SIM8500平台编译文档

智能模组

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
技术支持邮箱：support@simcom.com
官网：www.simcom.com

名称：	SIM8500平台编译文档
版本：	1.00
日期：	2022.03.10
状态：	已发布

版权声明

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
邮箱：simcom@simcom.com
官网：www.simcom.com

了解更多资料，请点击以下链接：
http://cn.simcom.com/download/list-230-cn.html

技术支持，请点击以下链接：
http://cn.simcom.com/ask/index-cn.html 或发送邮件至 support@simcom.com

关于文档

版本历史

版本	日期	作者	备注
1.00	2022.3.10	yubin chen	第一版

适用范围

本文档适用于 SIMCom SIM8500 系列。

目录

1 要求

1.1 硬件环境要求

CPU: i5 8 核，**推荐** i7
内存: 至少需要 16 GB，**推荐** 64GB
硬盘: 1T，**推荐**固态硬盘

1.2 软件环境要求

1.2.1 操作系统要求
建议选择 64 位 16.04 ubuntu 系统。
查看 ubuntu 的具体版本号命令为:
sb_release –a

1.2.2 软件包
额外需要的软件包主要有：

- python.org 中提供的 Python 2.6 - 2.7。
- gnu.org 中提供的 GNU Make 3.81 - 3.82。
- git-scm.com 中提供的 Git 1.7 或更高版本。
- openjdk-8
- OpenSSL 1.0.1f 6 Jan 2014

说明
ubuntu10.04～14.04 版本或者更高版本 ubuntu18.04 也支持，但是不同版本依赖的编译支持工具略有差异。

2 搭建编译环境

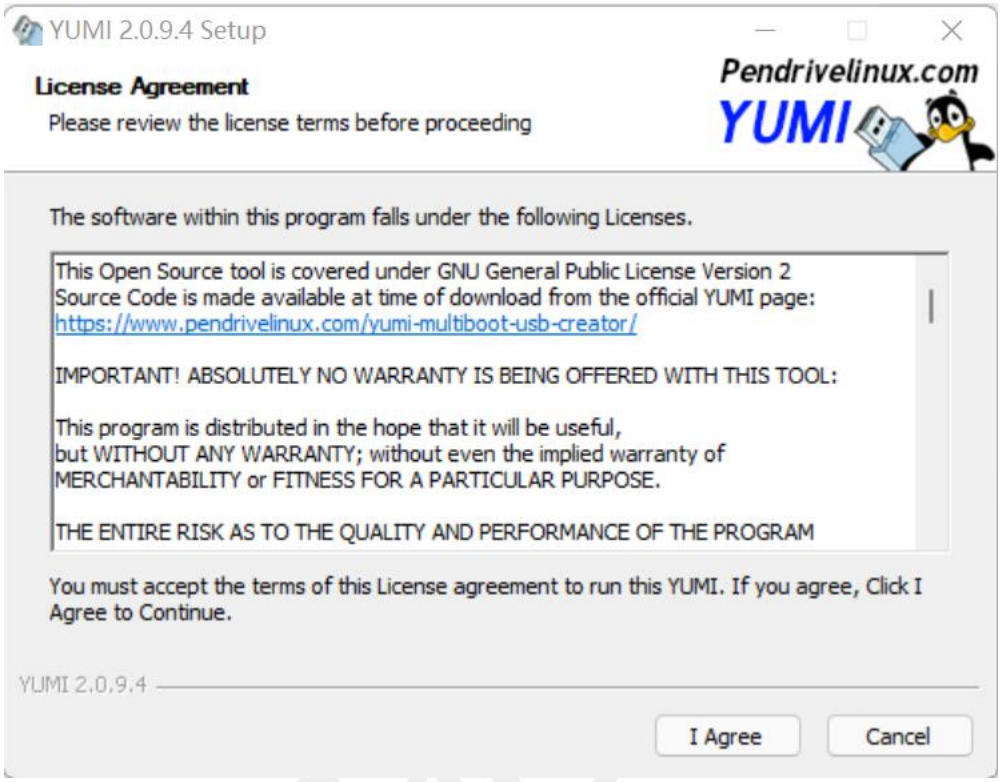
2.1 准备启动盘

2.1.1 镜像和工具

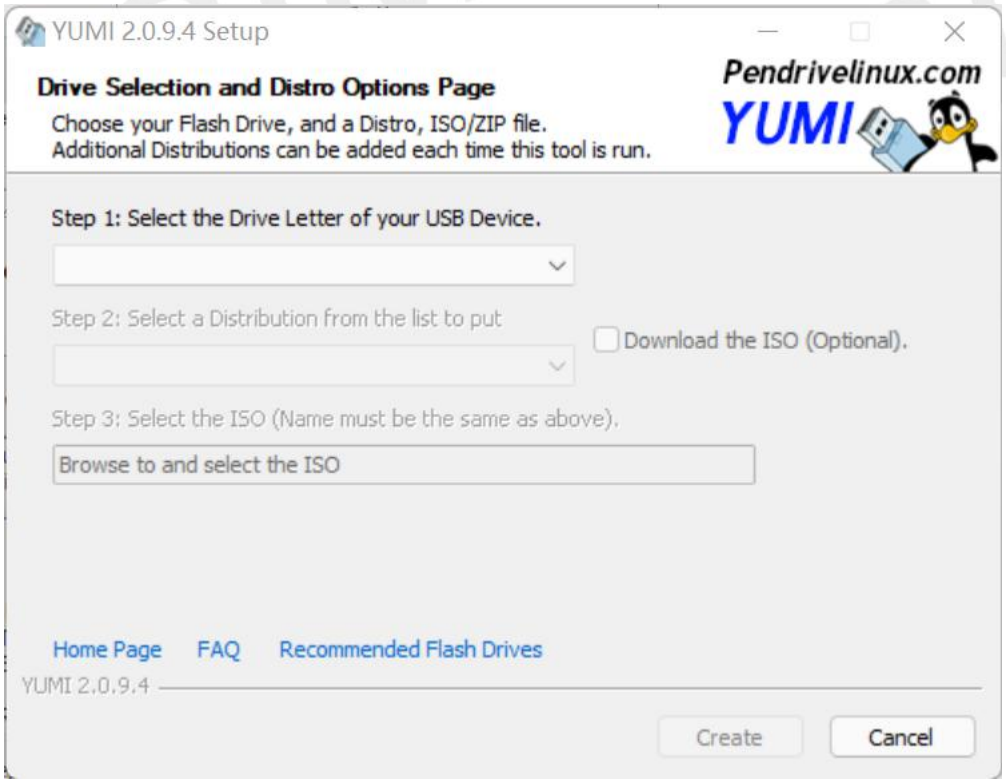
iso 镜像：
http://releases.ubuntu.com/xenial/ubuntu-16.04.7-desktop-amd64.iso
制作工具：
https://www.pendrivelinux.com/downloads/YUMI/YUMI-2.0.9.4.exe

