whenever available.

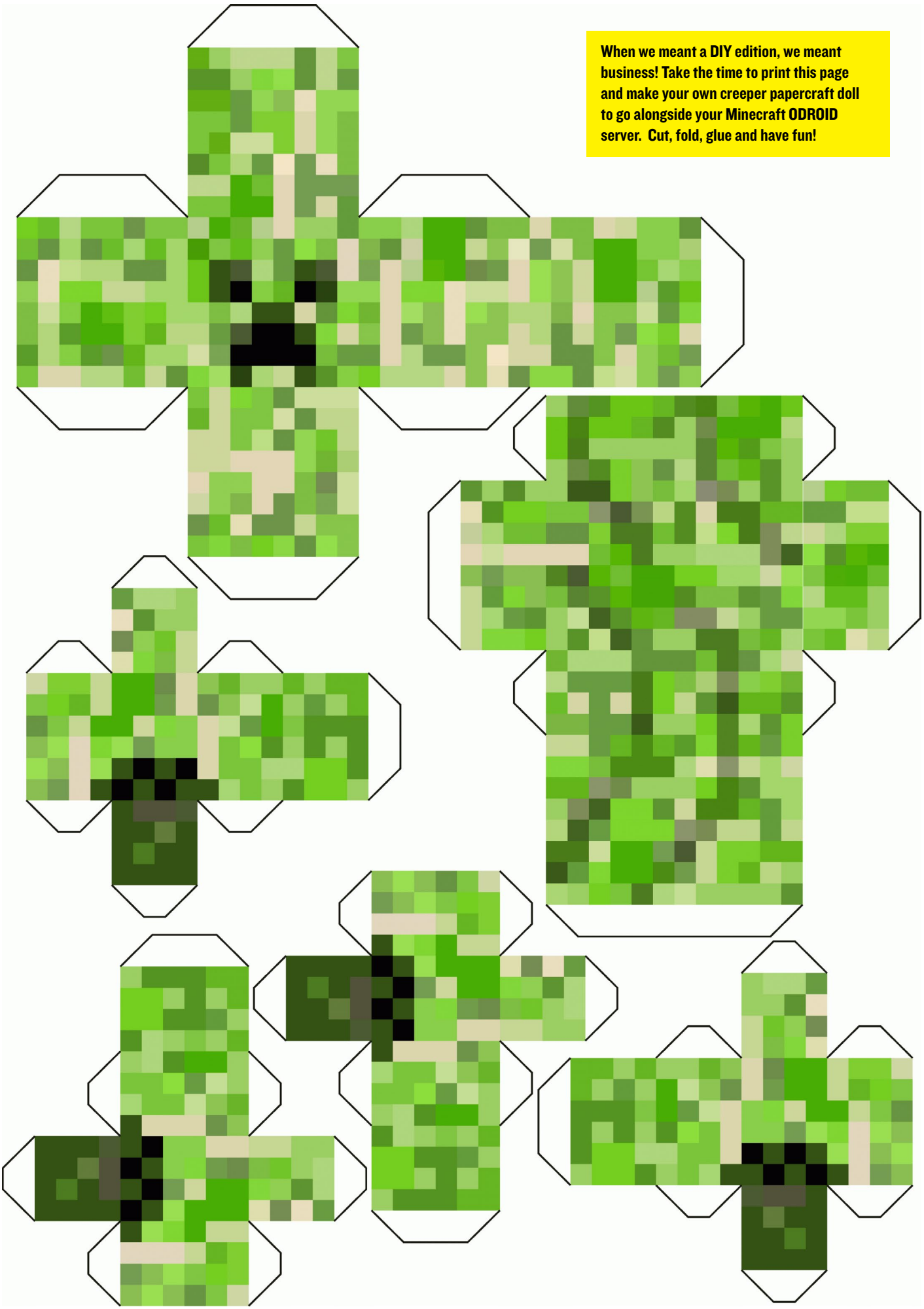
## Copying your server to an external hosting service

Using an open-source version of Minecraft allows you to change any aspect of the server, including fixing bugs and installing addons. Since Minecraft for ODROID is written in Java, it's easy for beginners and experts alike to improve the software and customize it to their own needs.

Once you have your world ready, you can migrate your Minecraft creation to a high-traffic server so that it can accommodate more players. Simply upload all of the server files from the minecraft, spigot or craftbukkit directory on the ODROID via the web hosting service's administration panel.

Enjoy your new ODROID Minecraft Server, and remember to stay out of the lava! For additional information or questions, please visit the original forum thread at `http://forum.odroid.com/viewtopic.php?f=52&t=84`.

When we meant a DIY edition, we meant business! Take the time to print this page and make your own creeper papercraft doll to go alongside your Minecraft ODROID server. Cut, fold, glue and have fun!



# LEARN REBOL

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

By Nick Antonaccio and Bohdan Lechnowsky

In the first installment of Learn Rebol, we discussed the motivation behind Rebol and learned how easy it is to create a GUI-based program in Rebol on Android. We expanded on these examples in last month's issue. This month, we delve even deeper into what can be done with Rebol3 on ODROID and other platforms.

In this installment, we'll list the web addresses of where to get the most up-to-date version of Rebol for different platforms. The non-ARM binaries are listed so you can try your Rebol 3 programs on your laptop and desktop computers as well (note, not all Rebol 3 binaries have the graphical component available yet).

It's also my pleasure to announce that the current Rebol 3 builds for Linux ARM hard-float are being compiled and tested on ODROID computers!

And remember, you can run any app you create in Rebol 3 for ODROID 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 the ARM version (currently titled "**Linux (ARM v7 with hardware floating point support) Great for ODROID!**") from `http://atronixengineering.com/downloads.html`.

Perform the following commands in the terminal emulator in the directory where you downloaded r3:

```
sudo mv r3-armv7hf-view-linux r3
sudo chmod +x r3
sudo ./r3
```

**Windows (x86), Linux (x86), OSX (x86):**

`http://atronixengineering.com/downloads.html` or `http://rebolsource.net` *

**Windows (x64), Linux (x64):**

`http://atronixengineering.com/downloads.html`

**OSX (PPC), Haiku (x86), Linux ARM (soft-float), Linux (IA64), OSX (x64):**

`http://rebolsource.net` *

(* These builds do not contain the graphical components yet)

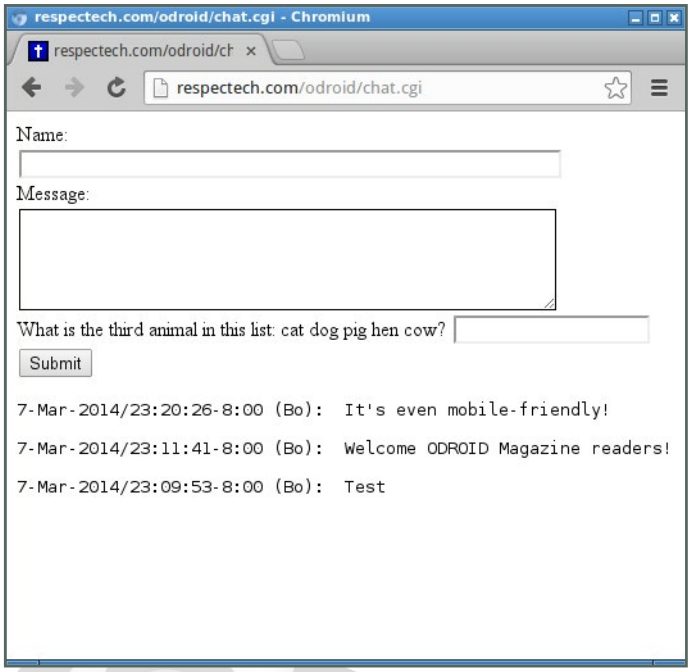
## Writing More Programs in Rebol

The focus of these examples is not to teach programming in Rebol, but rather to show how much is possible with how little. For further learning resources, see the end of this article.

