

## 最新最全网络首发安卓 4.0 源码及内核下载编译教程（图文详解）

{本教程由TD屋（[WWW.TDWU.NET](http://WWW.TDWU.NET)）提供} QQ群：183940436 日期：2011-12-19

作者：TD-龙王

☐ 未测试    ☒ 已测试

**简介：** 安卓 4.0 源码下载及编译，内核下载及编译。模拟器启动，sdk 安装。一步到位！

现在很多机型都有尝鲜版的安卓 4.0 了，即便很多功能不能用，画面不够完善。大家依旧那么兴奋的烧鸡测试。大家想不想把自己的机器也搞成 4.0，下面由我从 4.0 源码及编译的最底层来揭晓安卓 4.0 的奥秘吧。

**注：** 感谢大家欣赏我的教程。本教程不属于小白教程，教程如有不足之处还望多多指出。

欢迎有兴趣的朋友加 QQ 群：183940436 参与内核移植让 1.6 和 2.3 等机型升级

## 二下载并编译 Android4.0 源码（图文）

### 1 开始

本教程笔者是在 ubuntu10.04 上用虚拟机完成的，建议将虚拟机设成 80G 以上，内存为 1.5G，否则编译过程会出错。

#### 1.1 初始化构建环境

这节主要是描述如何设置你的本地工作环境，怎么使用 repo 这个工具获取 Android 文件，怎么创建你机器上的文件。使用 Linux 或 Mac Os 才能构建 Android 源文件，Windows 目前不支持。

**注：** 源文件有 6G 大小。构建时，你需要 25G 空间来完成单独构建，而完整构建时需要 80G 空间。

#### 1.2 设置 Linux 构建环境

**注：** 在虚拟机内构建 Android 也是可以的。如果那样的话，那么你至少需要 16G 内存或交换分区和 30G 以上的磁盘空间。

一般来说你需要以下工具：

Python 2.4 -- 2.7, 你可从这下载 [python.org](http://python.org).

JDK 6 (Gingerbread 或更新版本), JDK 5 (Froyo 或更老版本.) 这些你都可以从 [java.sun.com](http://java.sun.com) 下载.

Git 1.7 或更新版本. 你可以 [git-scm.com](http://git-scm.com) 下载.

##### 1.2.1 安装 JDK

Sun JDK 不再包含在 Ubuntu 的服务器上了，如果你要下载，你需要添加源和指明那个版本是你想要下载的。

安装 Java6 的方法:

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install sun-java6-jdk
```

### 1.2.2 安装需要的包

安装如下包:

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential \
zip curl zlib1g-dev libc6-dev libncurses5-dev x11proto-core-dev \
libx11-dev libreadline6-dev libgl1-mesa-dev tofrodos python-markdown \
libxml2-utils
```

执行如下指令:

```
$ sudo ln -s /usr/lib/i386-linux-gnu/libX11.so.6 /usr/lib/i386-linux-gnu/libX11.so
```

### 1.3 配置 USB

在 GNU/Linux 系统下, 默认情况下, 正常用户不能访问 USB 设备, 而此时需要配置其可以访问. 建议方法是在目录 `/etc/udev/rules.d/` 下以 root 权限创建一个 `51-android.rules` 文件:

```
$ sudo gedit /etc/udev/rules.d/51-android.rules
```

然后将以下的内容复制并保存, 注: 需要将字符串 `username` 替换成你的用户名, 即开机登陆时你的用户名.

```
# adb protocol on passion (Nexus One)
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e12",
MODE="0600", OWNER="<username>"
```

```
# fastboot protocol on passion (Nexus One)
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="0bb4", ATTR{idProduct}=="0fff",
MODE="0600", OWNER="<username>"
```

```
# adb protocol on crespo/crespo4g (Nexus S)
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e22",
MODE="0600", OWNER="<username>"
```

```
# fastboot protocol on crespo/crespo4g (Nexus S)
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e20",
MODE="0600", OWNER="<username>"
```

```
# adb protocol on maguro (Galaxy Nexus)
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="04e8", ATTR{idProduct}=="6860",
MODE="0600", OWNER="<username>"
```

```
# fastboot protocol on maguro (Galaxy Nexus)
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e30",  
MODE="0600", OWNER="<username>"  
# adb protocol on panda (PandaBoard)  
SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d101",  
MODE="0600", OWNER="<username>"  
# fastboot protocol on panda (PandaBoard)  
SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d022",  
MODE="0600", OWNER="<username>"  
这样配置完成只有重新插入 USB 才会生效.
```

## 2 下载源文件

### 2.1 下载源文件树

#### 2.1.1 安装 repo

Repo 是 Android 的源码管理工具，以下步骤是安装，初始化，配置 repo。  
第一步:在主目录下新建一个 bin 文件夹，并将其设置为环境参数 PATH 内。

```
$ mkdir ~/bin  
$ PATH=~/bin:$PATH
```

第二步:下载repo脚本文件，并使其可执行。

```
$ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo  
$ chmod a+x ~/bin/repo
```

#### 2.1.2 初始化repo客户端

第一步:在主目录上新建一个文件夹WORKING\_DIRECTORY,然后进入这个路径。

```
$ mkdir WORKING_DIRECTORY  
$ cd WORKING_DIRECTORY
```

第二步:清单库初始化

```
$ repo init -u https://android.googlesource.com/platform/manifest  
这个命令会要求你输入你的用户名和邮箱
```

## 2.2 获取Android源文件

同步:

```
$ repo sync
```

接下来就是正式开始下载源码了，要很长时间才能下完，可以使用

```
$ repo sync -j10
```

来加快下载速度，即使用 10 个线程并行下载。

接下来就是等了，一直等到下载完,估计要两三天才能下完哦，要做好心理准备.