2.1.2 制作启动盘

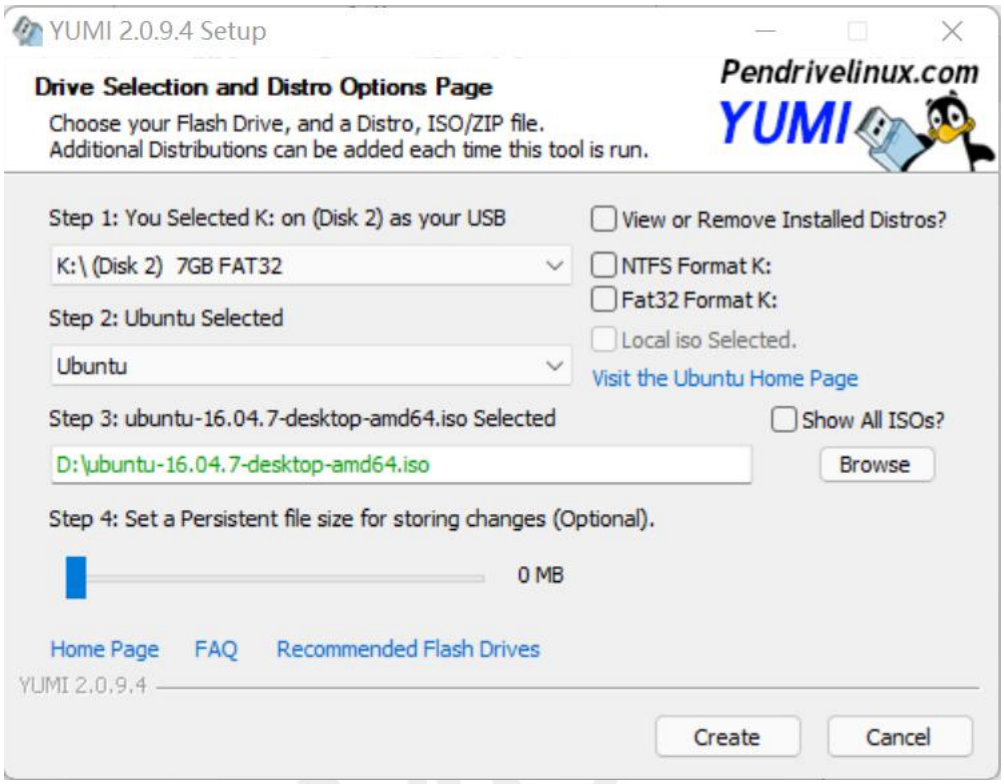
插入一个 U 盘，U 盘上至少有 2GB 可用空间
启动 YUMI



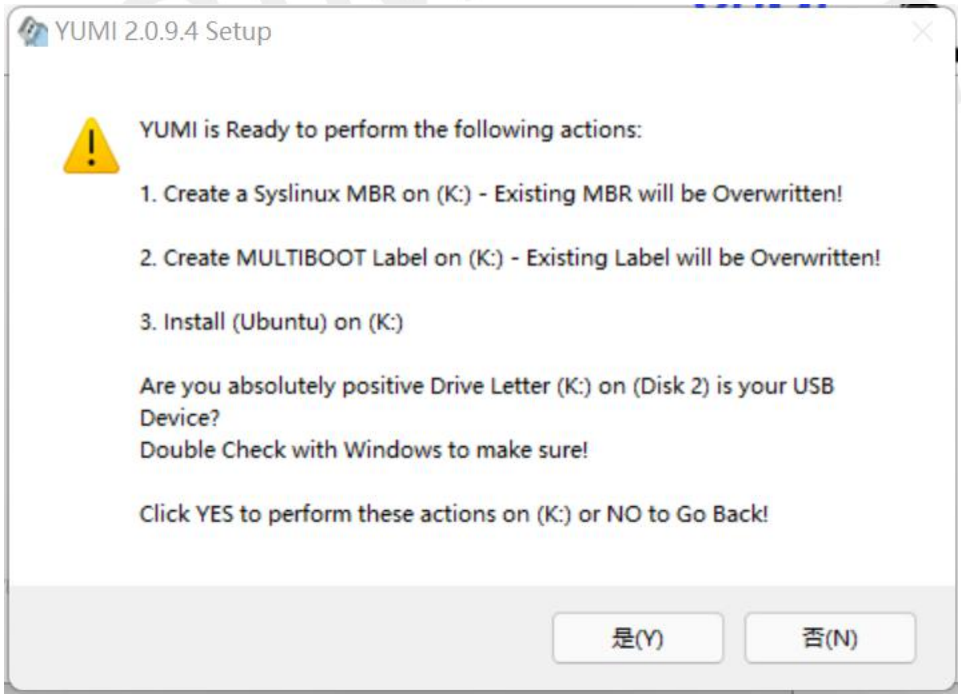
选择 I Agree



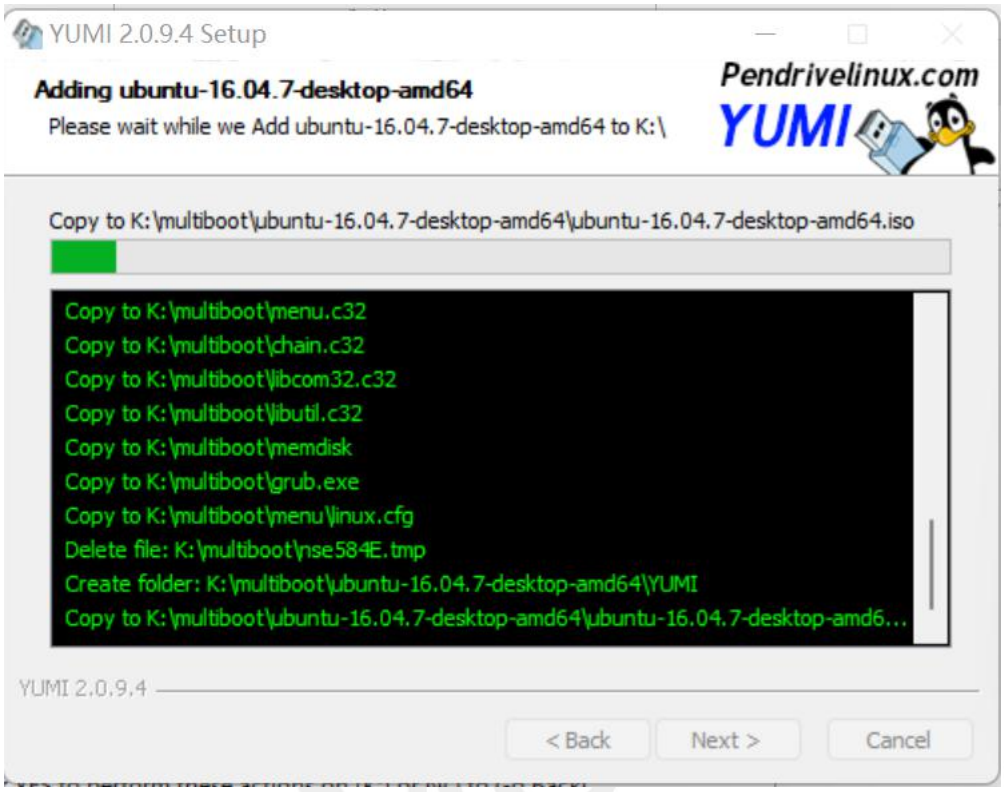
选好 U 盘、系统类型、iso 镜像

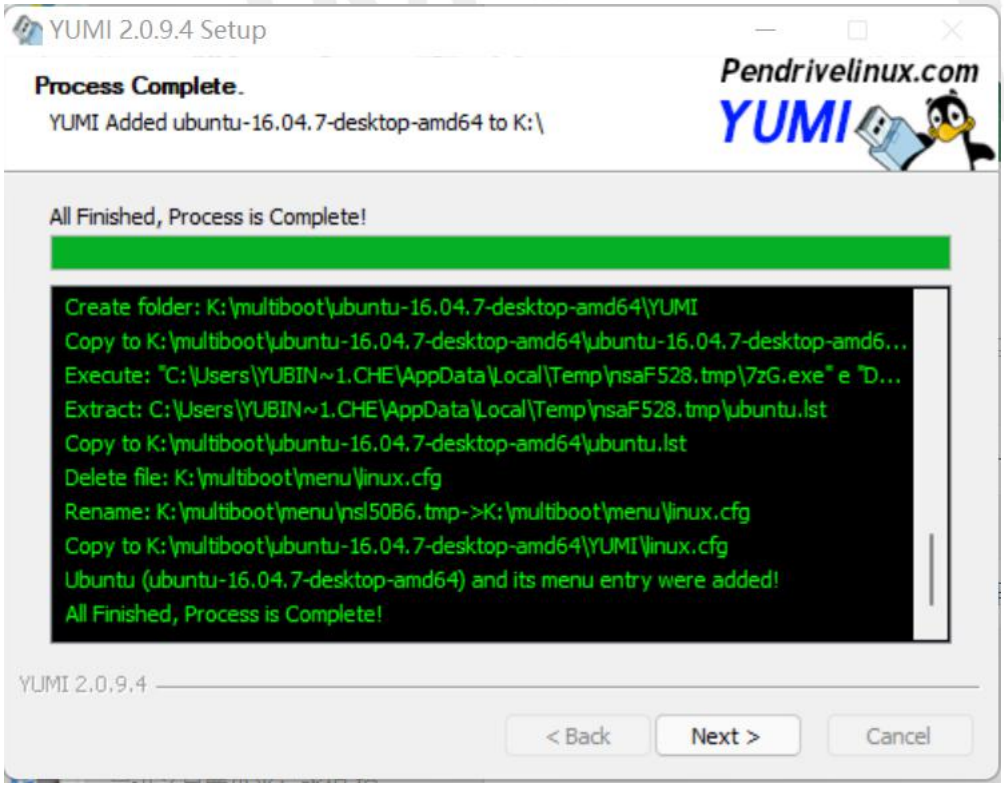

YUMI 2.0.9.4 Setup
Drive Selection and Distro Options Page
Choose your Flash Drive, and a Distro, ISO/ZIP file.
Additional Distributions can be added each time this tool is run.
Pendrivelinux.com
YUMI
Step 1: You Selected K: on (Disk 2) as your USB
K:\ (Disk 2) 7GB FAT32
View or Remove Installed Distros?
NTFS Format K:
Fat32 Format K:
Step 2: Ubuntu Selected
Ubuntu
Local iso Selected.
Visit the Ubuntu Home Page
Step 3: ubuntu-16.04.7-desktop-amd64.iso Selected
Show All ISOs?
D:\ubuntu-16.04.7-desktop-amd64.iso
Browse
Step 4: Set a Persistent file size for storing changes (Optional).
0 MB
Home Page
FAQ
Recommended Flash Drives
YUMI 2.0.9.4
Create
Cancel

选择 Create


YUMI 2.0.9.4 Setup
YUMI is Ready to perform the following actions:
1. Create a Syslinux MBR on (K:) - Existing MBR will be Overwritten!
2. Create MULTIBOOT Label on (K:) - Existing Label will be Overwritten!
3. Install (Ubuntu) on (K:)
Are you absolutely positive Drive Letter (K:) on (Disk 2) is your USB Device?
Double Check with Windows to make sure!
Click YES to perform these actions on (K:) or NO to Go Back!
是(Y)
否(N)

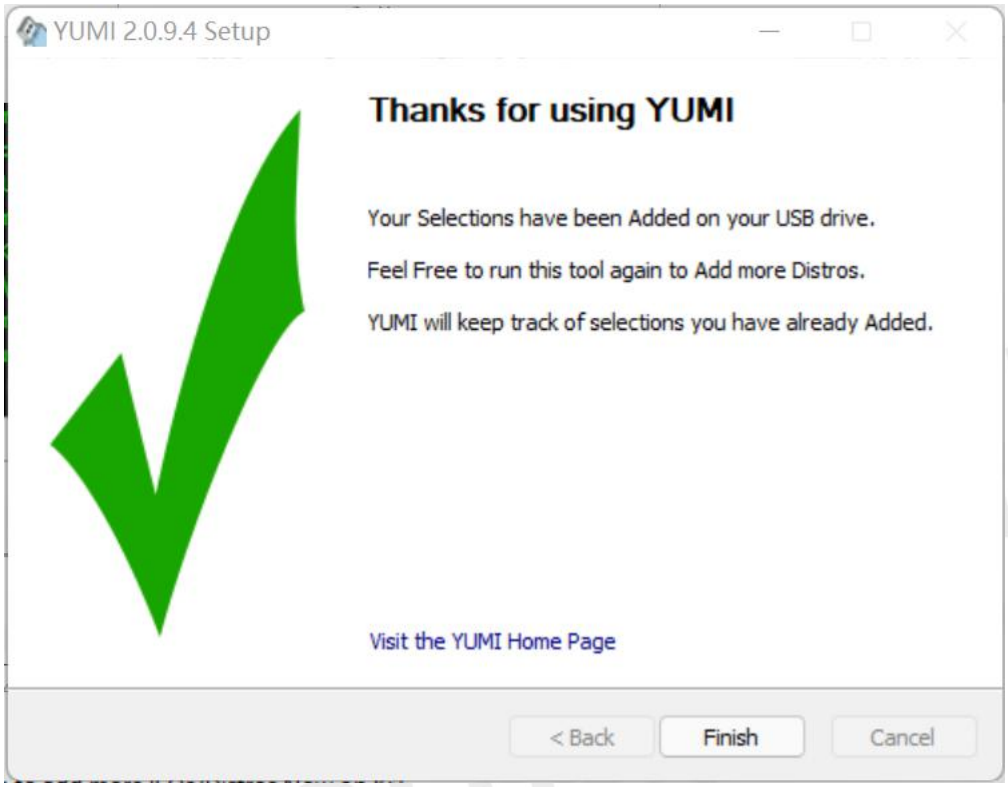
选 **是**


YUMI 2.0.9.4 Setup
Adding ubuntu-16.04.7-desktop-amd64
Please wait while we Add ubuntu-16.04.7-desktop-amd64 to K:\
Pendrivelinux.com
YUMI
Copy to K:\multiboot\ubuntu-16.04.7-desktop-amd64\ubuntu-16.04.7-desktop-amd64.iso
Copy to K:\multiboot\menu.c32
Copy to K:\multiboot\chain.c32
Copy to K:\multiboot\libcom32.c32
Copy to K:\multiboot\libutil.c32
Copy to K:\multiboot\memdisk
Copy to K:\multiboot\grub.exe
Copy to K:\multiboot\menu\linux.cfg
Delete file: K:\multiboot\nse584E.tmp
Create folder: K:\multiboot\ubuntu-16.04.7-desktop-amd64\YUMI
Copy to K:\multiboot\ubuntu-16.04.7-desktop-amd64\ubuntu-16.04.7-desktop-amd6...
YUMI 2.0.9.4
< Back
Next >
Cancel