Here's a little web chat app running at `http://respectech.com/odroid/chat.cgi`, complete with a simple verification system to make it harder for the spambots to post. The verification system uses a feature of Rebol where data and code are interchangeable. This makes doing things like a verification system much simpler:

```
#!./rebol3 -cs
REBOL [title: "Group Chat"]

;The following line is required as the first
line in cgi output
print {content-type: text/html^/}

;Define where the chat messages are stored
url: %./chat.txt
```

```
;Initialize the username
username: copy ""

;Read the POST string to see if there is data
to be processed
if attempt [
    submitted: parse (to string! read system/
ports/input) "&="
][
    ;Only process the following lines if POST
data was submitted

    ;In POST data, spaces are replaced by "+",
so change them back to
    ; spaces
    foreach item submitted [replace/all item
"+" " "]

    ;If there was some data to process and the
verification question was
    ; correctly answered, add the message to
the end of the chat file
    if all [
        submitted/2 <> none

        ;The "load" statement takes the ordi-
nal value picked at random
        ; (e.g. The word "first") and converts
it to a Rebol word.
        ; The "do" statement tells Rebol to
evaluate what follows it,
        ; in the case of this example, the
command "first", which picks
        ; the first item out of a series.
        submitted/6 = do load submitted/5
parse "cat dog pig hen cow" ""
    ][
        write/append url mold rejoin [
            now " (" submitted/2 "):  "
submitted/4 "^/^/"
        ]
        username: submitted/2
    ]
]

;Convert the chat file into plain text, includ-
ing any new message that was
; just added above.  Display it in reverse or-
der so the newest messages
; stay on top, right after the input section.
notes: head reverse load dehex copy read/
string url

;Generate the pivotal part of the verification
question
random/seed now/time/precise
ordinal: to-string pick [first second third
fourth fifth] random 5

;Output the HTML page
print rejoin [
    {<FORM METHOD="POST">
        Name:<br>
        <input type=text size="65"
name="username" value="} username {"><br>
        Message:<br>
        <textarea name=messwage rows=5
cols=50></textarea><br>
        What is the } ordinal { animal in this
list: cat dog pig hen cow?
        <br><input type=text name="} ordinal
{"><br>
        <input type="submit" name="submit"
value="Submit">
    </FORM>}
    "<pre>" notes "</pre>"
]
```




Important Note: In order to allow more efficient execution of the examples from now on, we are going to download the r3-gui.r3 graphic dialect definition to the local storage of your device instead of downloading it each time. We can do this from within Rebol itself. On your device, simply type the following:

```
write %r3-gui.r3 read/string
http://www.atronixengineering.com/r3/r3-gui.r3
```

If you get an error when running any of the example scripts below on your device, try this instead:

```
write %r3-gui.r3 read/string http://
development.saphirion.com/resources/r3-gui.r3
```

Rebol 3 is open source, and there are several groups working on enhancements. This leads to having different versions for different devices in slightly different states at any given time. This will solidify and these issues will go away as time moves on.

Doing the above will speed up execution greatly as the r3-gui dialect doesn't need to be downloaded each time. However, on most non-rooted Android tablets and phones, superuser access is not allowed, so you won't be able to write to the root directory and the above command will fail. This shouldn't be a problem on your ODROID running Android. In this case, either continue to use load-gui or write r3-gui.r3 to another location, like the sdcard, with a command like this:

```
write %/sdcard/r3-gui.r3 read/string
http://.../r3-gui.r3
```

(Replace the "…" with one of the URL paths from the examples above.)

I've modified the examples on the website to check for r3-gui.r3 in the current directory and the root of the sdcard, and if it doesn't exist in either location, then it uses load-gui. I did this by replacing the load-gui in the following examples with this code:

```
foreach cmd [[do %r3-gui.r3][do %/sdcard/r3-
gui.r3][load-gui]][
    if attempt [do probe cmd][break]
]
```

Basically, there are three different ways to load the r3-gui dialect specified, and it tries each one until one works without error.

To run the examples off the website instead of typing them in, just type:

```
do http://respectech.com/odroid/learnrebol/
file.r
```

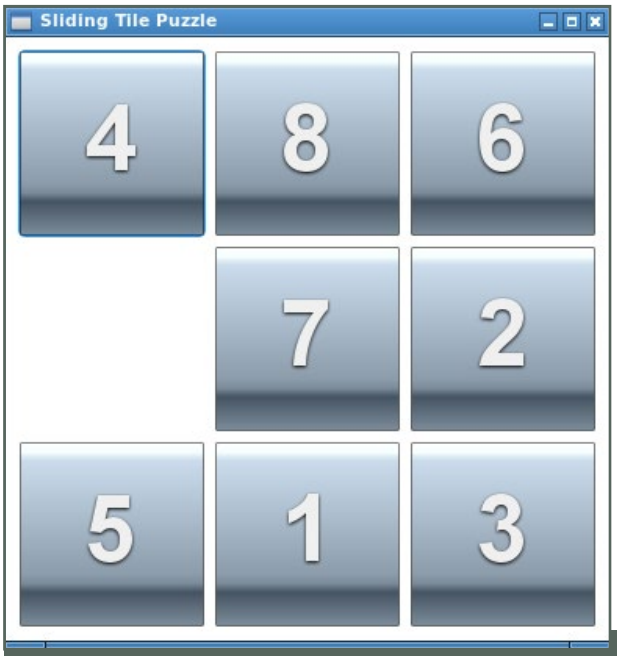
Replace file.r with the filename in the Rebol header (leave off the "%" though).

Here's a small graphic sliding tile game, and no complex GUI builder tool was required to create this code. It's simple and readable enough that a text editor and the built in help facilities of Rebol are all you need. The actual layout code is 5 lines. Have you ever seen code this simple used to create a game for Android (or even a desktop machine)? No IDE, SDK or build scripts are needed either - just download the small R3 interpreter to your Android device or your PC, click the plain text code file, and it runs the same on every platform, with graphics, touch events and all, without any changes to the code:

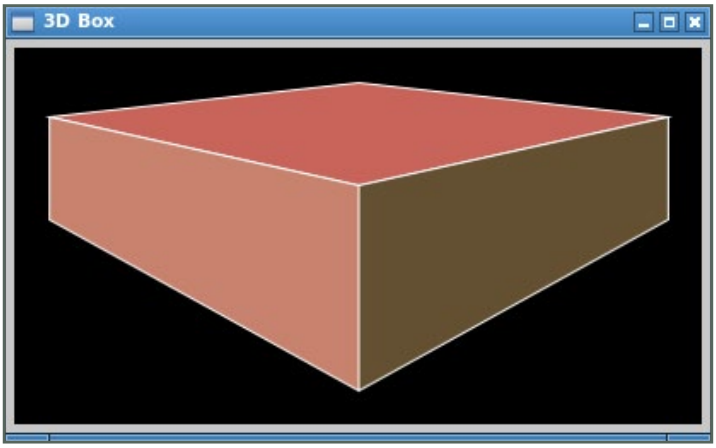
```
REBOL [title: "Sliding Tile Puzzle" file:
%sliding-tile-game.r]
load-gui
sz: 120x120
fontize [
    p: button [font: [size: 60]]
]
stylize [
    p: button [
        facets: [text-style: 'p init-size: sz
max-size: sz]
        actors: [
            on-action: [
                t: reduce [face/gob/offset x/
gob/offset]
                face/gob/offset: t/2  x/gob/
offset: t/1
            ]
        ]
    ]
]
view/options [
    hgroup [
        p "8"   p "7"   p "6"   return
        p "5"   p "4"   p "3"   return
        p "2"   p "1"   x: box sz white
    ]
] [bg-color: white]
```

While on the topic of games, it should be noted that R3 allows you to draw graphics and create animations very easily. Here's a quick example:

```
REBOL [title: "3D Box" file: %3d-box.r]
load-gui
bck: make image! 400x220
view/no-wait [image bck]
draw bck to-draw [
    fill-pen 200.100.90
    polygon 20x40 200x20 380x40 200x80
```

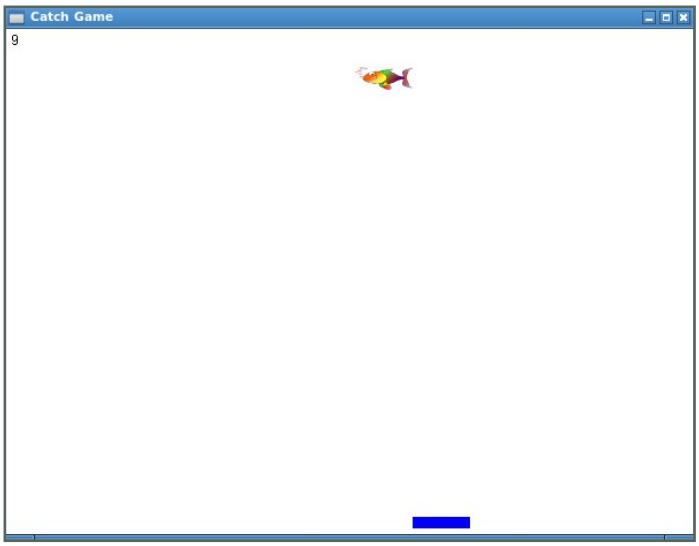



```
    fill-pen 200.130.110
    polygon 20x40 200x80 200x200 20x100
    fill-pen 100.80.50
    polygon 200x80 380x40 380x100 200x200
] copy []
do-events
```




Here's a complete arcade game with image animation, collision detection, keyboard event controls, score keeping, and more. Try to catch the falling fish. Be careful, it gets faster as you go!