注：在下载过程中，在WORKING\_DIRECTORY目录会自动生成一个隐藏的目录.repo,这个目录就是下载的内容，不过是隐藏的，用ls是看不见的.

如果在下载过程中出现错误的话，可再打开另一个终端，输入以下命令：

```
$ PATH=~/.bin:$PATH
```

```
$ repo init -u https://android.googlesource.com/platform/manifest
```

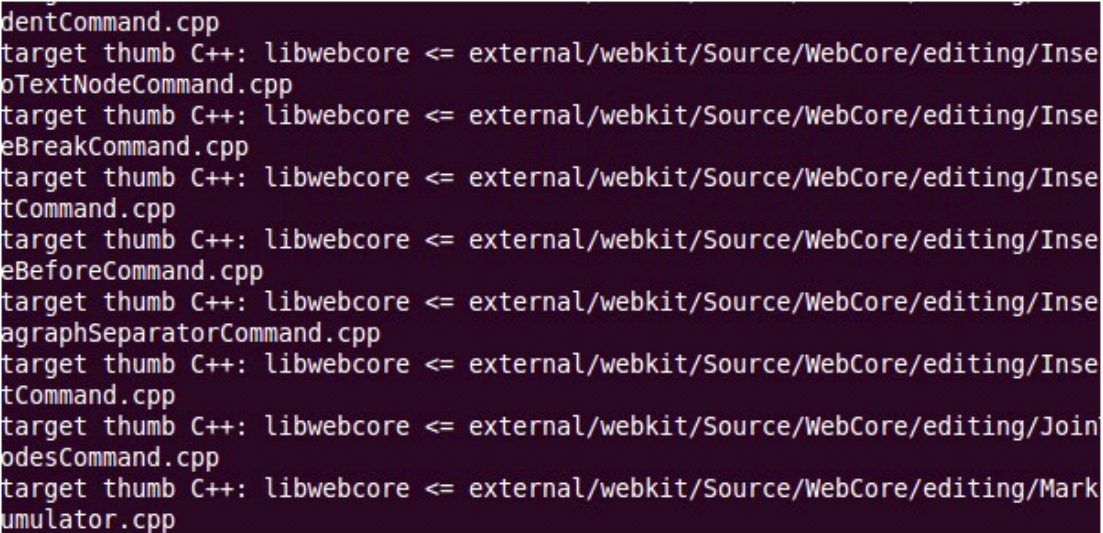
注意：以上这个下载的是安卓 4.0 完整版版本号包括 4.0.1—到 4.0.9

```
$repo init -u https://android.googlesource.com/platform/manifest -b  
android-4.0.1\_r1
```

注意：单独下载的话可以选 4.0.1 的这个

```
$ repo sync
```

然后再重复上述三条指令即可，如此，就可以完成源码下载了。



```
dentCommand.cpp  
target thumb C++: libwebcore <= external/webkit/Source/WebCore/editing/Inse  
oTextNodeCommand.cpp  
target thumb C++: libwebcore <= external/webkit/Source/WebCore/editing/Inse  
eBreakCommand.cpp  
target thumb C++: libwebcore <= external/webkit/Source/WebCore/editing/Inse  
tCommand.cpp  
target thumb C++: libwebcore <= external/webkit/Source/WebCore/editing/Inse  
eBeforeCommand.cpp  
target thumb C++: libwebcore <= external/webkit/Source/WebCore/editing/Inse  
agraphSeparatorCommand.cpp  
target thumb C++: libwebcore <= external/webkit/Source/WebCore/editing/Inse  
tCommand.cpp  
target thumb C++: libwebcore <= external/webkit/Source/WebCore/editing/Join  
odesCommand.cpp  
target thumb C++: libwebcore <= external/webkit/Source/WebCore/editing/Mark  
umulator.cpp
```