YUMI 2.0.9.4 Setup
Process Complete.
YUMI Added ubuntu-16.04.7-desktop-amd64 to K:\
Pendrivelinux.com
YUMI
All Finished, Process is Complete!
Create folder: K:\multiboot\ubuntu-16.04.7-desktop-amd64\YUMI
Copy to K:\multiboot\ubuntu-16.04.7-desktop-amd64\ubuntu-16.04.7-desktop-amd6...
Execute: "C:\Users\YUBIN~1.CHE\AppData\Local\Temp\nsaF528.tmp\7zG.exe" e "D...
Extract: C:\Users\YUBIN~1.CHE\AppData\Local\Temp\nsaF528.tmp\ubuntu.lst
Copy to K:\multiboot\ubuntu-16.04.7-desktop-amd64\ubuntu.lst
Delete file: K:\multiboot\menu\linux.cfg
Rename: K:\multiboot\menu\nsl50B6.tmp->K:\multiboot\menu\linux.cfg
Copy to K:\multiboot\ubuntu-16.04.7-desktop-amd64\YUMI\linux.cfg
Ubuntu (ubuntu-16.04.7-desktop-amd64) and its menu entry were added!
All Finished, Process is Complete!
YUMI 2.0.9.4
< Back
Next >
Cancel

选 Next

选 **否(N)**



完成！

2. 2 安装系统

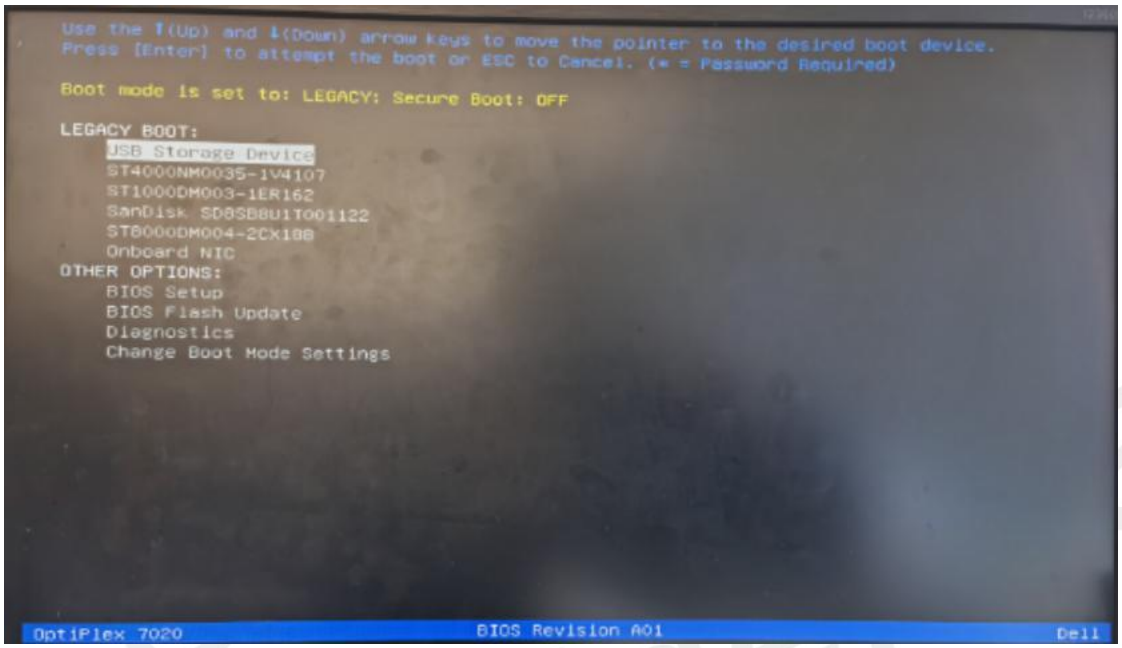
2.2.1 硬盘分区

建议：
从硬盘开始分配 50GB，安装系统
从硬盘末端分配 20GB（大小为物理内存的 1.5 倍），配成 SWAP
其余空间可分配为资料区
说明
这样做的好处是：即使今后系统出了问题，重新安装系统也不影响其他分区数据

2.2.2 BIOS 设置

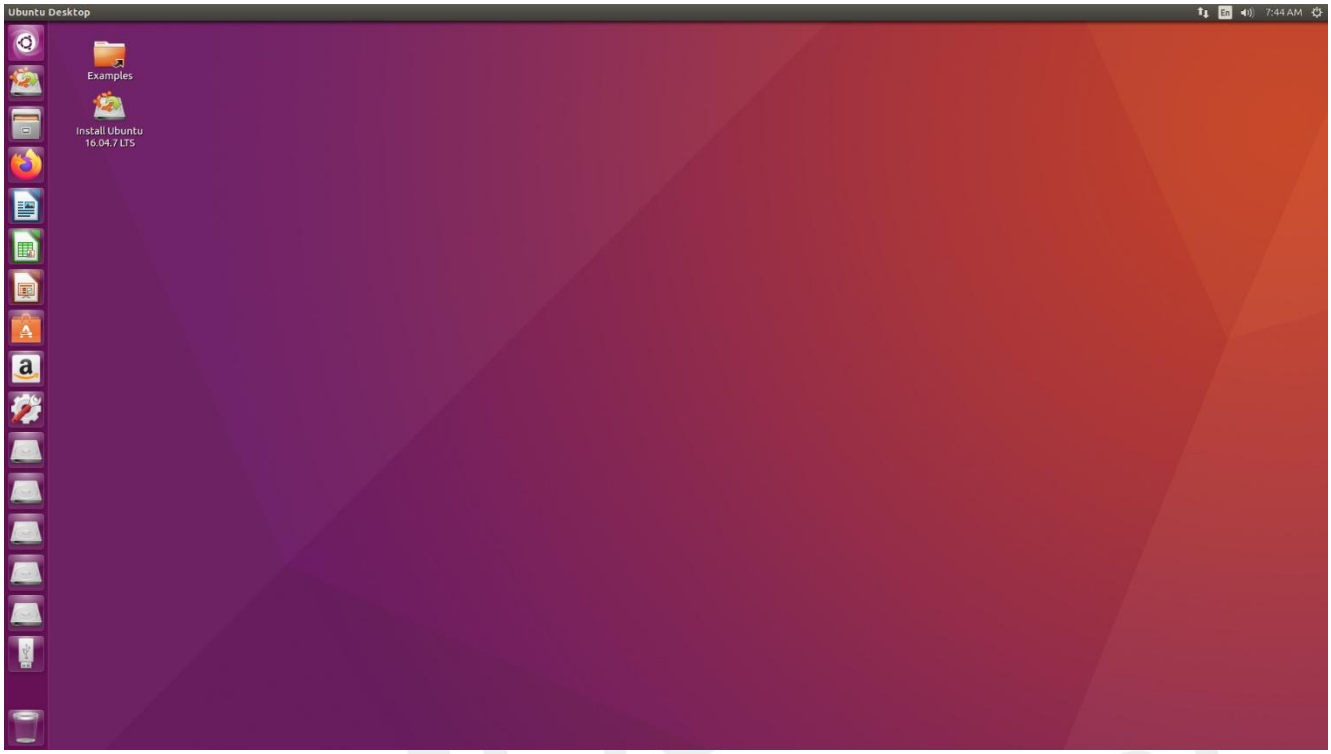
安装前，需要设置下电脑 BIOS。
开机后按 F2，进入 BIOS 设置界面，修改 2 个配置：
Boot mode：LEGACY
Secure Boot：OFF