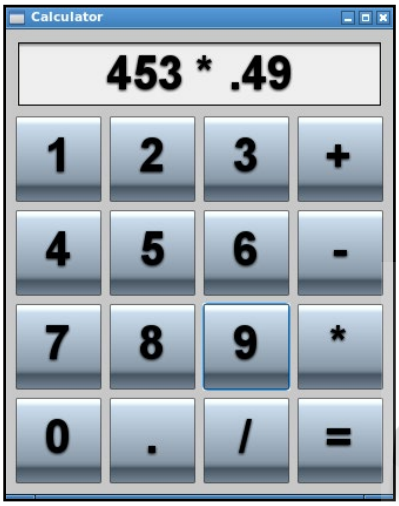
```
REBOL [title: "Catch Game" file: %catch-game.r]
load-gui
fish: load http://learnrebol.com/r3book/fish2.
png
s: 0  p: 3  random/seed now/time
stylize [
    paddle: box [facets: [max-size: 50x10]]
    img: image [facets: [max-size: 50x20 min-
size: 50x20]]
]
view/no-wait/options [
    t: text"ARROW KEYS" y: img 50x20 (fish) pad
z: paddle blue
    return
    arrow left 120x120 arrow right 120x120
] [
    shortcut-keys: [
        left  [z/gob/offset/1: z/gob/offset/1 -
50 draw-face z]
        right [z/gob/offset/1: z/gob/offset/1 +
50 draw-face z]
    ]
    min-hint: 600x440  bg-color: white
]
forever [
    wait .02
    y/gob/offset/2: y/gob/offset/2 + p draw-face
y show-now y
    if inside? y/gob/offset (z/gob/offset -
49x0) (z/gob/offset + 49x10)[
        y/gob/offset: random 550x-20 s: s + 1
set-face t form s  p: p + .3
    ]
    if y/gob/offset/2 > 425 [alert join "Score:
" s unview unview break]
]
```




Here's an R3 version of a program found in virtually every GUI instructional text - a basic calculator. Blink, and you'll miss the code for this one. There are no other files, layout templates, initialization scripts, or tools required to run this app on any platform. This is the entire, completely portable program. As you can imagine, with so little code, there's a short learning curve to fully understand how examples like this work. Compare this code to C++ (`http://afsalashya-`

na.blogspot.com/2012/06/gui-simple-calculator-visual-c-source.html), Visual Basic (http://archive.msdn.microsoft.com/spektrum1calculator), or even the simplest possible RFO Basic example (http://rfobasic.com/#section-12.2). That last example was written by the author of this text to demonstrate the nearest comparably easy and productive Android development tool available - and each of those examples runs only on a single operating system. Here's a minimal HTML5 example (http://thecodeplayer.com/walkthrough/javascript-css3-calculator). It requires multiple pages of HTML, CSS and Javascript code. All those examples just scratch the surface of complexities found in other development environments:

```
REBOL [title: "Calculator" file: %calc.r]
load-gui
sz: 100x100
fontize [btn: button [font: [size: 60 color:
black]]]
stylize [
    btn: button [
        facets: [text-style: 'btn init-size:
sz max-size: sz]
        actors: [on-action:[set-face f join
get-face f get-face face]]
    ]
    field: field [
        facets: [text-style: 'btn init-size:
415x60 max-size: 415x60]
    ]
]
view [
    hgroup [
        f: field return
        btn "1"  btn "2"  btn "3"  btn " + "
return
        btn "4"  btn "5"  btn "6"  btn " - "
return
        btn "7"  btn "8"  btn "9"  btn " * "
return
        btn "0"  btn "."  btn " / "   btn "="
on-action [
            attempt [set-face f form do get-
face f]
        ]
    ]
]
```




## Resources

### Online Chat and Help:

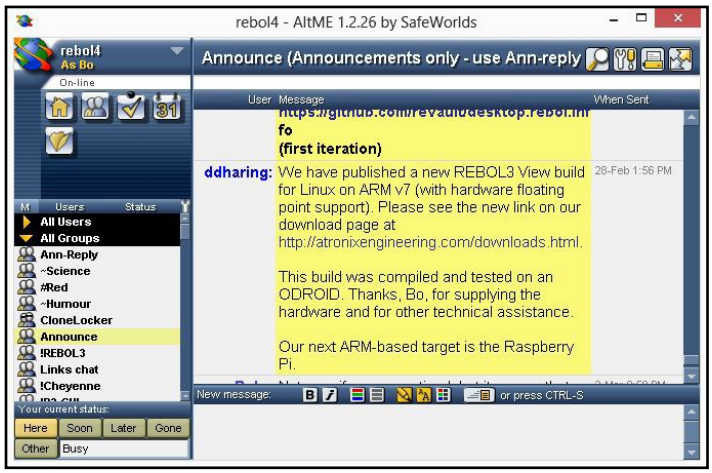
### StackOverflow.com:

There are currently over 1100 questions (and answers) related to Rebol on StackOverflow.com (http://stackoverflow.com/search?q=rebol).

20 Points are required to chat on StackOverflow.com (http://chat.stackoverflow.com/rooms/291/rebol-and-red). If you don't have 20 points (or an account at all for that matter), come on by anyway and look up [Rebol and Red] under the chat rooms. We are usually one of the most active. We'll help you get the 20 points you need to chat.

### AltME:

To join the Rebol-powered AltME world, send an email to user bo at the domain respectech.com asking to be invited. We are a closed community to avoid spam. Don't be shy, we've been called the friendliest software development community




on the planet.

### Facebook:

https://www.facebook.com/groups/rebol/

# BE HEARD WITH ÜBERCASTER

## A REAL-TIME AUDIO BROADCASTER HOTSPOT

by K.J Yoo of Echos Design (www.echosdesign.com)

The Ubercaster, build on the ODROID platform, is a 21st century way to listen to live music through your smartphone without having to push up to the front of the crowd.

The year was 2010. On the streets of the altstadt in Marburg, Germany, I was playing the violin as street musician. Some found my music distracting and yet some found it beautiful. As a curious engineering student, I thought about a better medium to present my music so that only those who were interested may hear what I played seamlessly. After realizing FM transmitter systems were quite expensive, bulky, old and simply not practical. I decided to take matters into my own hands. The solution was simple: Broadcast audio to people's favorite device: the smartphone.

### Design Goal

I wanted anyone to easily plug in any audio into the Übercaster; whether it came from an instrument, TV, iPod or microphone, it didn't matter. The Übercaster starts broadcasting the sound locally. Then multiple listeners would use their smartphone devices to connect to the Übercaster like a wifi hotspot to "tune-in." I also wanted the Übercaster to be an elegant and intuitive device in-line with Dieter Ram's 10 Principles of Good Design.

### Development

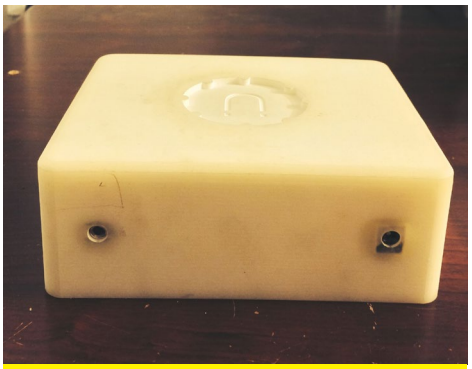
I have been developing the Übercaster with ODROID X2/U2/U3 development boards since August 2013, and it consists of device and client mobile apps.

Essentially, the Übercaster device is an ODROID U3 running Hostap. (For those of you who are not familiar with Hostap, check out Mauro Ribeiro's article from the February issue "Using an ODROID-XU as a WiFI Router".) The device is running Ubuntu 13.06 with a custom ODROID-3.8.y kernel. The Übercaster application captures audio with ALSA, encodes the captured audio with OPUS (`http://www.opus-codec.org`) and then packetizes the raw OPUS packets for UDP-based multicasting. This process takes on average 8ms and requires about 6-9% of the CPU. I will admit the ODROID U3 might be overkill for what I am doing, but I was not able to find a small dev board with a high quality audio codec.