下载结束如下图所示：

```

    arthur@arthur-laptop: ~/WORKING_DIRECTORY
Unpacking objects: 100% (5/5), done.
From https://android.googlesource.com/platform/ndk
   80af6c7..bebbc18  master    -> aosp/master
remote: Counting objects: 201, done
remote: Counting objects: 15, done
remote: Finding sources: 100% (9/9)
remote: Getting sizes: 100% (8/8)
remote: Compressing objects: 100% (2/2)
remote: Total 9 (delta 5), reused 8 (delta 5)
Unpacking objects: 100% (9/9), done.npacking objects:  33% (3/9)
remote: Finding sources: 100% (127/127)
remote: Getting sizes: 100% (134/134)
From https://android.googlesource.com/platform/system/core
   beec006..a51b3cc  gingerbread -> aosp/gingerbread
remote: Compressing objects: 100% (60/60)
remote: Total 127 (delta 66), reused 121 (delta 66)
Receiving objects: 100% (127/127), 88.18 KiB, done.
Resolving deltas: 100% (66/66), completed with 42 local objects.
From https://android.googlesource.com/platform/sdk
   3f69598..3ce45b2  master    -> aosp/master
Fetching projects: 100% (222/222), done.
Syncing work tree: 100% (221/221), done.

arthur@arthur-laptop:~/WORKING_DIRECTORY$
```

OK, 下载完了,查看一下你的 WORKING\_DIRECTORY 目录,下面就是 Android4.0 的源文件了。(下载过程是很漫长的笔者下载了 8-10 小时。建议晚上睡觉前挂机下载)

### 3 开始编译

#### 3.1 初始化环境

```
$ source build/envsetup.sh
```

#### 3.2 选择目标

```
$ lunch full-eng
```

大概会再现如下提示:

```
arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY$ source build/envsetup.sh
including device/samsung/maguro/vendorsetup.sh
including device/samsung/tuna/vendorsetup.sh
including device/ti/panda/vendorsetup.sh
including sdk/bash_completion/adb.bash
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY$ lunch full-eng

=====
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=4.0.1
TARGET_PRODUCT=full
TARGET_BUILD_VARIANT=eng
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=arm
TARGET_ARCH_VARIANT=armv7-a
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=ITL41D
=====

arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY$
```

### 3.3 编译源码

\$ make -j4

大概会出现如下提示:

```
arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY
TARGET_ARCH=arm
TARGET_ARCH_VARIANT=armv7-a
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=ITL41D
=====

arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY$ make -j4
=====
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=4.0.1
TARGET_PRODUCT=full
TARGET_BUILD_VARIANT=eng
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=arm
TARGET_ARCH_VARIANT=armv7-a
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=ITL41D
=====
```

接下来就会编译很长时间,然而编译过程并不会是一帆风顺的,如果你的 OS 是 32 位 Ubuntu 10.04,有可能会出现很多错误,笔者出现了至少 5 个错误,编译过程是锻炼开发者毅力的时候一个错误下来就要重新编译一次就是 5 个小时以上,希望大家,有恒心有毅力!