启动盘插到电脑接口，开机后按 F12，选 USB Storage Device

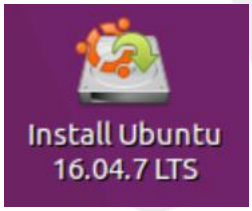


2.2.3 开始安装

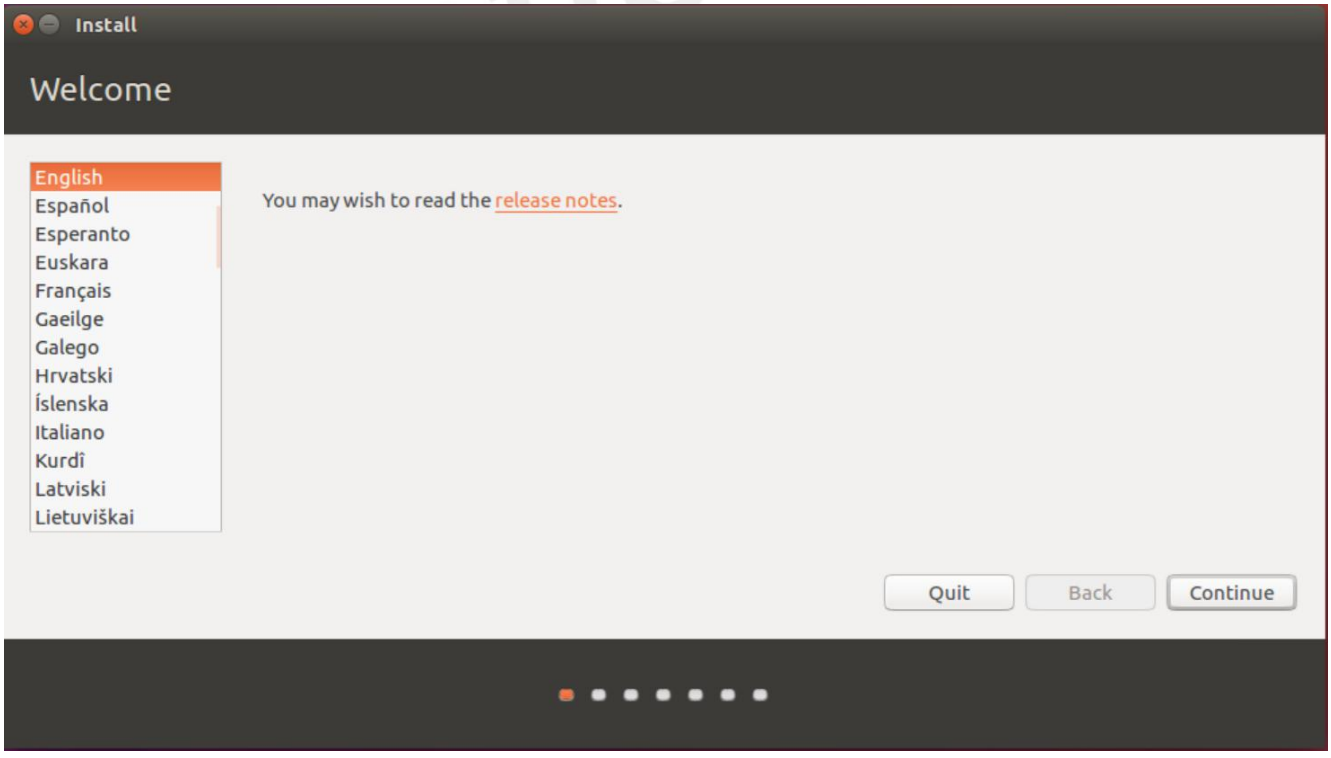
下图中，第一个选项是从硬盘启动，第二个选项是从启动盘启动。选第二个



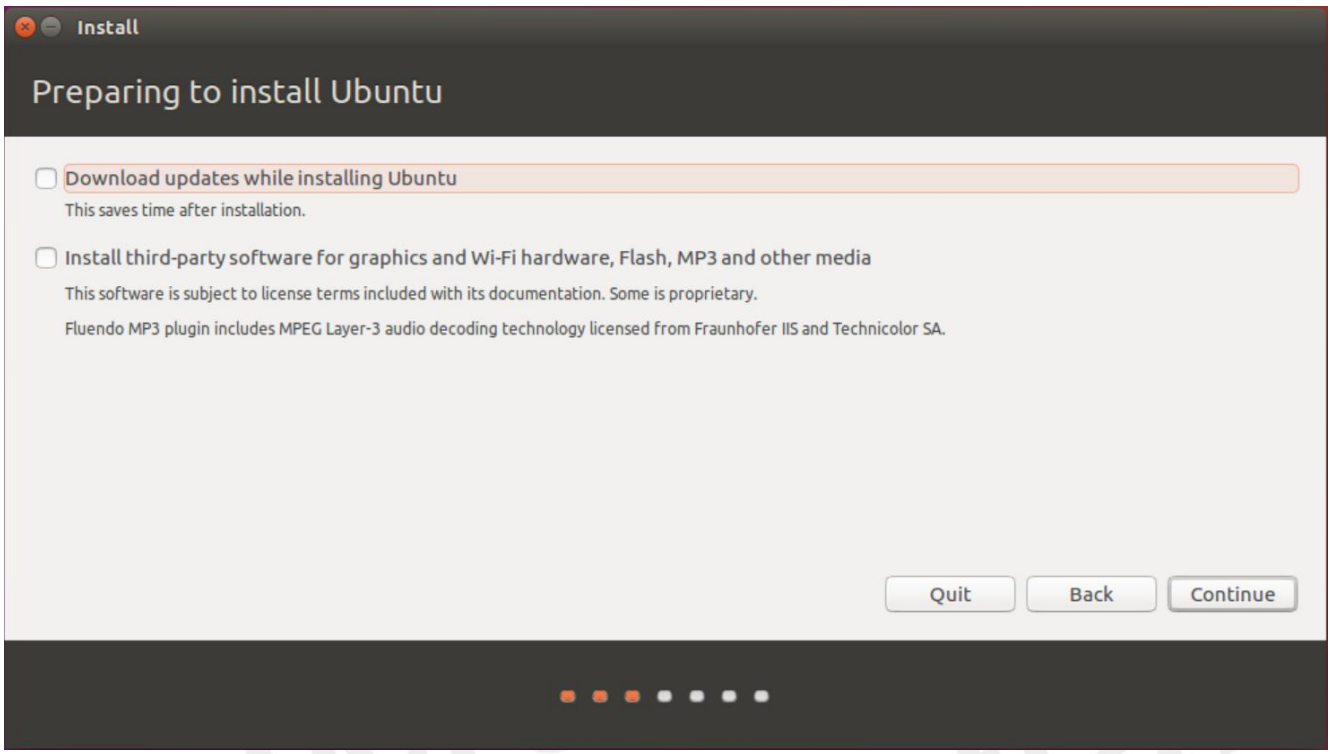
选择桌面的 **Install Ubuntu 16.04.6 LTS**



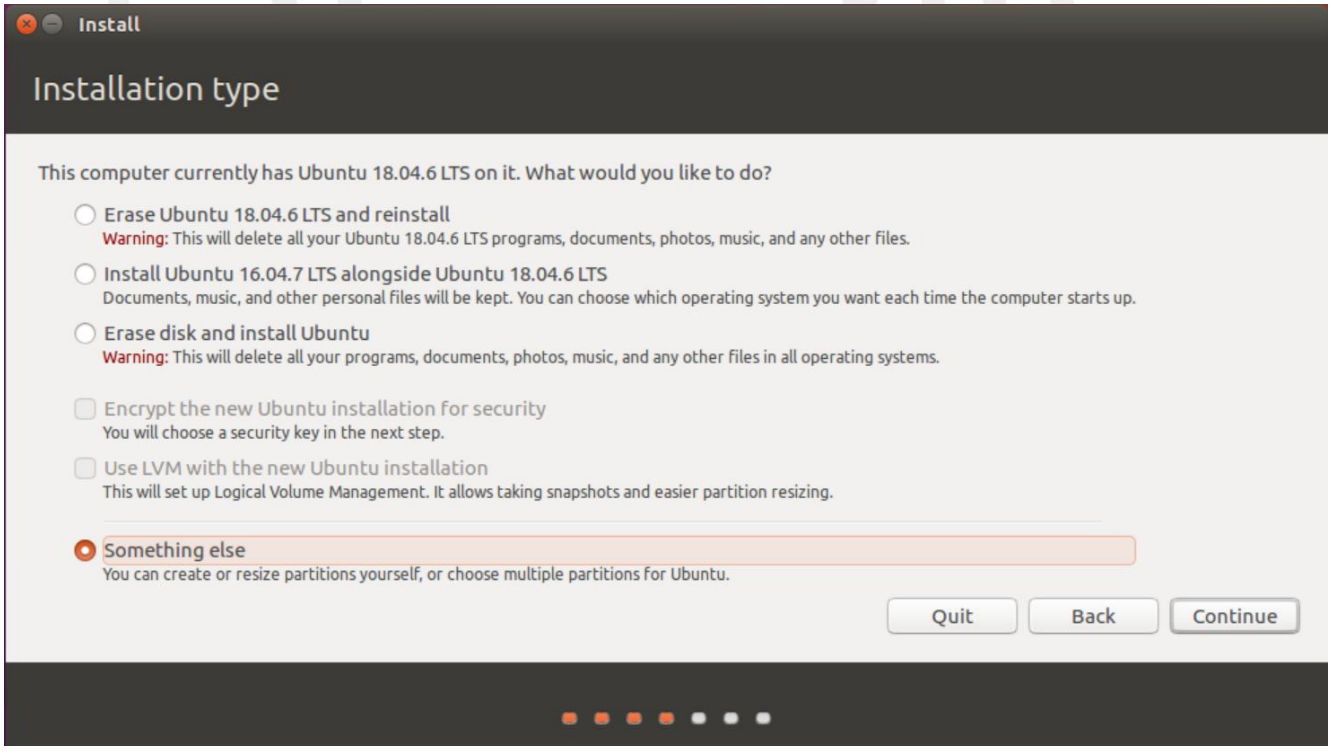
第一步：系统语言，默认英文



第二步：保持默认，不勾选

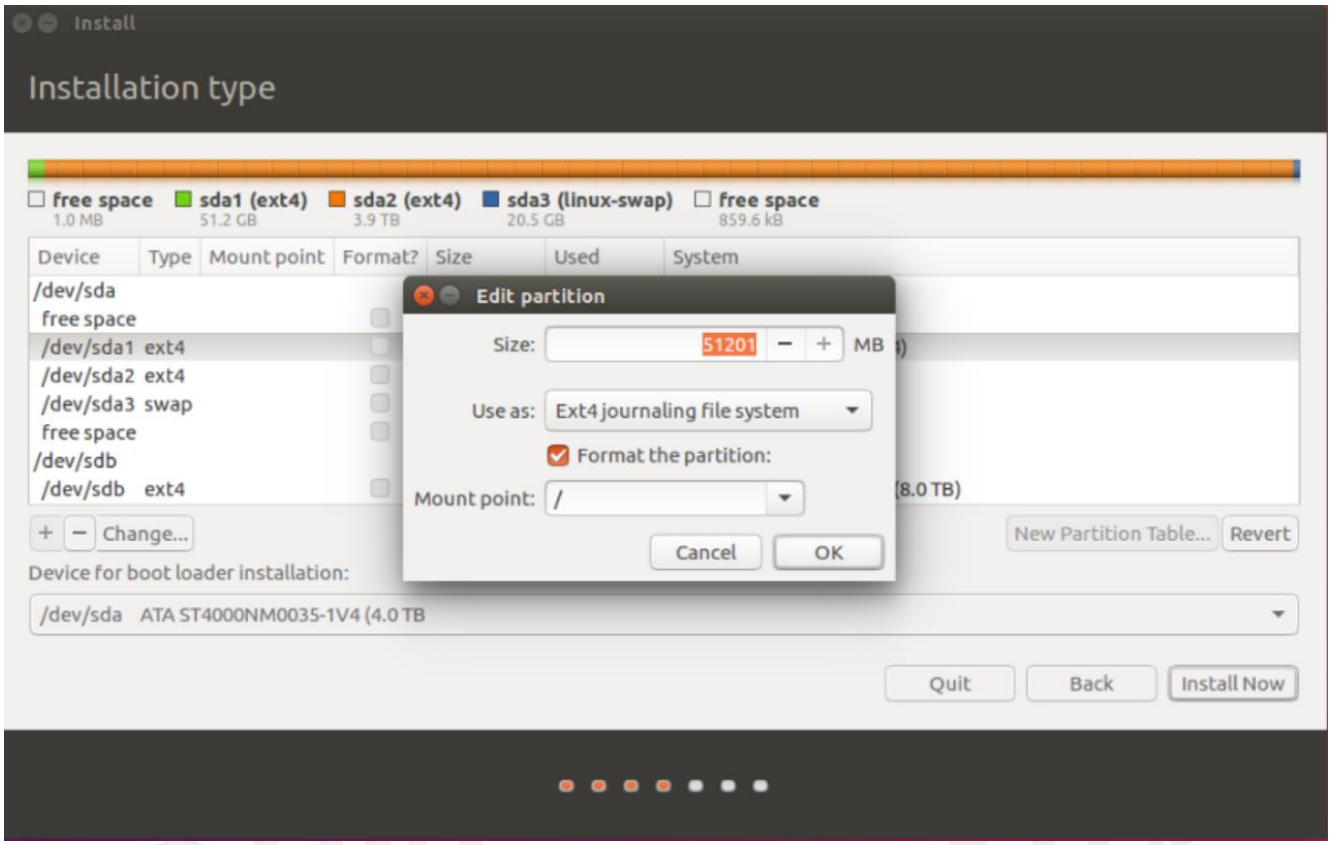


第三步：安装类型，建议选最后一项 **Something else**

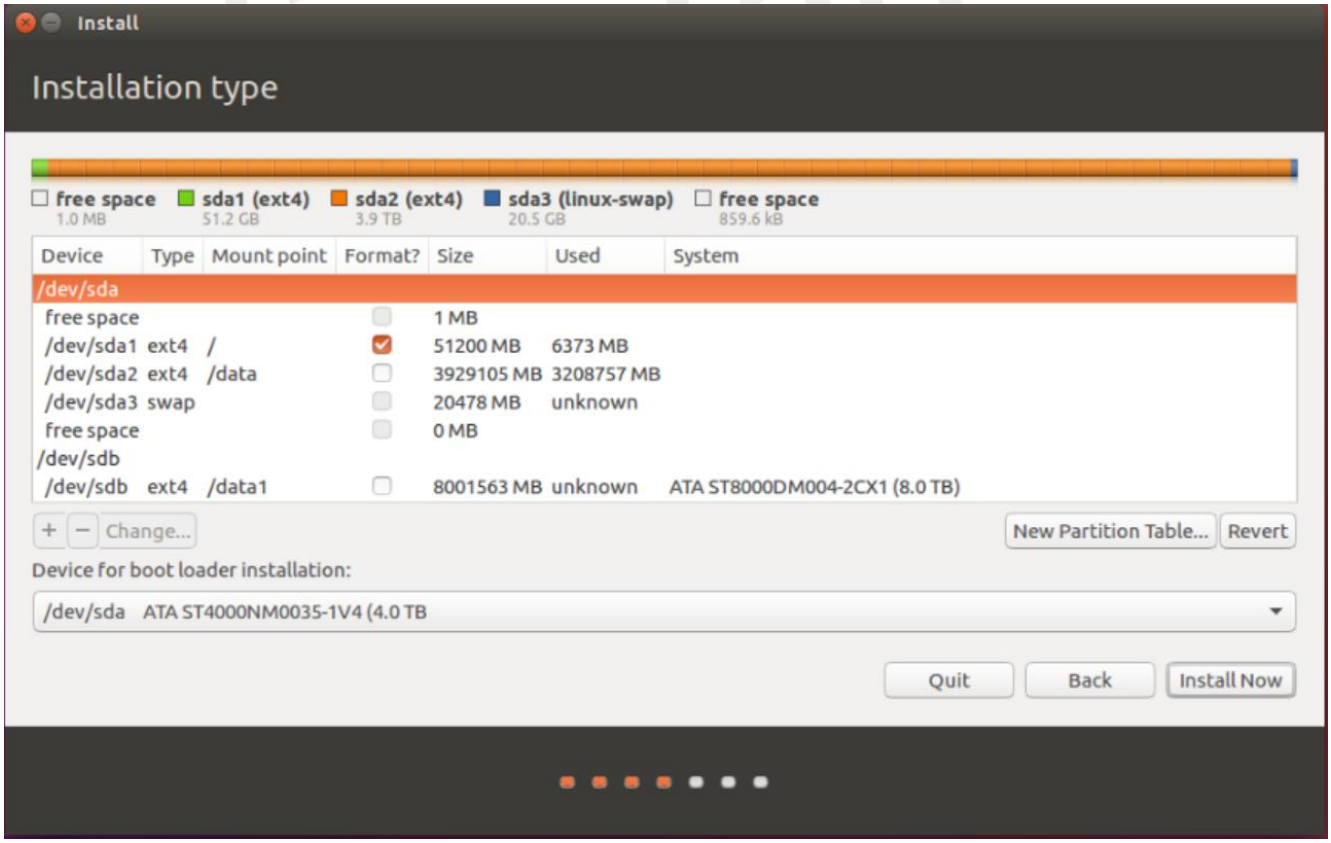


第四步：选择对应分区、文件系统格式、挂载节点

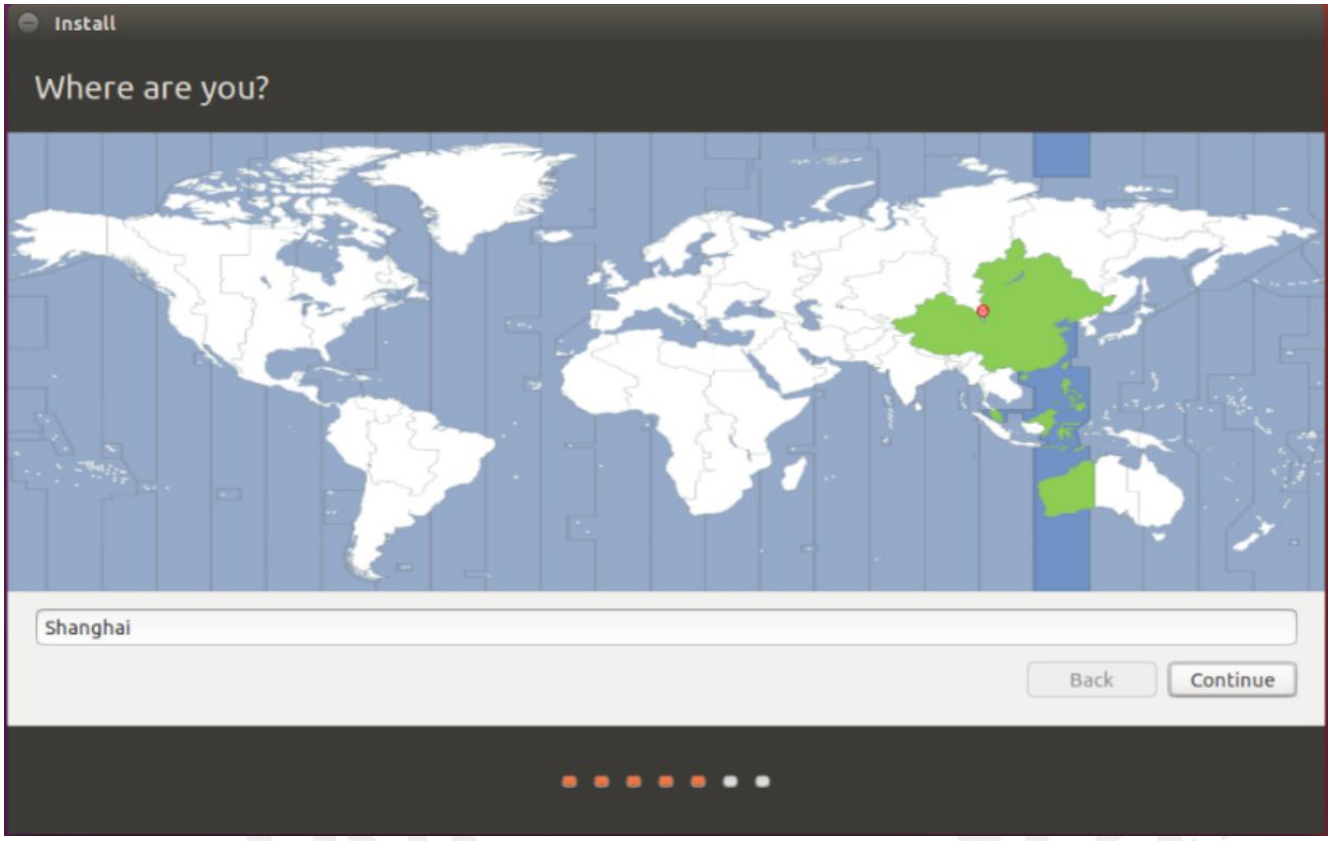
系统分区，需要勾选格式化



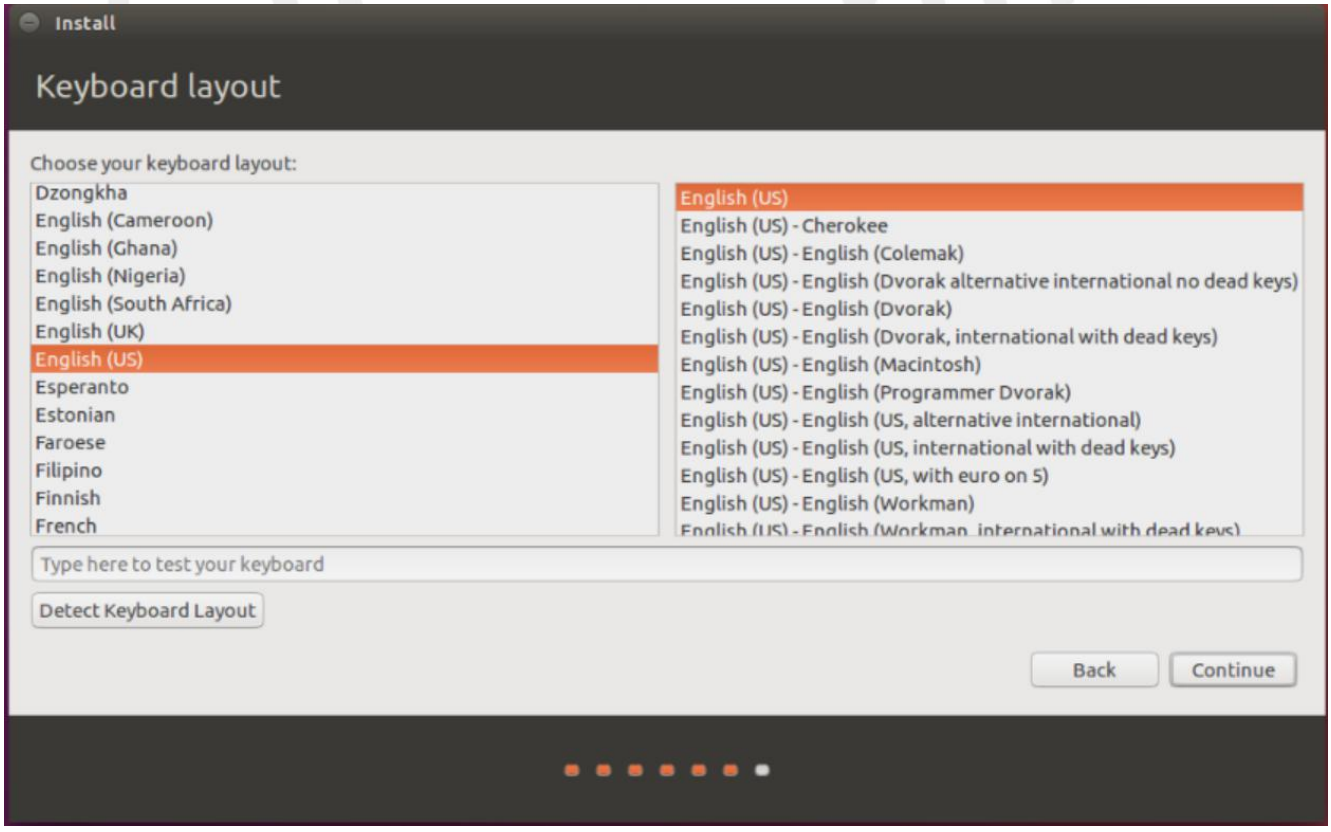
配置完成后的各分区



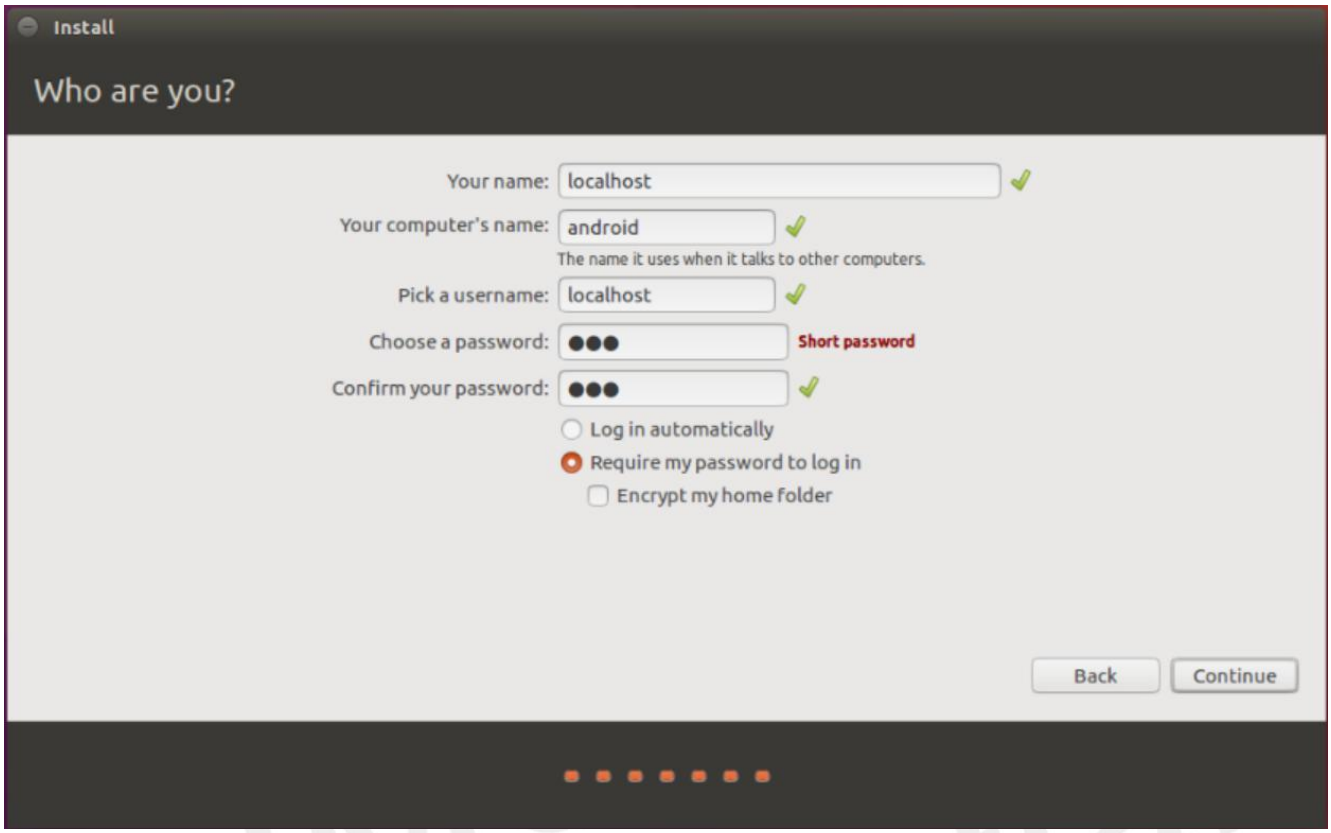
第五步：选择系统时区 Shanghai



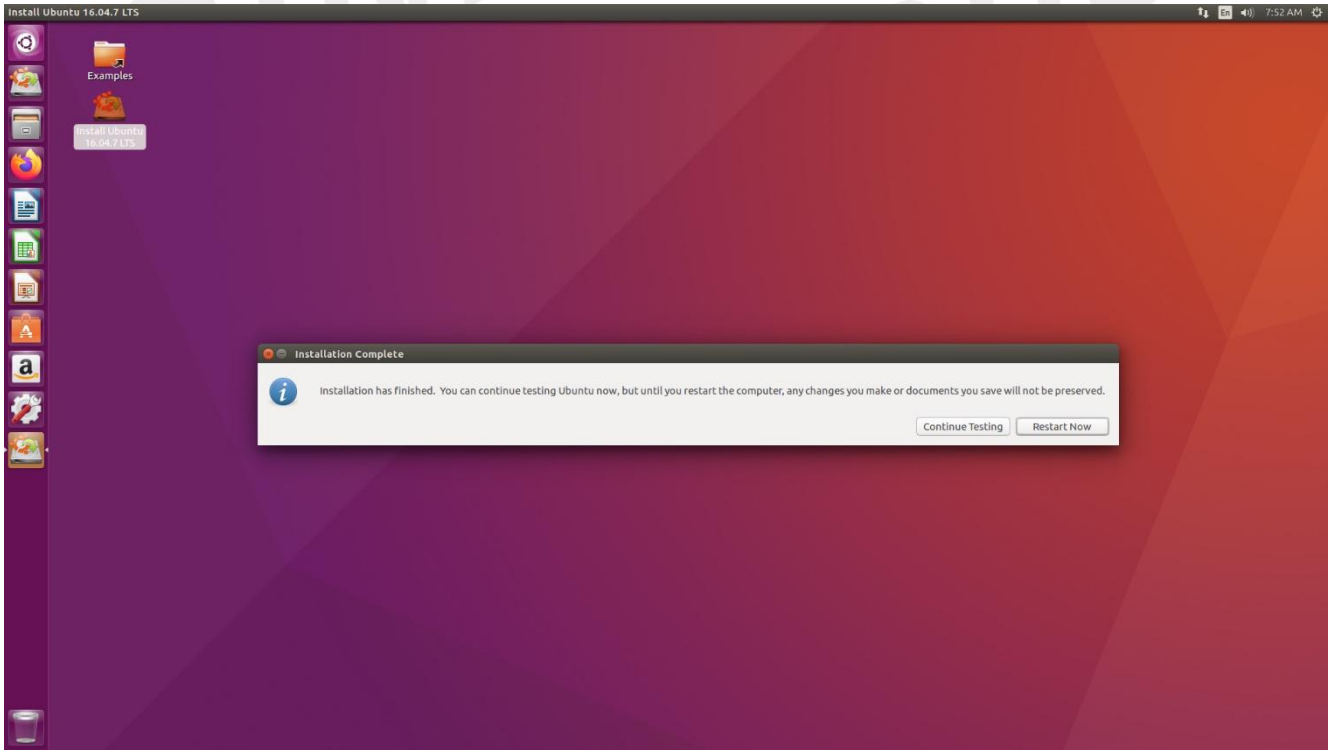
第六步：键盘布局，默认 English(US)



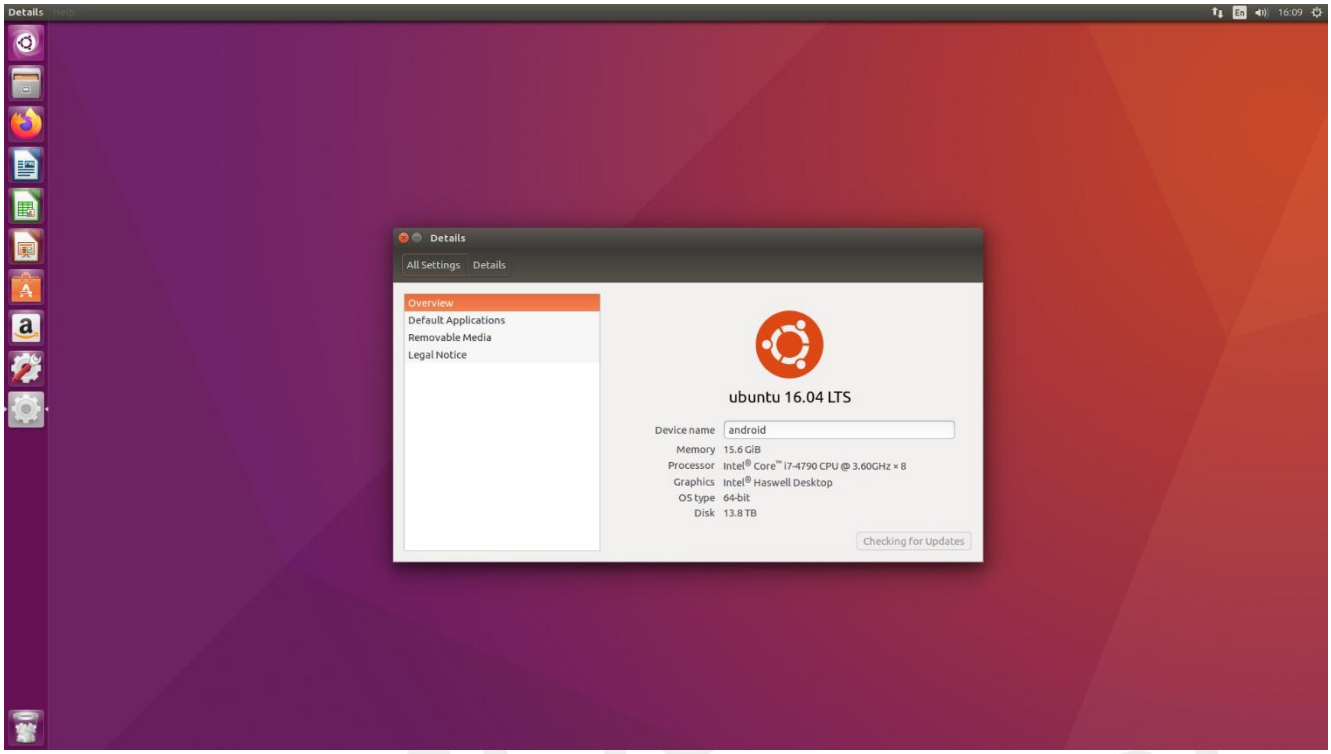