So the ODROID works perfectly, and kudos to Hardkernel!

Via Hostap, wifi capable devices such as smartphones, tablets and computers can connect to the Übercaster device, which is running isc-dhcp-server to handle all the clients. As soon as a connection is established, the Übercaster mobile client app can be used to listen to whatever the Übercaster device is broadcasting. The app listens to the broadcast IP address on the device, receives the packets, decodes it, and plays back the sound.

Now at a first glance, it seems like a basic streaming application like VLC or Icecast. However, Übercaster offers real-time performance. Real-time is relative and subjective depending on the applications, but for the Übercaster system, the goal is to have the total audio latency below 25ms. How I measure audio latency is the delay between the time audio goes into the



An early 3D printed prototype of the Ubercaster device, not to be confused with a headphone-ready bar of soap.

Übercaster device and when it plays back on an iPhone 5S. (iOS has a lower audio latency than Android devices.) 25ms audio latency is not an arbitrary number, but rather the supposed maximum audio latency before a person is able to perceive the delay. Currently, the audio latency is < 50ms on iOS devices and on Android devices it varies significantly from device to device. On the Google Nexus 7 (2013), the latency is 80ms. I have tested the Übercaster with multiple participants and even though the total latency is currently double of my goal and the latency varies between iOS and Android devices, 95% of the listeners were not able to perceive any delay when watching TV or a movie.

So how many clients can the Übercaster support? I have tested up to 25 clients. However, it is theoretically possible to have many more. After the server-client relationship is established, the Übercaster is basically a one-way system. The Übercaster broadcasts UDP-based packets and the clients merely tune in on an IP address. That is it. However, there is a trade off: UDP isn't always reliable. The trade off is that UDP delivers packets faster and more efficiently than TCP because it uses non-ack. This is why the Übercaster transmits using small packet frame sizes to hedge against high packet loss rate, which gives smoother playback.

## Demonstration Video

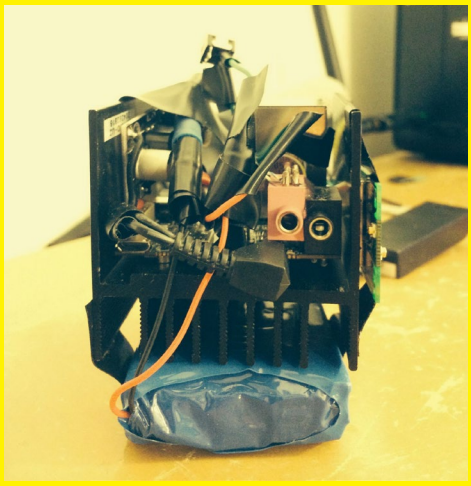
Please view the following demonstration of the Übercaster.

**Übercaster Zwei:**
`vimeo.com/85006122`
**Übercaster Drei:**
`vimeo.com/88467399`

## Dealing with Issues

1. To minimize frequency interference, I am mainly using 802.11n at the 5Ghz band. The 2.4Ghz never works even in a moderately crowded area. While using the 5Ghz band, the range is shorter and requires a bit more power, but it is very stable.

The Ubercaster has evolved into a sleek, sexy machine from its early tape-and-chewing-gum prototype.



Using a minimum of hardware, the Ubercaster delivers high-fidelity sound while consuming only 8W of power

So, at CES 2014 in Las Vegas, I had no problem giving a demonstration in the middle of the densely packed South Hall. (It will be very interesting to work with an 802.11ac module very soon!)

2. In order to reduce latency, I use OPUS, SPSC Circular Queue and a custom protocol that is based on UDP. I tried RTMP, RTSP and HTTP, but these really didn't work out for me. Originally, I wanted to use VLC or another RTSP client to stream content on client devices, but the latency was very high. This is why I chose to go with native apps, which are very light. I am currently creating an API that makes it very easy for mobile developers to integrate the Übercaster stream function. A quick tip concerning Android: it is important to match the sample rate and buffer size for minimum latency. Check out this interesting talk from Google I/O 2013 about High Performance Audio on Android: `https://www.youtube.com/watch?v=d3kfEeMZ65c`.

## The Application

Übercaster started with a simple question: how can individuals have complete freedom and seamless control of what they hear in a local surrounding? Or, how can individuals have complete freedom to seamlessly and easily broadcast sound to audience members in the local surrounding?

It turns out that places like gyms, restaurants, tour guides, music venues, sports bars and airports have been thinking of innovative ways to broadcast sound. There have been attempts at using FM and infrared, however it didn't prove to be practical, and is expensive and complicated to use.

From the very beginning, mobile was at the heart of the product. Currently 65% of all mobile phone users in the US use a smartphone. It is widely adopted and it is growing at a staggering rate. So everyone essentially has Übercaster-capable receiver devices already.

Imagine going into a sports bar and listening to any TV or tuning into the breaking news while you wait for the flight to Frankfurt or listening with perfect clarity to the street musician playing guitar 50 feet away or experiencing a tour of Rome through your smartphone.

Übercaster not only offers a richer and higher audio quality than current products, but it also makes an incredible seamless experience for both those transmitting and those listening. Übercaster simplifies, reduces and enhances local audio broadcasting into just a single device.

## The Vision

Sound is a stepping-stone for me to test if local public content distribution works. I want to broadcast video in real-time. I think of the future a lot, and it is clear that the frequency bands are getting crowded; people want more bandwidth and faster information. I think that in public spaces, there are too many data/bit redundancies. If a lot of people in a public area are interested in knowing more about something like the Real Madrid game, it is redundant for their devices to access information from the same server a thousand miles away in Texas or California. TVs in a public area are in essence a form of local broadcast. People within 50 feet see the TV. However I am not satisfied with how it works currently. So my goal is local distribution of content. Let's say someone sees a TV in the airport broadcasting CNN with a breaking news story. They should be able to have access to the sound at a minimum -- eventually real-time HD video streaming to their phone at a local distance and also additional web content relating to that news that is constantly aggregating on the Übercaster device for distribution. It is more efficient; people get information more quickly and seamlessly.

If you are interested in knowing more about the Übercaster or have interest in the technology, please email me at KJ@EchosDesign.com.

# ODROID U3 I²C COMMUNICATION

## INTER-INTEGRATED CIRCUITS FOR THE REST OF US

**by John Taylor**

After ordering my ODROID-U3 specifically for I2C communication with several slave devices, I was unable to find a comprehensive guide explaining the process of how to set everything up. In the interest of sharing with others what I've learned, I put together my own guide for setting up an I2C system on the ODROID platform.

The goal of this article is to introduce you to I2C communication using the ODROID-U3 as a master. We will communicate with an LED matrix from Adafruit. I initially planned to write this tutorial on communicating with an MSP430 microprocessor from Texas Instruments, which I have successfully set up. I decided, however, that the materials and additional programming needed for that project are beyond the scope of this article.

### Gathering the Equipment

- **ODROID-U3**
- **I2C LED Matrix** http://www.adafruit.com/products/1049
- **Level Shifter** http://www.adafruit.com/products/757?gclid=CINsJL057wCFURk7AodZkAArg

## Setting up the ODROID-U3

We need to install i2c-tools so that we can probe the I2C bus. This is easily done by running the following command in terminal, which will take a few minutes to install:

```
sudo apt-get install i2c-tools
```

Now that we have the i2c-tools package, we need to load the i2c-dev module so that we can use it. You can do this using with the modprobe command, but every time the ODROID is reset we will

have to reload the module. To avoid this annoyance, we will add i2c-dev to the list of modules that are loaded at startup. Open the `/etc/modules` file with your favorite text editor such as nano, and add i2c-dev to the list.

```
nano /etc/modules
```

```
 GNU nano 2.2.6                File: /etc/modules

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
i2c-dev

                              [ Read 5 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

**The /etc/modules file being edited using Nano.**

Once you have saved the file, reboot the ODROID and make sure that when you type the following command you get a similar result to Figure below.