稍后请留意我的一篇[编译错误集结篇](#)。

解决问题后：重新 `make -j4` 即可

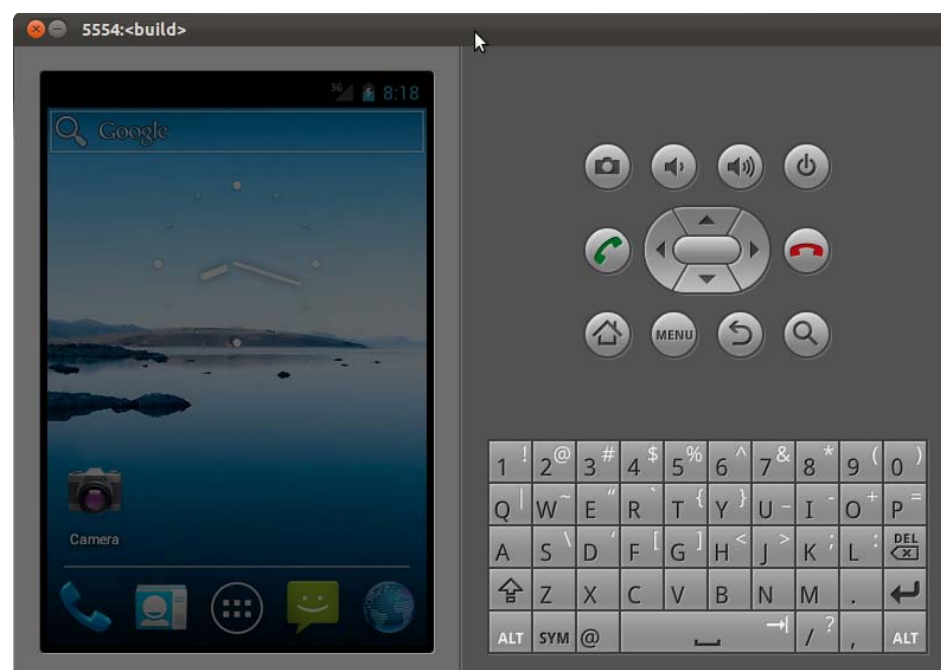
编译过程大概花了 2+ 个小时。

完成后如下图所示：

```
arthur@arthur-laptop: ~/WORKING_DIRECTORY
x'...
'out/target/common/obj/APPS/Contacts_intermediates//classes.dex' as 'classes.de
x'...
Processing target/product/generic/obj/APPS/SystemUI_intermediates/package.apk
Done!
Install: out/target/product/generic/system/app/SystemUI.odex
Install: out/target/product/generic/system/app/SystemUI.apk
Processing target/product/generic/obj/APPS/Contacts_intermediates/package.apk
Done!
Install: out/target/product/generic/system/app/Contacts.odex
Install: out/target/product/generic/system/app/Contacts.apk
'out/target/common/obj/APPS/Settings_intermediates//classes.dex' as 'classes.de
x'...
Processing target/product/generic/obj/APPS/Settings_intermediates/package.apk
Done!
Install: out/target/product/generic/system/app/Settings.odex
Install: out/target/product/generic/system/app/Settings.apk
Finding NOTICE files: out/target/product/generic/obj/NOTICE_FILES/hash-timestamp
Combining NOTICE files: out/target/product/generic/obj/NOTICE.html
Installed file list: out/target/product/generic/installed-files.txt
Target system fs image: out/target/product/generic/obj/PACKAGING/systemimage_int
ermediates/system.img
Install system fs image: out/target/product/generic/system.img
arthur@arthur-laptop:~/WORKING_DIRECTORY$
```

### 3.4 运行

\$emulator



第一次打开模拟器会很慢。请大家，多等待

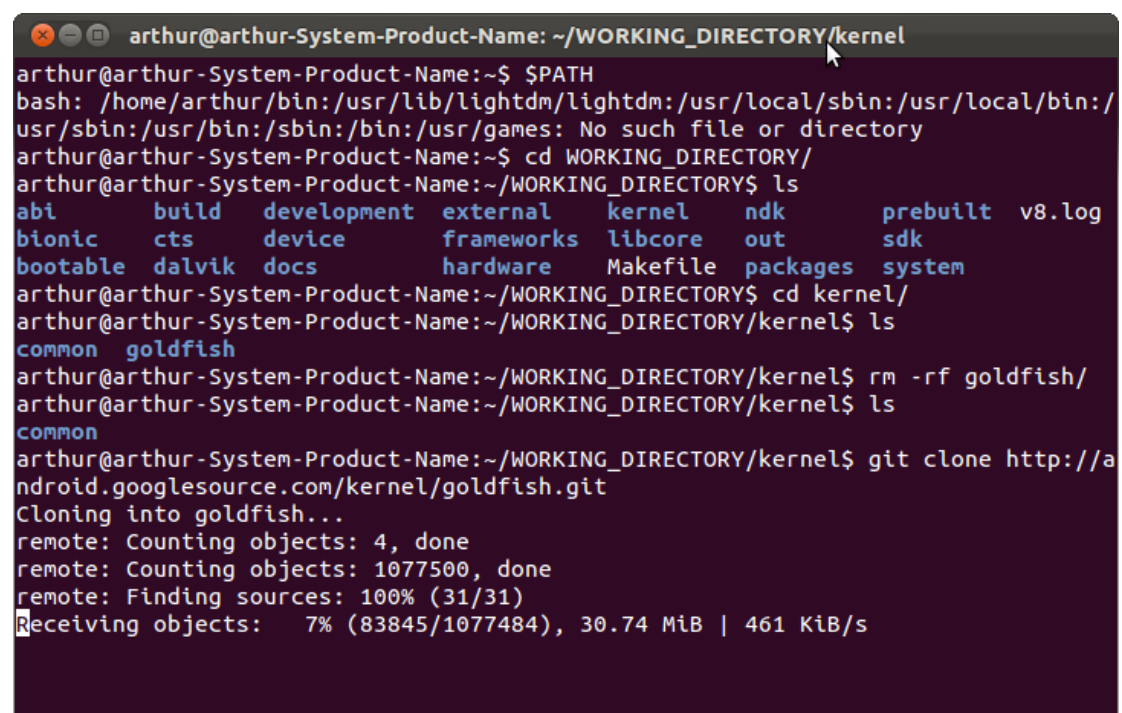


## 二下载并编译 Android4.0 内核 goldfish(图文)

第一步:下载 goldfish 源码

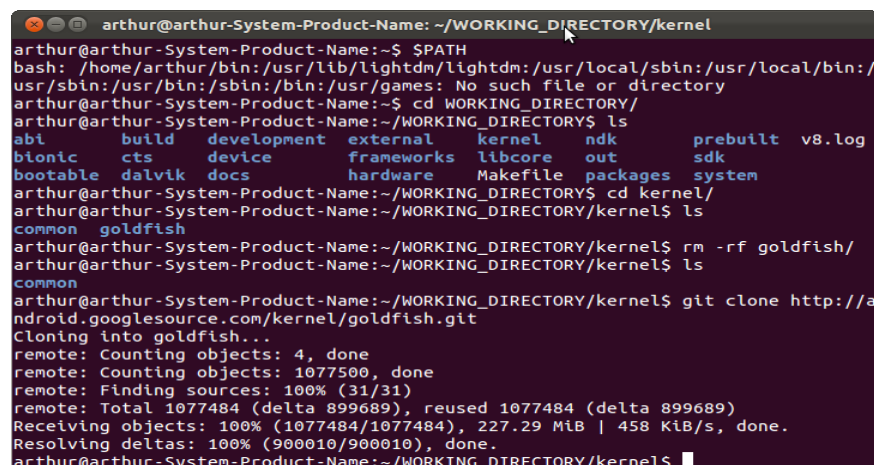
在 Android 源码根目录下新建 kernel 文件夹

```
1. $mkdir kernel  
2. $cd kernel  
  
1. $git clone http://android.googlesource.com/kernel/goldfish.git
```



```
arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY/kernel  
arthur@arthur-System-Product-Name:~$ $PATH  
bash: /home/arthur/bin:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games: No such file or directory  
arthur@arthur-System-Product-Name:~$ cd WORKING_DIRECTORY/  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY$ ls  
abi      build    development  external  kernel  ndk      prebuilt  v8.log  
bionic   cts      device      frameworks libcore  out       sdk  
bootable dalvik   docs        hardware  Makefile packages system  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY$ cd kernel/  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$ ls  
common  goldfish  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$ rm -rf goldfish/  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$ ls  
common  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$ git clone http://a  
ndroid.googlesource.com/kernel/goldfish.git  
Cloning into goldfish...  
remote: Counting objects: 4, done  
remote: Counting objects: 1077500, done  
remote: Finding sources: 100% (31/31)  
Receiving objects: 7% (83845/1077484), 30.74 MiB | 461 KiB/s
```