第七步：输入用户名、主机名、密码（根据需要自己设置）



安装完毕，选 Restart Now 重启



登入界面，输入密码即可进入系统



2.3 软件包安装

从桌面左上角，输入 Terminal 打开一个终端，依次输入指令：
$ sudo apt-get update
$ sudo apt-get install git-core gnupg flex bison gperf build-essential zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z-dev ccache libgl1-mesa-dev libxml2-utils xsltproc unzip openjdk-8-jdk libssl-dev

2.4 创建 ssh key

在 Terminal 终端，输入
ssh-keygen -t rsa -C <邮箱>
按回车确认（不需输入密码）

2.5 配置 git 账号

$ git config --global user.name "<用户名>"
$ git config --global user.email <邮箱>

2.6 配置 bash

$ sudo dpkg-reconfigure dash
选 **<No>**，回车确认

2.7 安装 make

下载链接：http://ftp.gnu.org/pub/gnu/make/make-3.81.tar.bz2
解压后依次执行下面指令：

```
$ ./configure
$ make
$ sudo make install
$ sudo mv /usr/bin/make /usr/bin/make-4.1
$ sudo ln -sf /usr/local/bin/make /usr/bin/make
```

2.8 安装 openssl

下载链接：https://www.openssl.org/source/old/1.0.1/openssl-1.0.1f.tar.gz
解压后依次执行下面指令：

```
$ ./config --prefix=/usr/local --openssldir=/usr/local/openssl
$ sudo make
$ sudo mv /usr/bin/pod2man /usr/bin/pod2man_bak
$ sudo make install
$ sudo mv /usr/bin/pod2man_bak /usr/bin/pod2man
$ sudo mv /usr/bin/openssl /usr/bin/openssl.old
$ sudo mv /usr/include/openssl /usr/include/openssl.old
$ sudo ln -sf /usr/local/bin/openssl /usr/bin/openssl
$ sudo ln -sf /usr/local/include/openssl/ /usr/include/openssl
$ openssl version -a
```

3 Android 编译系统

3.1 编译系统简介

要了解 Android 的编译过程，需要知道几个主要的概念。

- Android.mk

整个系统中，包含了大量的模块，每个模块都有一个专门的 Make 文件，这类文件的名称统一为 “Android.mk”，该文件中定义了如何编译当前模块。Build 系统会在整个源码树中扫描名称为 “Android.mk”的文件并根据其中的内容执行模块的编译。