```
i2cdetect -l -y
```

This command tells the computer to detect and list (`-l`) all of the I2C ports available. If you don't use the –y flag the computer will ask you if you are sure you want to perform this action and warn you of possible damages that can be done by messing with I2C busses.

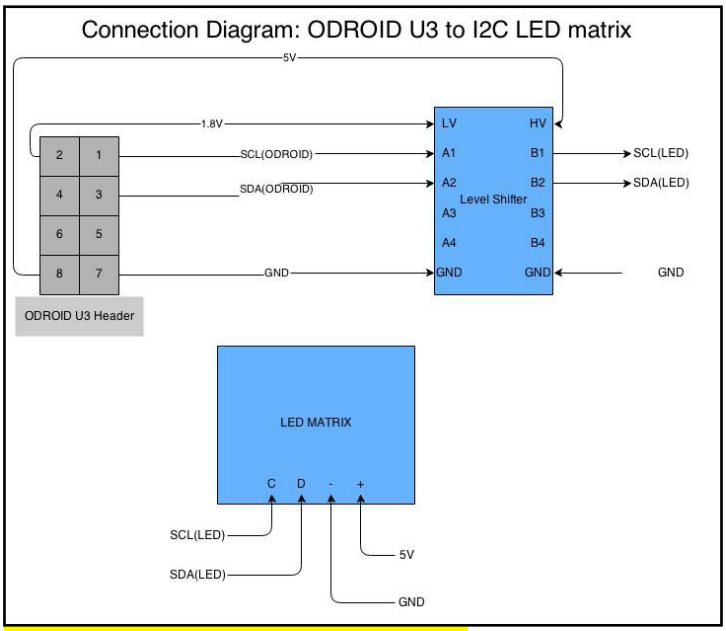
```
i2c-0   i2c             s3c2410-i2c                     I2C adapter
i2c-1   i2c             s3c2410-i2c                     I2C adapter
i2c-2   i2c             i2c-gpio2                       I2C adapter
i2c-3   i2c             s3c2410-i2c                     I2C adapter
i2c-4   i2c             i2c-gpio4                       I2C adapter
i2c-7   i2c             s3c2410-i2c                     I2C adapter
i2c-8   i2c             s3c2410-i2c                     I2C adapter
```

**A list of I2C ports available on the ODROID-U3.**

The bus that is mapped to the 8-pin connector is i2c-4. We will cover its usage after we connect a slave device to it.

## Wiring

Now that we have our ODROID set up to do I2C communication, we can connect our slave device. The device that we will be using is an LED matrix from Adafruit Industries. Since the ODROID-U3 is a 1.8V device and our LED matrix is a 5V device, we will use an I2C-safe level shifter, also from Adafruit Industries.




**A simple high-level diagram of the interaction between the U3 and the I2C LED Matrix.**

## Communicating

We now want to make sure that we have connected everything correctly. Luckily, we can do this easily using i2c-tools. After everything is connected execute the following command:

```
i2cdetect -y 4
```

This command tells the ODROID to list all of the I2C devices connected to bus 4. As you can see from the following figure, our LED matrix shows up at address 70.

If you do not see a device at address 70, double-check the wiring.

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

**A matrix of I2C peripherals, showing the ODROID's I2C device at address 70.**

## C code

Once we know that everything is connected properly, we can write some simple C code to control the LED matrix. The code shown below initializes the LED Matrix and sequentially lights every LED.

After you compile and run this code on the ODROID, you

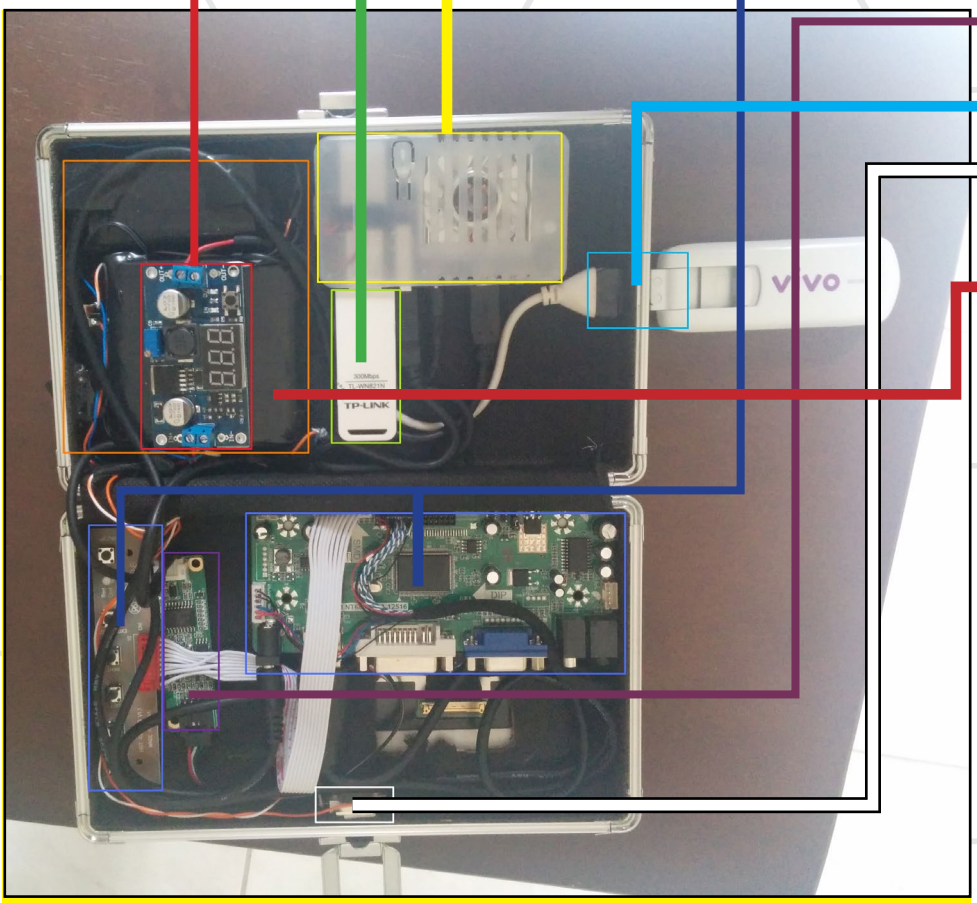
will see output similar to the video shown at `http://bit.ly/1fMOyMt`. The code can be easily can be modified to display other patterns/shapes on the LED Matrix.

```
#include <stdlib.h>
#include <unistd.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>

int i = 0;
int j = 0;
int k = 0;

int main(void)
{
char recieveBuffer[32];  //The Buffer that will
hold onto recieved data
     char transferBuffer[32]; //The buffer that
holds data that we will send
     int address = 0x70;       //The address of
the LED matrix
     int tenBitAddress = 0;    //variable that
says we aren't using 10-bit
    //addressing

     //Initialize the I2C channel
     int i2cHandle = open("/dev/i2c-4",O_
RDWR);

//Tell the I2C channel we aren't using ten bit
addressing
     ioctl(i2cHandle,I2C_
TENBIT,tenBitAddress);

//Tell the I2C channel we have a slave at Ad-
dress 70
     ioctl(i2cHandle,I2C_SLAVE,address);

//make sure there is no data in our buffers
memset(recieveBuffer, 0 , sizeof(recieveBuffer)
);
memset(transferBuffer,0,sizeof(transferBuffer));

//start internal oscillator on the LED matrix
by sending 0x21 command
transferBuffer[0] = 0x21;
write(i2cHandle, transferBuffer, 1);

//enable display and turn blink off by sending
0x81
transferBuffer[0] = 0x81;
write(i2cHandle, transferBuffer,1);

//set brightness to max by sending 0xEF
transferBuffer[0] = 0xEF;
write(i2cHandle, transferBuffer,1);

//top level loop keeps track of which column
we are on
for(i = 0; i<16;i=i+2)
{
for(j = 0; j<9;j++)
          {
//we send two bytes in this case, so we load
the
//transfer buffer with 2 bytes
//and set the first Byte to transfer to the
column number
transferBuffer[0] = i;

//set the second Byte to transfer to the
lights to turn on
transferBuffer[1] = 0x01 <<  j;
write(i2cHandle, transferBuffer,2);