下载完毕如下图:



```
arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY/kernel  
arthur@arthur-System-Product-Name:~$ $PATH  
bash: /home/arthur/bin:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games: No such file or directory  
arthur@arthur-System-Product-Name:~$ cd WORKING_DIRECTORY/  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY$ ls  
abi      build    development  external  kernel  ndk      prebuilt  v8.log  
bionic   cts      device      frameworks libcore  out       sdk  
bootable dalvik   docs        hardware  Makefile packages system  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY$ cd kernel/  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$ ls  
common  goldfish  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$ rm -rf goldfish/  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$ ls  
common  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$ git clone http://a  
ndroid.googlesource.com/kernel/goldfish.git  
Cloning into goldfish...  
remote: Counting objects: 4, done  
remote: Counting objects: 1077500, done  
remote: Finding sources: 100% (31/31)  
remote: Total 1077484 (delta 899689), reused 1077484 (delta 899689)  
Receiving objects: 100% (1077484/1077484), 227.29 MiB | 458 KiB/s, done.  
Resolving deltas: 100% (900010/900010), done.  
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel$
```



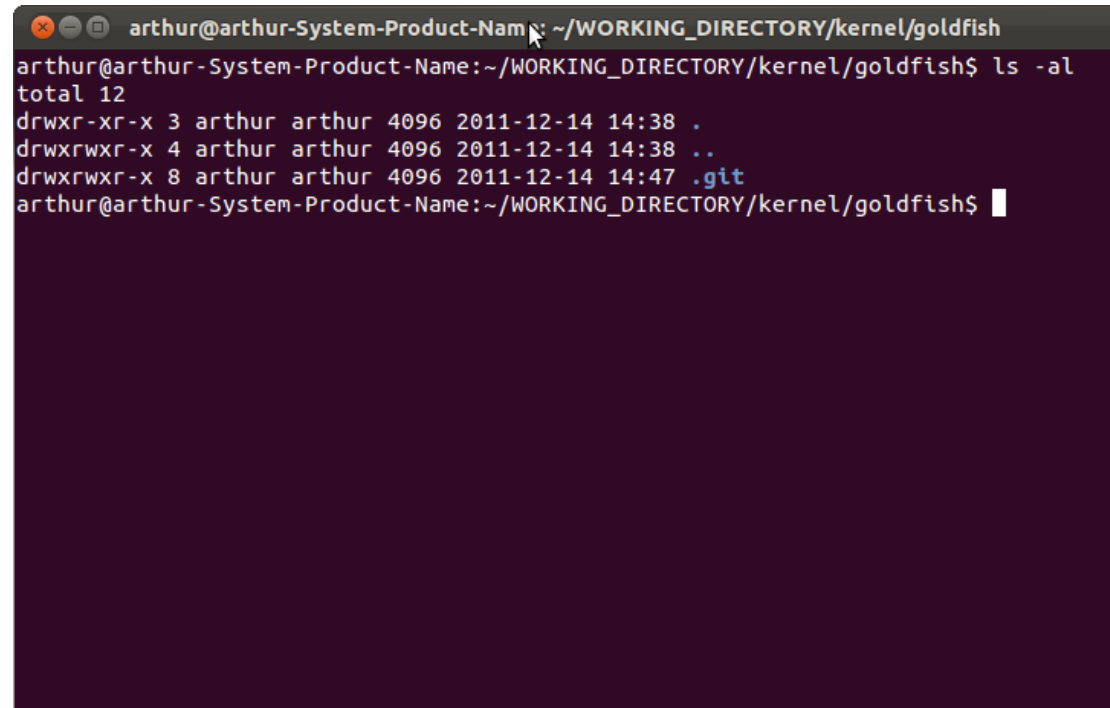
此时在 `kernel` 目录下会生成一个 `goldfish` 文件夹。进入此目录:

```
1. $cd goldfish
```