- Android.bp

Android.bp 的出现就是为了替换 Android.mk 文件。bp 跟 mk 文件不同，它是纯粹的配置，没有分支、循环等流程控制，不能做算数逻辑运算。如果需要控制逻辑，那么只能通过 Go 语言编写。

- Ninja

Ninja 是一个编译框架，会根据相应的 ninja 格式的配置文件进行编译，但是 ninja 文件一般不会手动修 改，而是通过将 Android.bp 文件转换成 ninja 格式文件来编译。

- Soong

Soong 类似于之前的 Makefile 编译系统的核心，负责提供 Android.bp 语义解析，并将之转换成 Ninja 文

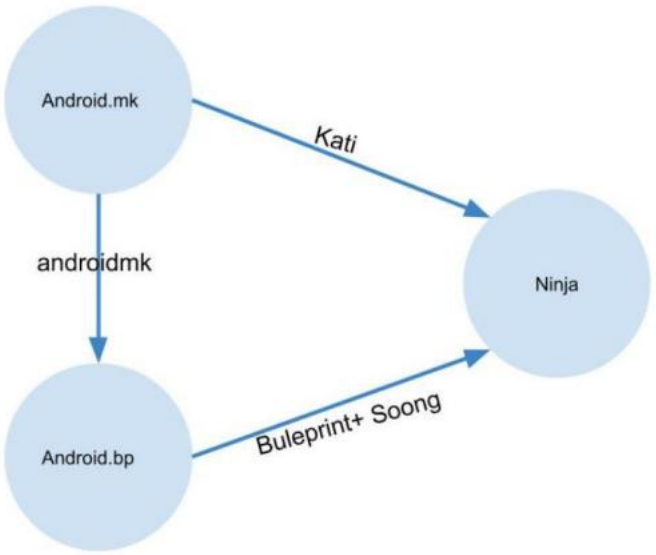
件。Soong 还会编译生成一个 androidmk 命令，用于将 Android.mk 文件转换为 Android.bp 文件，不过 这个转换功能仅限于没有分支、循环等流程控制的 Android.mk 才有效。

■ Blueprint

Blueprint 是生成、解析 Android.bp 的工具，是 Soong 的一部分。Soong 负责 Android 编译而设计的工 具，而 Blueprint 只是解析文件格式，Soong 解析内容的具体含义。Blueprint 和 Soong 都是由 Golang 写的项目，从 Android 7.0，prebuilts/go/目录下新增 Golang 所需的运行环境，在编译时使用。

■ Kati

kati 是专为 Android 开发的一个基于 Golang 和 C++的工具，主要功能是把 Android 中的 Android.mk 文件转换成 Ninja 文件。代码路径是 build/kati/，编译后的产物是 ckati。 这六者的关系可由下图表示：



3.2 Android.mk 语法

下面是 Android.mk 具体的一些语法说明：

LOCAL_PATH := $(call my-dir)

LOCAL_PATH 是必须有的，返回当前目录路径。

LOCAL_MODULE_TAGS ：=user \ eng \ tests \ optional

LOCAL_MODULE_TAGS 指定当前模块的标志，user 表示只在 user 版本才编译；eng 表示只在 eng 版本才编译；tests 表示只在 test 版本才编译；optional 表示在所有版本中都会编译。

include $(CLEAR_VARS)

CLEAR_VARS 指向一个特殊的 GNU Makefile，会清除许多 LOCAL_XXX 变量，但不会清除 LOCAL_PATH。

LOCAL_MODULE := XXX//

LOCAL_MODULE_FILENAME := mymodulename

LOCAL_MODULE 表示模块名，必须唯一，不含空格，构建系统会自动添加前后缀（lib，.a，.so）；

LOCAL_MODULE_FILENAME 则可以自定义模块名，不会自动添加前后缀。

LOCAL_SRC_FILES := \

XXX.cpp \

LOCAL_SRC_FILES 列举源文件，也可以通过$(call all-java-files-under,)获取指定目录下的所有 Java 文件，或通过$(call all-c-files-under,)来获取指定目录下的所有 C 语言文件。

LOCAL_RESOURCE_DIR := $(addprefix $(LOCAL_PATH)/, $(res_dir))
LOCAL_RESOURCE_DIR 导入相关的 res 资源目录。

LOCAL_USE_AAPT2 := true
AAPT（Android Assert Packaging Tool）是编译和打包资源的工具，AAPT2 是在 APPT 基础上做了优 化，AAPT2 将原先的资源编译打包过程拆分成编译和链接两个部分。

LOCAL_CPP_EXTENSION := .cxx
LOCAL_CPP_EXTENSION 可以为 C++源文件指定.cpp 以外的文件扩展名。

LOCAL_CPP_FEATURES := rtti
可以指定代码依赖的特定 C++功能。
LOCAL_C_INCLUDES := \
$(LOCAL_PATH)/include
LOCAL_STATIC_ANDROID_LIBRARIES ： = \
LOCAL_STATIC_JAVA_LIBRARIES := \
LOCAL_JAVA_LIBRARIES := \
以上四个 LOCAL_变量分别是指明编译时添加到 include 搜索路径中的目录，Android 静态库，Java 静态库和 Java 库。

LOCAL_CFLAGS := \
-Wall \
-Werror
LOCAL_CPPFLAGS := -std=gnu++98
LOCAL_CFLAGS 设置在构建 C 和 C++ 源文件时构建系统要传递的编译器标记，使用 LOCAL_CPPFLAGS 仅为 C++ 指定标记。

LOCAL_PRODUCT_MODULE := true
LOCAL_VENDOR_MODULE := true
以上两个变量分别表示模块会安装在 product 分区和 vendor 分区。

LOCAL_SDK_VERSION := current
若是在 Android.mk 中添加该选项，则编译时会忽略源码隐藏的 API，即不能使用@hide API。

include $(BUILD_EXECUTABLE)
include $(BUILD_SHARED_LIBRARY)
include $(BUILD_STATIC_LIBRARY)
include $(BUILD_PACKAGE)
以上 include 是用来确认构建的内容和方式，如 BUILD_EXECUTABLE 对应可执行文件，BUILD_SHARED_LIBRARY 对应共享库，BUILD_STATIC_LIBRARY 对应静态库，BUILD_PACKAGE 对应 APK 应用。

3. 3 编译准备

3.3.1 复制文件

客户版本需要在编译前进行一下复制操作，在 vendor/sprd/release/IDH/{board_name}/下存在两个子目 录：bsp/和 out/，包括了一些闭源的 bin 和文件，在编译时需要使用到。请将这两个目录复制到源码树根 目录下：

$ cp –r bsp/ ~/root_dir
$ cp –r out/ ~/root_dir

3.3.2 初始化编译环境

如果没有初始化编译环境，需要在源码树的根目录执行一次 source build/envsetup.sh：

$ source build/envsetup.sh
including vendor/sprd/external/tools-build/vendorsetup.sh
including vendor/sprd/feature_configs/vendorsetup.sh

该脚本的作用是初始化编译环境，并引入一些辅助的 Shell 函数，这其中就包括第二步使用 lunch 函 数。下面介绍一些 build/envsetup.sh 中定义的常用函数：

名称	说明
croot	切换到源码树的根目录
m	在源码树的根目录执行 make
mm	编译当前目录下的模块，但不包括依赖
mma	编译当前目录下的模块，包括它们的依赖
mmm	编译指定目录下的模块，但不包括依赖
mmma	编译指定目录下的模块，包括它们的依赖
cgrep	在所有的 c/c++文件上执行 grep
jgrep	在所有的 Java 文件上执行 grep
resgrep	在所有的 上执行
godir	转到包含某个文件的目录路径
printconfig	显示当前编译的配置信息
lunch	指定编译的目标设备及编译类型

3.3.3 lunch 函数

如上小节所述，lunch 函数可以指定编译的目标设备及编译类型。 可以直接执行 lunch，不带参数，系统会

显示一个列表供选择：

……

62. sl8541e_1h10_32b_Natv-userdebug-gms

63. sl8541e_1h10_32b_Natv-userdebug-native

……

Which would you like? [aosp_arm-eng]

如要编译 sl8541e_1h10_32b_Natv-userdebug-gms，可以选择对应的序号 62 即可。

也可以直接 lunch 带参数来执行，效果与不带参数是一样的。

$ lunch sl8541e_1h10_32b_Natv-userdebug-gms

对于 user 版本的编译目标，是没有列举出来的，如果有需要编译 user 版本的，可以将对应的目标的“userdebug”改为“user”即可，如下

$ lunch sl8541e_1h10_32b_Natv-user-gms

3.3.4 kheader 函数

kheader 函数的作用是在用户空间编译中导入内核头文件，定义在 vendor/sprd/external/tools-build/vendorsetup.sh。

在进行工程编译之前，要经过以上三个指令的执行，做一些准备工作，再次整理如下：

$ source build/envsetup.sh //初始化编译环境

$ lunch //lunch 后会出现工程选择，选择需要编译的工程以完成工程相关的编译配置

$ kheader //完成安装 kernel 提供给用户态程序使用的头文件

3. 4 系统编译

通常在拿到代码，经过前面的初始化编译环境后，会完成一次完整的系统全编译，在源码树根目录下， 可以通过以下指令执行

make -jX

说明

支持多线程编译，X 是 cpu 核的数量

$ m //相当于 make

$ make //make 默认的目标是 droid

$ make droid //droid 是整个 Android 系统的编译目标

编译完成，需要备份 vmlinux 文件

8500CE 系列：bsp\out\sl8541e_1h10_32b\obj\kernel\vmlinux

8500E 系列：bsp\out\sl8541e_1h10wifi5g_32b\obj\kernel\vmlinux

3. 5 模块编译

在进行模块编译之前，建议先进行一次完整的编译，因为有的模块有依赖于其他模块。以 packages/apps/Email/为例，如果当前在根目录下，可以通过以下指令进行编译：