//wait a while
for(k = 0; k < 5000000;k++);
}

//make sure a column is completely off before
leaving it
transferBuffer[1] = 0x00;
write(i2cHandle, transferBuffer,2);
}
}
```

# HEAVY-DUTY PORTABLE LINUX TABLET

## WITH LTE ROUTER

by Mauro Ribeiro




I wanted to build a ODROID-based tablet computer that was durable, rugged, and built of readily available components. A list of the hardware that I used is shown below the image to the right:

### Now for the Software

The software part of this project gave me the most headache! During my first tests with Ubuntu 13.10, I couldn't connect to the LTE network and was stuck with 3G, but I couldn't figure out why. It took me a while to discover that the ModemManager version on Ubuntu 13.10 is slightly out of date and didn't properly support LTE. Updating ModemManager by itself is nearly impossible since it's wired to NetworkManager and it has a lot of dependencies, so I needed something newer. My best option was to use ArchLinux ARM, which always contains the latest package releases.

Once I got ArchLinux ARM running, I needed a nice user interface that would play well with the touchscreen, since I don't have a keyboard or mouse connected to this. I tested a few UIs, including KDE Plasma Active, but then I saw some news about Mate 1.8 and decided to give it a try. It worked very nicely.

However, I still couldn't use my Modem to send or receive SMS, and I needed something to monitor the connection health. At first, my idea was to put all of those features on a custom app, but I

**The case** is made from a combination of aluminum and wood and measures 21x13x6cm, which I purchased at a local office supply store.

**The screen** is a 9" touch screen LCD. Now, I know that everyone will want to know where I got this, and I have some good news for you! The screen is a prototype for a kit that Hardkernel will be selling very soon: a 9" HDMI monitor with a built-in touchscreen.

I've opted to use an **ODROID-U3** mainly because of power consumption concerns. I wanted to keep that low since the screen and the LTE dongle also draw power. The LTE dongle uses nearly 500ma of current on its own!

For the **battery**, I used 6 Li-Ion cells wired together as two banks of 3 cells, which yields a possible 11.1V(12.3V) and 5000mAh. I salvaged them from a laptop battery.

A **step-down** converter (aka Buck regulator). I'm using a pre-made LM2596 kit. This IC can handle 2A without a heatsink, so its enough for our project.

**The LTE Dongle** was included for free from my mobile service carrier. It's a Huawei E3276 CAT 4 adapter and can reach a max speed of 150Mbps.

**The WiFi dongle** is based on the Realtek 8192CU chipset, which is very common. As an example, the one that I used is the TP-Link model TL-WN821N.

**White LED** is from Aliexpress. It is a 3x3 LED matrix rated at 10W. I'm using it only at 0.5W to illuminate the internals.

**RED** is the buck controller attached to the battery.

**YELLOW** is the ODROID-U3

**GREEN** is the WiFi Dongle

Two **BLUE** marks are the HDMI->LVDS board and the On-Screen display board.

**PURPLE** is the USB Touchscreen controller.

**CYAN** is the External USB port with the LTE Dongle connected.

**WHITE** is a 10W LED running at only 0.5W to light the internals in case we are in the dark.

**ORANGE** is the battery

The 6-cell battery is wired as shown in the image. This gives me 11.1V and 5000mAh.

Charging Li-Ion batteries isn't complicated, but it does require some small knowledge.

Each Li-Ion cell must be charged with 0.4V more than its rated voltage (3.7V) and you can only feed half of its rated current as charging current.

So for my case, since I'm using 3 cells in series (3.7 x 3 = 11.1V + 3x 0.4V = 1.2V = 12.3V), my charging voltage is 12.3V. Since I know that the total capacity is 5000mAh (~800mAh per battery), I won't use a charger rated over 2.5A. So I ended up using a 13V 2A PSU to charge the batteries.

I used a Dremel to make the external holes for USB, Power Connector, Power switch, USB, LVDS Cable and Touch cable.

ended up finding Modem Manager GUI (http://linuxonly.ru/cms/page.php?7) that does all of that. However, there was still one missing feature! I needed to control the modem so I could lock it to a certain network type (2G/3G/4G).

So, I made a custom app. I chose QT 5, since QT Creator makes Linux application development very easy. I needed to know what commands the modem required in order to force it to a certain network type. I couldn't find any documentation about it online, so I took the hard path! I connected the modem to a Windows computer, installed a serial port sniffer, and used the application provided with the modem to control it. Once I knew the commands, I finished my app.

In case you are curious, the commands for my HUAWEI Modem are:

**For automatic network selection:**

```
AT^SYSCFGEX="00",3fffff,1,2,5a,"",""
```

**For 2G Only mode:**

```
AT^SYSCFGEX="01",3fffff,1,2,5a,"",""
```

**For 3G Only mode:**

```
AT^SYSCFGEX="02",3fffff,1,2,5a,"",""
```

**And for LTE Only:**

```
AT^SYSCFGEX="03",3fffff,1,2,5a,"",""
```

Those commands are sent to a "control" serial port that the modem creates. Some modems (like mine) even have AT commands to allow grabbing its internal temperature! They provide a lot of good information that you can extract to learn the quality of your connection/signal.

Another issue I had to deal with was NetworkManager. You can't start hostapd to create a wifi network if NetworkManager is managing an interface. Even if you are disconnected from the wifi network, it's still possible to tell NetworkManager not to manage that interface.

You just add the following line to [Editor's note: need filename ]:

```
[keyfile]
unmanaged-devices=\
mac:xx:xx:xx:xx:xx:xx
```

Where xx:xx:xx:xx:xx:xx is the mac address of the device that you don't want to manage. I then added another feature to my application for turning AP on and off.

**Turn AP on:**

Tell NetworkManager to not manage my wifi adapter.

Start hostapd to create my wifi network.

Create a single iptables rules to setup NAT (share the Internet connection).

Start DNSMASQ to provide DHCP and DNS server.

**Turn AP off:**

Kill DNSMASQ

Clear firewall rules

Stop hostapd

Tell NetworkManager to manage my wifi again.

You may wonder, why not leave it unmanaged all the time? Because I still want to use the wifi as a client when I'm at home, so I can perform package upgrades and poke around.

Another feature required for the tablet was to install an on-screen keyboard on Linux, which is available in both Ubuntu 13.10 and ArchLinux via a package called onboard. Onboard is a highly configurable and customizable on-screen keyboard with many features. It works very well!

Finally, I needed to enable right button emulation while using the touchscreen. This is done by adding the following configuration to the /etc/X11/org.conf file, or to a new configuration file in the directory /etc/X11/xorg.conf.d.