此目录下有一个隐藏的目录`.git`,通过

```
1. $ls -al
```

可看到此目录:



```
arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY/kernel/goldfish
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$ ls -al
total 12
drwxr-xr-x 3 arthur arthur 4096 2011-12-14 14:38 .
drwxrwxr-x 4 arthur arthur 4096 2011-12-14 14:38 ..
drwxrwxr-x 8 arthur arthur 4096 2011-12-14 14:47 .git
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$
```

查看所有分支:

```
1. git branch -a
```

如下图:

```

arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY/kernel/goldfish
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$ ls -al
total 12
drwxr-xr-x 3 arthur arthur 4096 2011-12-14 14:38 .
drwxrwxr-x 4 arthur arthur 4096 2011-12-14 14:38 ..
drwxrwxr-x 8 arthur arthur 4096 2011-12-14 14:47 .git
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$ git branch
h -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/android-goldfish-2.6.29
  remotes/origin/master
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$
```

1. `$git checkout remotes/origin/android-goldfish-2.6.29`

```

arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY/kernel/goldfish
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$ git checkout
remotes/origin/android-goldfish-2.6.29
Checking out files: 100% (26801/26801), done.
Note: checking out 'remotes/origin/android-goldfish-2.6.29'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 46b05b2... goldfish: Enable CONFIG_TUN
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$
```

此时你会看到 goldfish 目录下会出现很多文件:

1. `$ls`

```
arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY/kernel/goldfish
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$ git checkout
remotes/origin/android-goldfish-2.6.29
Checking out files: 100% (26801/26801), done.
Note: checking out 'remotes/origin/android-goldfish-2.6.29'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

    git checkout -b new_branch_name

HEAD is now at 46b05b2... goldfish: Enable CONFIG_TUN
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$ ls
arch      crypto    fs        Kbuild    Makefile  REPORTING-BUGS  sound
block     Documentation  include  kernel    mm        samples        usr
COPYING   drivers   init      lib        net       scripts        virt
CREDITS   firmware  ipc       MAINTAINERS  README   security
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$
```

这个时候 goldfish 源码就已经下下来了，接下来的事情就是编译了。

## 第二步:编译 goldfish

导出交叉编译工具目录到\$PATH 环境变量中去。

```
1. export PATH=$PATH:~/WORKING_DIRECTORY/prebuilt/linux-x86/toolchain/arm-
eabi-4.4.3/bin
```

2. 我们将使用上述这个目录下的交叉编译器 **arm-eabi-gcc**
3. 然后在 goldfish 目录下用 gedit 打开 Makefile 文件，找到这两行文字:

4. **# ARCH**

5. **?= (SUBARCH)**

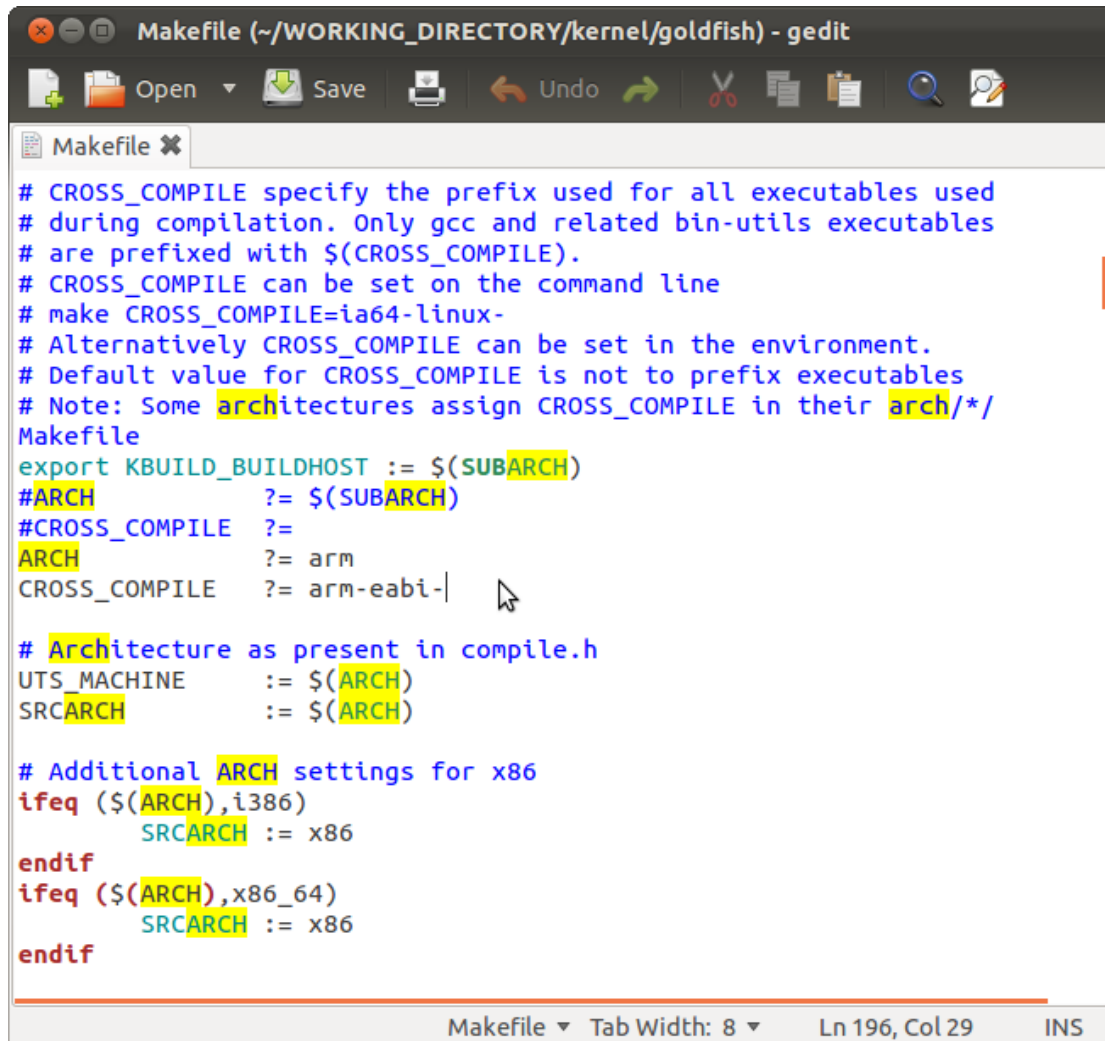
6. **# CROSS\_COMPILE?=**

修改为:

**ARCH ?= arm**

7. **CROSS\_COMPILE**      **?= arm-eabi-**

```
8. $gedit Makefile
```



```
# CROSS_COMPILE specify the prefix used for all executables used
# during compilation. Only gcc and related bin-utils executables
# are prefixed with $(CROSS_COMPILE).
# CROSS_COMPILE can be set on the command line
# make CROSS_COMPILE=ia64-linux-
# Alternatively CROSS_COMPILE can be set in the environment.
# Default value for CROSS_COMPILE is not to prefix executables
# Note: Some architectures assign CROSS_COMPILE in their arch/*/
# Makefile
export KBUILD_BUILDHOST := $(SUBARCH)
#ARCH ?= $(SUBARCH)
#CROSS_COMPILE ?=
ARCH ?= arm
CROSS_COMPILE ?= arm-eabi-

# Architecture as present in compile.h
UTS_MACHINE := $(ARCH)
SRCARCH := $(ARCH)

# Additional ARCH settings for x86
ifeq ($(ARCH),i386)
    SRCARCH := x86
endif
ifeq ($(ARCH),x86_64)
    SRCARCH := x86
endif
```

注意:ARCH?=arm,的 arm 后边不要有空格,不然就会出现如下类似错误:

Make:...../kernel/goldfish/arch/arm: Is a directory. Stop.

关闭 gedit,接下来就开始 make 了,执行如下指令:

1. \$ make goldfish\_armv7\_defconfig
2. \$ make

注 用\$make goldfish\_defconfig这样配置也可以编译通过,模拟器也可以启动,但是 Android 的开机画机就显示不了,\$adb shell 也死活连不上,原因就是在这个 goldfish\_defconfig 这个配置文件问题.

注意: 编译过程如有错误请到我的[编译错误集结篇](#)上查看!

提示:

\$make goldfish\_armv7\_defconfig 指令的意思是将目录 WORKING\_DIRECTORY/kernel/goldfish/arch/arm/configs/下的 goldfish\_armv7\_defconfig 文件内的 Kconfig 配置内容复制到 WORKING\_DIRECTORY/kernel/goldfish/ 目录下的 .config 文件中, .config 文件是一个隐藏目录,保存着各个目录下 Kconfig 文件的配置. 最终结果如下图所示:

```
arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY/kernel/goldfish
UPD      include/linux/compile.h
CC       init/version.o
LD       init/built-in.o
LD       .tmp_vmlinux1
KSYM     .tmp_kallsyms1.S
AS       .tmp_kallsyms1.o
LD       .tmp_vmlinux2
KSYM     .tmp_kallsyms2.S
AS       .tmp_kallsyms2.o
LD       vmlinux
SYSMAP   System.map
SYSMAP   .tmp_System.map
OBJCOPY  arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS       arch/arm/boot/compressed/head.o
GZIP     arch/arm/boot/compressed/piggy.gz
AS       arch/arm/boot/compressed/piggy.o
CC       arch/arm/boot/compressed/misc.o
LD       arch/arm/boot/compressed/vmlinux
OBJCOPY  arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$ ls
```