$ mmma packages/apps/Email/

如果当前在 packages/apps/Email/下，可以直接执行 mm 来编译。

3.6 单独镜像编译

在执行完编译准备的指令后，可以根据需要来进行单独编译某个镜像，具体如下：

■ 单独编译 u-boot

$ make bootloader

主要生成的目标文件：fdl2-sign.bin，u-boot_autopoweron-sign.bin，u-boot-sign.bin

■ 单独编译 chipram

$ make chipram

主要生成的目标文件：u-boot-spl-16k-sign.bin，fdl1-sign.bin，ddr_scan-sign.bin

■ 单独编译 kernel

$ make bootimage

主要生成的目标文件：boot.img，ramdisk.img，dtb.img

■ 单独编译 dtbo

$ make dtboimage

主要生成的目标文件：dtbo.img

■ 单独编译 system,img

$ make systemimage

主要生成的目标文件：system.img

■ 单独编译 userdata.img

$ make userdataimage

主要生成的目标文件：userdata.img

■ 单独编译 recovery.img

$ make recoveryimage

主要生成的目标文件：recovery.img，ramdisk-recovery.img

3.7 编译 pac 包

编译 pac 前要先 make 编译所有相关的镜像，以及添加相关的 modem bins 到对应目录。再执行以下两 条指令：

$ cp_sign

$ makepac

cp_sign 对应 build/make/envsetup.sh 中的 function cp_sign()，执行 vendor/sprd/proprietaries-source/packimage_scripts/sign_cp.sh

```
function cp_sign(){
echo enter cp_sign
secboot=$(get_build_var PRODUCT_SECURE_BOOT) .
$(gettop)/vendor/sprd/proprietories-source/packimage_scripts/sign_cp.sh $(gettop) $secboot
}
```

sign_cp.sh 中主要是执行两个脚本函数：

■ doPacpy()：解析 boad.ini，创建 pac.ini。

■ doImgCopyShark()：拷贝 modem bins 并签名。

makepac 对应 build/make/envsetup.sh 中的 function makepac()

```
function makepac(){
python vendor/sprd/release/pac_script/makepac.py
```

python vendor/sprd/release/pac_script/symbols.py
}
执行以上两个指令后，如果生成 pac 包成功，将有以下 log(以 8500CE 为例):
sl8541e_1h10_32b_Natv-userdebug-gms_SHARKLE_8541e_32b_halo.pac [PASS]

注意生成 pac 包需要两个配置文件：
一是.ini 文件，具体路径：vendor/sprd/release/pac_config/{product}.ini；
二是分区表，具体路径：device/sprd/{chip}/{product}/{product}.xml。
两个文件的路径如下：
===8500CE===
vendor\sprd\release\pac_config\sl8541e_1h10_32b.ini
device\sprd\sharkle\sl8541e_1h10_32b\sl8541e_1h10_32b.xml
===8500E===
vendor\sprd\release\pac_config\sl8541e_1h10wifi5g_32b.ini
device\sprd\sharkle\sl8541e_1h10wifi5g_32b\sl8541e_1h10wifi5g_32b.xml

3.8 其他编译指令

■ umake
umake 编译时会忽略 kati build.ninja regen 流程（重新生成 ninja），大大加快模块编译速度。
第一次编译使用 make module_name; 之后再次编译该模块，如果模块是在 Android.mk 中配置的，只是 修改源码、未修改.mk 或.bp 文件时，可以使用 umake 加速编译；如果模块是在 Android.bp 中配置的， 使用 umake 不受限制。
需要注意的是，umake 没有包含 image 签名流程，所以如果使用 umake 编译类似 bootloader 等 image 模块的话，编译结束后需要手动执行签名动作 build_tool_and_sign_images bootloader。
■ make clean
可以删除 out/target/product/{board_name}/下的大多数 img 文件。
■ 快速打包 image
编译过一次 image 后，可以直接使用以下目标打包：

Image	打包目标
system.img	• snod • systemimage-nodeps
userdata.img	userdataimage-nodeps
cache.img	cacheimage-nodeps
prodnv.img	prodnvimage-nodeps
vendor.img	• vnod • vendorimage-nodeps
product.img	• pnod • productimage-nodeps
ramdisk.img	ramdisk-nodeps
boot.img	bootimage-nodeps
recovery.img	recoveryimage-nodeps
vbmeta.img	vbmetaimage-nodeps

■ 快速定位变量

在 source build/envsetup.sh 和 lunch 后，可以使用 get_build_var 来定位变量，如下，定位变量 PRODUCT_USE_DYNAMIC_PARTITIONS

$ get_build_var PRODUCT_USE_DYNAMIC_PARTITIONS

表示在 build/make/core/product_config.mk 、 device/sprd/sharkle/common/DeviceCommon.mk 有关于 PRODUCT_USE_DYNAMIC_PARTITIONS 的定义。

■ 保存所有编译 log

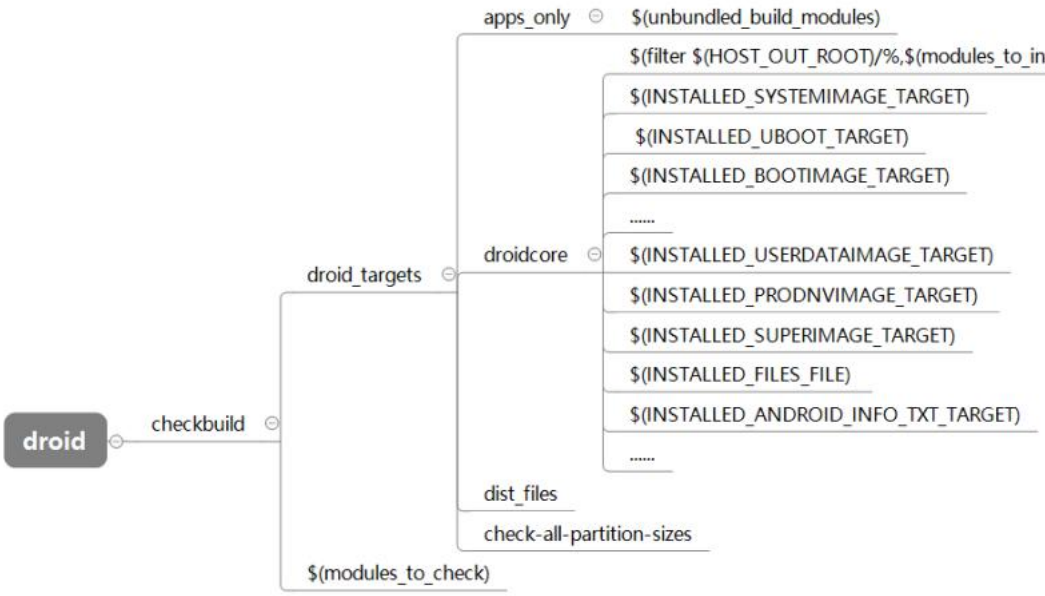
make -k | tee fulllog.txt 2>&1 完整的 log 存放在 fulllog.txt，其中-k 可以在遇到错误时继续编译，一次性把所有错误暴露出来。

4 Makefile 文件说明

4.1 main.mk

在源码树的根目录下，一个 Makefile 文件中 include build/make/core/main.mk，而 main.mk 定义了整个 Android 的编译关系。

下面，再分析一下编译整个 Android 工程的目标依赖关系：



droid 是整个编译的默认目标，当在根目录执行 make 时，其实是相当于 make droid 的，可以编译出整个 Android 工程的完整编译。根据编译目标关系图，可以看到主要依赖于 droid_targets 和 $(modules_to_check)。

$(modules_to_check)：该目标用来确保我们定义的构建模块是没有冗余的。

droid_targets 又进一步依赖其他的编译目标：

- apps_only：该目标将编译出当前配置下不包含 user，userdebug，eng 标签的应用程序。
- droidcore：该目标仅仅是所依赖的几个目标的组合，其本身不做更多的处理。下面是几个主要的目 标介绍：
  - $(filter $(HOST_OUT_ROOT)/%,$(modules_to_install))：安装除了 out/root/ 下的模块 υ $(INSTALLED_XXXIMAGE_TARGET)：编译 XXX.img 的内容；
  - $(INSTALLED_FILES_FILE)：该目标会生成 out/target/product/< product_name >/ installedfiles.txt 文件，该文件中内容是当前系统镜像中已经安装的 文件列表。
  - $(INSTALLED_ANDROID_INFO_TXT_TARGET)：该目标会生成一个关于当前 Build 配置的 设备

信息的文件，该文件的生成路径是：out/target/product/< product_name >/android-info.txt

- dist_files：该目标用来拷贝文件到 /out/dist 目录，目前展锐没有编译出这个目录。
- check-all-partition-sizes：列出 super 分区各镜像的编译目标。

4.2 AndroidProducts.mk

AndroidProducts.mk 是产品版本定义文件，包含两个变量的定义，PRODUCT_MAKEFILES 是该产品下 所有 board 的.mk 文件，COMMON_LUNCH_CHOICES 是添加到菜单的选项，一般一个产品会有多个选项添加。文件路径：device/sprd/{chip_name}/AndroidProducts.mk。

8500CE/8500E 对应的路径是 device/sprd/sharkle/AndroidProducts.mk，内容如下：

```
PRODUCT_MAKEFILES += \
……（省略部分内容）
        sl8541e_1h10_32b_Natv:$(LOCAL_DIR)/sl8541e_1h10_32b/sl8541e_1h10_32b_Natv.mk\
        sl8541e_1h10wifi5g_32b_Natv:$(LOCAL_DIR)/sl8541e_1h10wifi5g_32b/sl8541e_1h10wifi5g_32b_Natv.mk\
……（省略部分内容）

COMMON_LUNCH_CHOICES := \
……（省略部分内容）
        sl8541e_1h10_32b_Natv-userdebug-gms \
        sl8541e_1h10wifi5g_32b_Natv-userdebug-gms \
……（省略部分内容）
```

4.3 AndroidBoard.mk

AndroidBoard.mk 主要定义了 board 的编译 uboot，chipram，sml，trusty 的伪目标。如 device\sprd\sharkle\sl8541e_1h10_32b\AndroidBoard.mk

```
……（省略部分内容）
 .PHONY: bootloader bootloader:
$(INSTALLED_UBOOT_TARGET)
$(INSTALLED_UBOOT_TARGET):
@cp $(TARGET_BSP_OUT)/u-boot15/u-boot*.bin $(PRODUCT_OUT)
@cp $(TARGET_BSP_OUT)/u-boot15/fdl2*.bin $(PRODUCT_OUT)

.PHONY: chipram chipram:
$(INSTALLED_CHIPRAM_TARGET)
$(INSTALLED_CHIPRAM_TARGET):
@cp $(TARGET_BSP_OUT)/chipram/u-boot*.bin $(PRODUCT_OUT)
@cp $(TARGET_BSP_OUT)/chipram/fdl1*.bin $(PRODUCT_OUT)
@cp $(TARGET_BSP_OUT)/chipram/ddr_scan*.bin $(PRODUCT_OUT)
……（省略部分内容）
```

4.4 BoardConfig.mk

BoardConfig.mk 是每个 board 的配置文件，包括具体 board 的镜像属性，功能属性的配置。

===8500CE===
文件位于 device\sprd\sharkle\sl8541e_1h10_32b\BoardConfig.mk
===8500E===
device\sprd\sharkle\sl8541e_1h10wifi5g_32b\ BoardConfig.mk
如下列出了关于分区的 配置：
……（省略部分内容）
CHIP_NAME := sharkle // 芯片名称
include device/sprd/sharkle/common/BoardCommon.mk // include 其他的.mk
……（省略部分内容）

ext4 partition layout
#BOARD_VENDORIMAGE_PARTITION_SIZE := 419430400 // vendor 分区的大小
BOARD_VENDORIMAGE_FILE_SYSTEM_TYPE := ext4 // vendor.img 的文件系统类型
TARGET_COPY_OUT_VENDOR=vendor // 在 out 目录下对应的目录为 vendor/
#creates the metadata directory
BOARD_USES_METADATA_PARTITION := true // 配置 META 分区
除此之外，该文件还有关于 camera，WCN 等的配置。