```
Section "InputClass"
    Option "EmulateThirdButton" "1"
    Option "EmulateThirdButtonTimeout" "750"
    Option "EmulateThirdButtonMoveThreshold" "30"
EndSection
```

The EmulateThirdButtonTimeout is the amount of time in milliseconds that you need to keep the touchscreen pressed in order to be identified as a right click. EmulateThirdButtonMoveThreshold is the amount of pixels that your finger can move and still be considered as the same position.

With all of that done, you now have a Linux-powered touch screen tablet, that also functions as an LTE router, enabling you to tether to a 4G network from anywhere!

# HOW I BUILT A TRUCK PC WITH MY ODROID

## NEVERMIND THE PRODUCTS ON THE MARKET, GET THE MOST BANG FOR YOUR BUCK!

by Jeremy Leesmann (Killer Turtle)

Cars with high-end navigation systems are becoming more common as in-vehicle technology improves. However, the main issues that many CarPC units have is that 1) the map updates are expensive, and 2) you are limited on functions and software. There are a number of Android-powered head units available now, but they generally run outdated versions of Android, and are consequently slow. So, I wanted to install a CarPC in my truck with the most bang for my buck, and chose the ODROID-U3 as the platform for my project.

This ODROID PC navigation system keeps Jeremy's truck on the road and out of the mud!

My goal was to have the following functions available through my Truck PC:

- **Navigation**
- **Radio**
- **MP3 player**
- **Physical button controls**

Here are the parts I used to build my system:

**ODROID-U3 with 16GB eMMC, and Real-Time Clock (RTC) battery**
**USB Wifi Dongle**
**Bluetooth dongle**
**USB self-powered hub**
**USB GPS (BU-353-S4)**
**Arduino Uno Rev 3**
**16 GB USB Drive**
**Lilliput 7" capacitive touchscreen**
**12 momentary push buttons**

A list of applications that I installed on the ODROID:

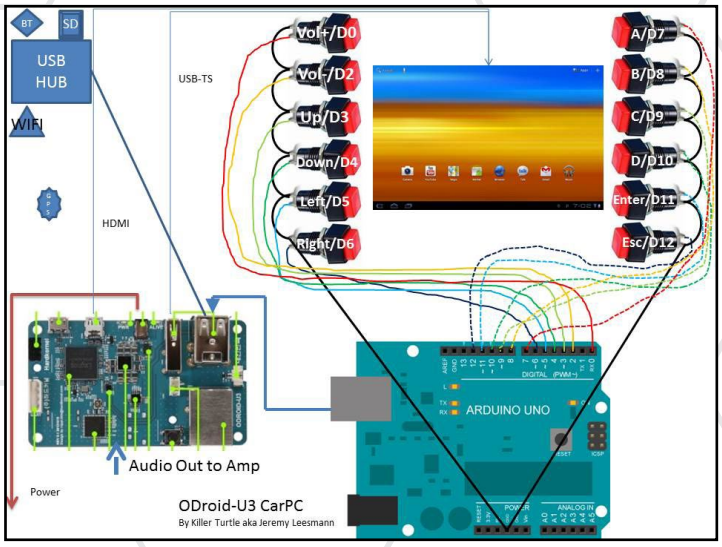
**Waze (Navigation)**
**Pandora (Internet Radio)**
**Google Play Music (MP3)**
**USBGPS (Gives access to GPS via the USB port)**
**Anycut (Gives access to Quick Launch settings)**
**Nova Launcher (customizable home screen)**
**SwiftKey keyboard (or any 3rd party keyboard)**
**GApps for Play Store**
**Custom kernel for touchscreen input (thanks to @mdrjr for building it)**
**And some PC applications used to set up the Arduino and USB GPS**
**Arduino IDE**
**SiRF Demo (to configure USB GPS for correct baud rate)**

### Setting up the Arduino as a Keyboard

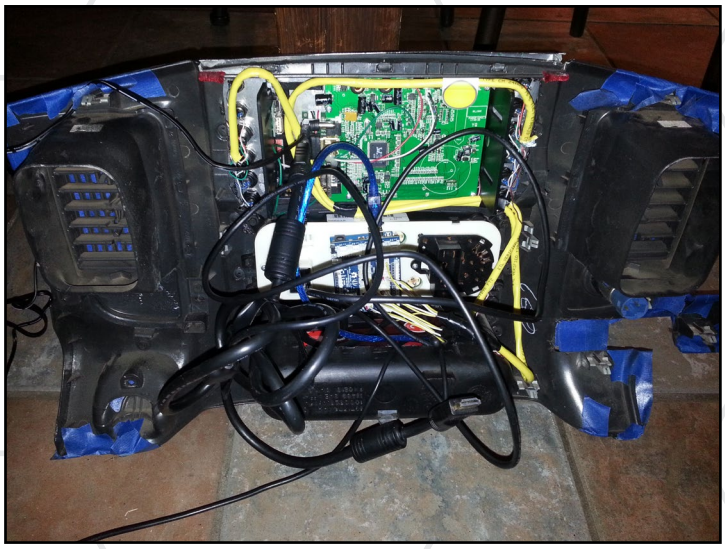
The first task is getting the Arduino to emulate a USB HID keyboard. Start by building the code for the Arduino, which is included at the end of this article.

Once the code is built and uploaded, go to http://hunt.net.nz/users/darran/ and follow the directions for putting the Arduino into DFU program mode, and program it to be a USB HID Keyboard. For reference, here is a map of all the keyboard codes: http://www.usb.org/developers/devclass_docs/Hut1_11.pdf.

I wired all the buttons to a common ground, and then each one to its respective pin on the Arduino. Then, all I needed to do was connect the Arduino to the ODROID.




**The Arduino Uno, when combined with an ODROID-U3, makes connecting interface buttons as easy as connecting the dots.**



## Setting up the ODROID

First, install the custom kernel for your touchscreen. For detailed instructions, please refer to the February issue of ODROID Magazine's Giant Android Tablet cover article.

Next, install GApps in order to get access to the Play Store, or you can install Amazon Appstore. [ Editor's note: There are several posts on the ODROID forums explaining how to install GApps on your ODROID. The simplest way is to use the Android Epic Loot Software Collection, available for free download from the forums, which includes a one-click Gapps installer app for Android versions 4.1.2 and 4.2.2 ]

For my home screen, I installed Nova Launcher because it looks great, but you can use any similar application to customize the desktop. To get the Arduino buttons to work as hotkeys for opening apps, go to the Play Store and install Anycut. After it's installed, add a shortcut, click activity, and choose the first Quick Launch that are shown (there are most likely 3 of them). This will place a shortcut on your home screen for the "Quick Launch" settings. Open the settings and assign the first four to your choice of apps. My Quick Launch icons are Waze, Pandora, and Play Music, with the last button going back to the Home screen.

Next, install a third-party keyboard. I have a Swiftkey, but any virtual keyboard will work. Once the keyboard is working, go to Settings, and Language and input, click on Default, and turn off Hardware keyboard. This will allow the virtual keyboard to work while a physical keyboard is attached.

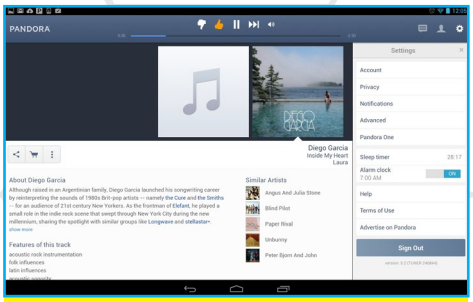
Now, attach your GPS. If you get the one I have (BU-353-S4), follow these instructions: http://bit.ly/1gzbAXr. Complete the software installation by installing any other apps that you may find useful, such as Skype or Google Hangouts.

## Installing the ODROID in your vehicle

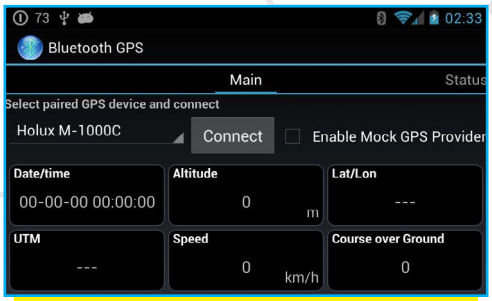
For my truck, I installed a 12V plug connected to a switched 12V line to run both the screen and USB hub. My PSU has a USB port rated up to 2.1 Amps which I use for powering the ODROID itself. I also installed a 400W 4-channel amp, connected everything up to the original stereo connections, and ran an RCA-to-headphone cord from the Audio Out on the ODROID to the input on the amp.

The ODROID connects to the Internet via a Wifi hotspot on my phone. You may need to mount the GPS receiver on the roof (or some other area with an unobstructed upward view) using a USB cord extension in order to ensure a stable connection. In my case, everything works great now, and I was able to build a fast, reliable Truck PC using an inexpensive ODROID-U3.

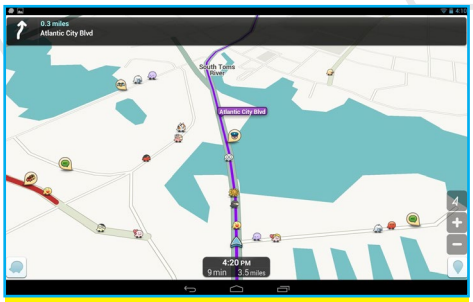
```
/* Arduino USB Keyboard HID for ODroid
* Made by Jeremy Leesmann aka Killer Turtle
*/

#define KEY_LEFT_CTRL 0x01
#define KEY_LEFT_SHIFT 0x02
#define KEY_RIGHT_CTRL 0x10
#define KEY_RIGHT_SHIFT 0x20
#define KEY_LEFT_GUI 0xE3

uint8_t buf[8] = { 0 }; /* Keyboard report
buffer */
#define PIN_VolP 7
#define PIN_VolM 8
#define PIN_Enter 5
#define PIN_Escape 6
#define PIN_Up 9
#define PIN_Down 10
#define PIN_Left 11
#define PIN_Right 12
#define PIN_A 0
```




This TruckPC is ready for a non-stop Pandora party on the beach.



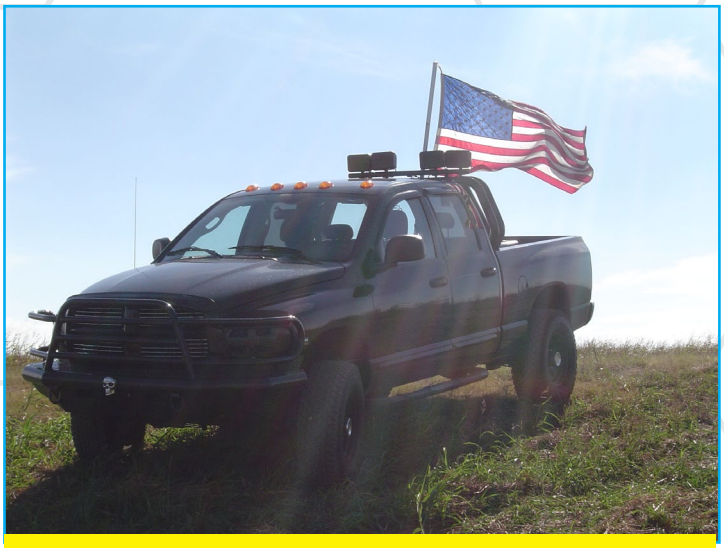

The GPS interface makes sure you can pick up your date on time for a romantic off-road adventure.




Why go through traffic when you can drive over it with your monster truck tires?