这样就表示编译成功了。

```
1. $ ls arch/arm/boot/
```

可以看到 zImage 文件

```
arthur@arthur-System-Product-Name: ~/WORKING_DIRECTORY/kernel/goldfish
LD       .tmp_vmlinux1
KSYM     .tmp_kallsyms1.S
AS       .tmp_kallsyms1.o
LD       .tmp_vmlinux2
KSYM     .tmp_kallsyms2.S
AS       .tmp_kallsyms2.o
LD       vmlinux
SYSMAP   System.map
SYSMAP   .tmp_System.map
OBJCOPY  arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS       arch/arm/boot/compressed/head.o
GZIP     arch/arm/boot/compressed/piggy.gz
AS       arch/arm/boot/compressed/piggy.o
CC       arch/arm/boot/compressed/misc.o
LD       arch/arm/boot/compressed/vmlinux
OBJCOPY  arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$ ls arch/a
rm/boot/
bootp  compressed  Image  install.sh  Makefile  zImage
arthur@arthur-System-Product-Name:~/WORKING_DIRECTORY/kernel/goldfish$
```

第三步:在模拟器中启动编译好的内核

```
1. $ export PATH=$PATH:~/WORKING_DIRECTORY/out/host/linux-x86/bin
2. $ export ANDROID_PRODUCT_OUT=~/WORKING_DIRECTORY/out/target/product/gen
eric
```

```
3. $ emulator -kernel ~/WORKING_DIRECTORY/kernel/goldfish/arch/arm/boot/zImage &
```

注意：我们还可以加载现前编译后的 system.img ramdisk.img userdata.img 等镜像来测试。

-system /root/WORKING\_DIRECTORY/out/target/product/generic/system.img

-data /root/WORKING\_DIRECTORY/out/target/product/generic/userdata.img

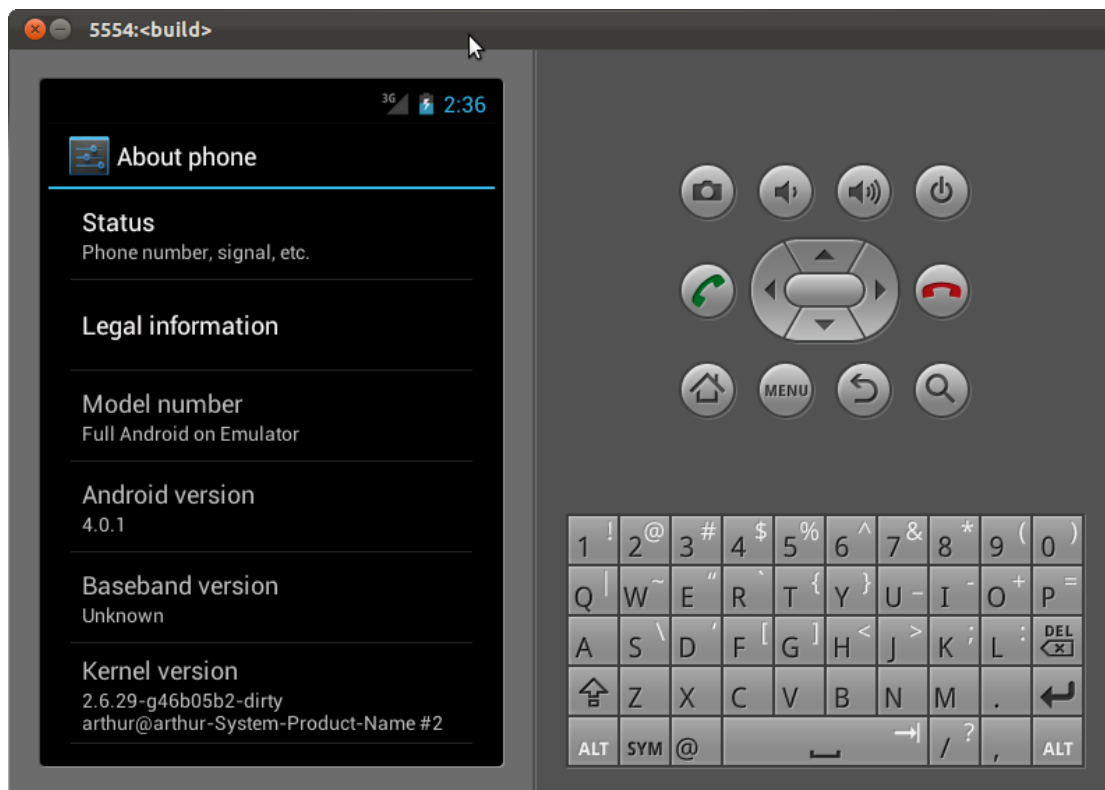
-ramdisk /root/WORKING\_DIRECTORY/out/target/product/generic/ramdisk.img

模拟器启动界面：



进入模拟器从设置里看版本信息：





从上图可以看出当前 Android 版本是 4.0.1,内核版本是 2.6.29,说明成功了



到这里。教程基本结束下面是位开发者讲到的开发环境搭建及 sdk 的安装及使用。

编译 sdk 因为下载的源码里面包含 SDK 的源码我们直接编译即可，想下载的可以上官网下载，这里我们主要讲的是底层源码编起。

进入 WORKING\_DIRECTORY

```
$cd WORKING_DIRECTORY
```

```
$make PRODUCT-sdk-sdk
```

等待 30 多分钟后编译生成后的 SDK 存放在 out/host/linux-x86/sdk/目录下。

（建议将 SDK 备份并牢牢记住 sdk 的存放位置）

下面可以在 sdk 里加载开发环境。

请看下一章！