```
#define PIN_B 2
#define PIN_C 3
#define PIN_D 4
#define PIN_Space 13

int state = 1;

void setup()
{
Serial.begin(9600);
pinMode(PIN_VolP, INPUT);
pinMode(PIN_VolM, INPUT);
pinMode(PIN_Enter, INPUT);
pinMode(PIN_Escape, INPUT);
pinMode(PIN_Up, INPUT);
pinMode(PIN_Down, INPUT);
pinMode(PIN_Left, INPUT);
pinMode(PIN_Right, INPUT);
pinMode(PIN_A, INPUT);
pinMode(PIN_B, INPUT);
pinMode(PIN_C, INPUT);
pinMode(PIN_D, INPUT);
pinMode(PIN_Space, INPUT);
// Enable internal pull-ups
digitalWrite(PIN_VolP, 1);
digitalWrite(PIN_VolM, 1);
digitalWrite(PIN_Enter, 1);
digitalWrite(PIN_Escape, 1);
digitalWrite(PIN_Up, 1);
digitalWrite(PIN_Down, 1);
digitalWrite(PIN_Left, 1);
digitalWrite(PIN_Right, 1);
digitalWrite(PIN_A, 1);
digitalWrite(PIN_B, 1);
digitalWrite(PIN_C, 1);
digitalWrite(PIN_D, 1);
digitalWrite(PIN_Space, 1);
delay(200);
}
void loop() {
```

```
state = digitalRead(PIN_VolP);
if (state != 1) {
buf[2] = 128; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_VolM);
if (state != 1) {
buf[2] = 129; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_Enter);
if (state != 1) {
buf[2] = 40; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_Escape);
if (state != 1) {
buf[2] = 41; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_Up);
if (state != 1) {
buf[2] = 82; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
```



**The ODROID TruckPC goes anywhere in style, including your favorite grassy hilltop.**

```
state = digitalRead(PIN_Down);
if (state != 1) {
buf[2] = 81; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_Left);
if (state != 1) {
buf[2] = 80; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_Right);
if (state != 1) {
buf[2] = 79; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_A);
if (state != 1) {
buf[0] = KEY_LEFT_GUI; // Windows Key
buf[2] = 4; // Letter a
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_B);
if (state != 1) {
buf[0] = KEY_LEFT_GUI; // Windows Key
buf[2] = 5; // Letter a
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_C);
if (state != 1) {
buf[0] = KEY_LEFT_GUI; // Windows Key
buf[2] = 6; // Letter a
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_D);
if (state != 1) {
buf[0] = KEY_LEFT_GUI; // Windows Key
buf[2] = 7; // Letter a
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
state = digitalRead(PIN_Space);
if (state != 1) {
buf[2] = 44; // Vol +
//buf[2] = 27; // Letter X
// buf[2] = 123; // Cut key: Less portable
Serial.write(buf, 8);// Ssend keypress
releaseKey();
}
}
void releaseKey()
{
buf[0] = 0;
buf[2] = 0;
Serial.write(buf, 8);// Release key
delay(500);
}
```

# MEET AN ODROIDIAN

## MARIAN MIHAILESCU: ONE OF OUR TOP FORUM CONTRIBUTORS

***edited by Rob Roy***

*Please tell us a little about yourself.*

I am a computer science research fellow currently living in Adelaide, Australia, working at the Teletraffic Research Centre, Adelaide University. Originally from Romania, I did my undergraduate studies in computer science in Bucharest, and then PhD at National University of Singapore.

Visiting Arapiles - Australia.

*How did you get started with computers?*

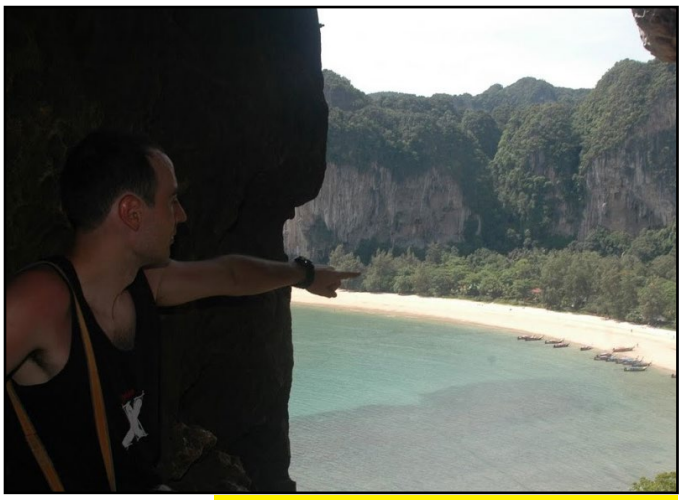
As a kid, I saw a Spectrum ZX at a family friend and liked the games. I am still very fond of many of those games, like Saboteur, Dizzy, Elite or Chuckie Egg. I had the opportunity to join a special secondary school class where the BASIC programming language was taught, and also joined the (small) local computer club. The high school that I went at also had an informatics degree program. Growing up in a communist country, import of technology was prohibited, so Spectrum was all I grew up with; I saw my first PC when I started high school.

Journeying through Nepal at 5000M high.

*What types of projects have you done with your Odroid?*

I initially got the ODROID U2 for a low-power HTPC, as my Raspberry Pi was too sluggish. From all the ARM A9-based boards at the time, the ODROID was the only quad core that had prospects for getting Open GLES working in Linux. I actually helped out a bit with this and with getting XBMC working, and won the ODROID XU as a monthly forum award. Currently, I am using the XU for more than just HTPC - it's also a home server (Apache, MySQL, SSH), download box, and home monitoring system (motion detection and tempera-

Climbing 4000M high at malaysian mountains.



Exploring exotic beaches at Thailand.

ture logging). I am working now on adding more functions - I want to use it to control my cat feeder, control the air conditioning unit, and the garage door. Also planned is integration of all of this functionality with SiriProxy, to get all this control on my phone via the Internet.

*What other hobbies and interests do you have?*

I love travelling, mountaineering, climbing and bouldering. Living in Singapore gave me the opportunity to visit a lot of places in southeast Asia, Thailand being the perfect place where you can enjoy sport climbing right on a beautiful beach. Adelaide, where I currently live, is also close to the Grampians National Park and Mount Arapiles, two landmark places for bouldering and traditional climbing. In terms of mountaineering, I am currently planning my second trip to Nepal - a visit to the Everest base camp.

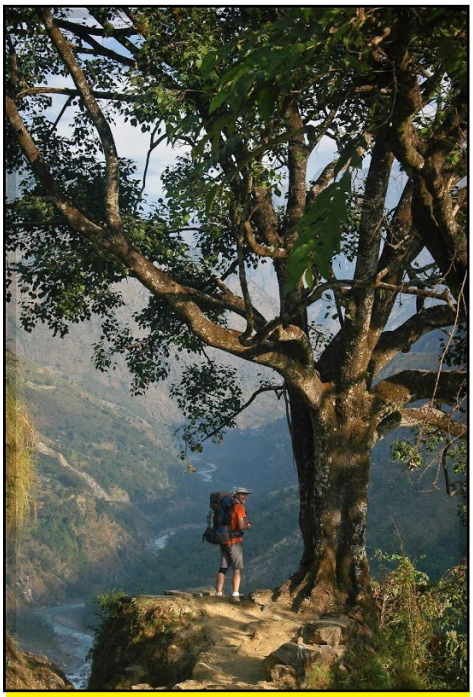
Rock climbing at Thailand, enjoying the nature and the adrenaline that comes with it.

*What do you like most about the ODROID community?*

It's the community itself that I like. If you are working on a project, you can ask for advice and people will go out of their way to help and give you valuable advice. If you have issues, you can raise them on the forums and, if a solution does not exist, a fix will be provided.

*6. Are you involved with any other software or hardware projects in addition to the ODROID?*

Besides work, my projects on the ODROID take up most of my time. So the answer is no. My old Raspberry Pi was completely replaced by the XU. Whatever project I do for myself, I open up and describe for others too - there are several HowTo's I contributed on the forum, a couple of articles in the ODROID Magazine, and the charting library that I created, which exports data from RRDs to highcharts. Most of the things there are general and should work on any platform, and aren't specific to the ODROID platform.



From Nepal's high mountains to its hidden valleys, we just hope Marian had the chance to use his ODROID to capture video of his spectacular travels.