Linux 开发基础

Develop in Linux

—《机器视觉技术实训》课程笔记

杨青青

2020 年 2 月 25 日

Email：qqyang@nit.zju.edu.cn

本次课程介绍 Linux 中进行开发所需要的常用工具，包括 GCC, CMAKE, GIT 等。了解这些工具并能够熟练运用日常开发中所需的常用命令，是 Linux 环境下进行机器视觉应用开发的基本技能。本次课程笔记首先介绍了 Ubuntu Linux 下的软件包管理工具，然后分别介绍 GCC、CMAKE 和 GIT 这三个工具的常用功能。

APT

Ubuntu 是一个基于 Debian 的 Linux 发行版，所以软件包管理使用了基于 Debian 的包管理工具。Debian 的软件都会打包成“.deb”后缀的软件包进行分发，`dpkg`是主要的包管理程序。

APT(Advanced Package Tool) 是 Debian 包管理系统的一个高级接口，底层使用了`dpkg`程序。因为其强大的软件包管理能力和简单方便的使用特性而被更多的用户使用，也成为了目前 Debian Linux 系统家族中最为广泛使用的接口。APT 中最常用的命令是`apt-get`、`apt-cache`和`apt-config`。由于`apt-*`系列命令比较容易混淆，从 Debian Jessie 版本开始，提供了一个新的程序：`apt`。`apt`命令结合了`dpkg`和`apt-*`系列命令最常用的功能，提供了一个对用户更友好的包管理命令子集。因此，我们以后都使用`apt`命令来进行软件包安装和更新。仅仅在`apt`命令不支持的情况下，根据需要使用`dpkg`或`apt-*`系列命令[1]

值得一提的是，`apt`并不能直接安装“`.deb`”软件包。它的工作方式是，指定软件包源文件（主文件存放在`/etc/apt/sources.list`中），并通过软件包的名字从包服务器中获取“`.deb`”包。在下载后，`apt`会调用`dpkg`来进行安装。

更多关于 Debian 包管理工具的内容请参考。`https://www.debian.org/doc/manuals/debian-faq/pkgtools.en.html`

常用的`apt`命令包括：

- `apt update`：根据软件包源更新系统中的包信息；
- `apt install foo`：安装名为`foo`的软件包[2]和它所需要的所有依赖包[3]；
- `apt remove foo`：从系统中卸载名为`foo`的软件包；

[1] 对于在安装脚本或者一些高级使用中，apt-get 仍然是必需的而且更推荐使用。

[2] 实际应用中根据需要将 foo 换成所需要安装的软件包名。

[3] 一个软件的运行可能需要调用其它的软件，我们将这些需要被调用的软件称为依赖项。依赖包可能会有其它的依赖包，有时候安装一个软件包，可能会同时安装很多依赖包。

- `apt purge foo`：从系统中卸载名为`foo`的软件包和所有配置文件；
- `apt list --upgradable`：列出有新版本的已安装软件包；
- `apt upgrade`：升级系统中所有的软件包（不会安装额外的包或者卸载包）；
- `apt search word`：在软件包库中查找并列出带有`word`描述的所有软件包；
- `apt show foo`：显示`foo`软件包的详细信息；

需要注意的是，因为软件包安装会修改系统目录，在 Ubuntu 中一般是`/usr/bin`、`/usr/lib`等目录。所以，我们需要管理员权限来运行`apt`命令。可以在命令前加`sudo`来提升该命令的执行权限。具体的写法参见后面小节中软件包的安装命令。

修改 APT 源

由于 Ubuntu APT 源的主服务器在国外，因此访问对网络要求比较高。为了世界各地的用户快速获得软件包，除了主服务器外，在世界各地分布了多个镜像服务器[4]。在国内，比较推荐的一个 Ubuntu APT 镜像服务器是阿里云镜像。下面介绍如何进行修改 APT 源文件来使用阿里云镜像。

[4] 所谓镜像服务器就是指，服务器上的内容和主服务器内容是一致的；一般镜像服务器会定时和主服务器内容进行同步。

Ubuntu 软件包源文件位于 `/etc/apt/sources.list`，因此，需要修改该文件。在修改之前，一个好的习惯是进行备份。

```
$ cd /etc/apt
$ sudo cp sources.list sources.list.back
```

由于该源文件是一个系统配置文件，因此需要使用 `sudo` 提升权限。使用 `gedit` 打开修改源文件：

```
$ sudo gedit sources.list
```

一个最简单的办法是，把所有的文本内容删除，然后加入以下内容：

```
deb http://mirrors.aliyun.com/ubuntu/ xenial main
deb-src http://mirrors.aliyun.com/ubuntu/ xenial main

deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-updates main

deb http://mirrors.aliyun.com/ubuntu/ xenial universe
deb-src http://mirrors.aliyun.com/ubuntu/ xenial universe
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates universe
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-updates universe
```

```
11
12 deb http://mirrors.aliyun.com/ubuntu/ xenial-security main
13 deb-src http://mirrors.aliyun.com/ubuntu/ xenial-security main
14 deb http://mirrors.aliyun.com/ubuntu/ xenial-security universe
15 deb-src http://mirrors.aliyun.com/ubuntu/ xenial-security universe
```

另一种方法是进行文本替换,将所有的 http://archive.ubuntu.com/替换成 mirrors.aliyun.com[5]。

替换以后，进行本地软件包目录更新：

```
$ sudo apt update
```

至此，软件包源修改完成。后续的软件安装就会使用阿里云的源了。

[5] 阿里云开发者社区教程：https://developer.aliyun.com/mirror/ubuntu?spm=a2c6h.13651102.0.0.3e221b11jlKqI2，注意找到对应的版本进行修改。

GCC

GCC(GNU Compiler Collection) 是 GNU 推出的一套编译器和标准库的集合，支持 C、C++、Objective-C、Fortran、Ada、Go 和 D 语言。我们使用 C 语言和 C++ 进行开发，对应的编译器是 gcc 和 g++。这两个编译器程序和标准程序库都包含在 GCC 套件里。在 Ubuntu 中，可以通过 build-essential 包来进行安装：

```
$ sudo apt install build-essential
```

这里，使用了 sudo 来提升权限，如果没有加 sudo ，系统会提示权限不足。

事实上，在开发软件时，很少直接使用 gcc 或 g++ 命令。一个程序往往由很多个源文件组成，Linux 提供了 make 程序来构建程序，这也是 GNU 程序的默认构建方式。在构建程序时， make 需要了解程序源文件之间的依赖关系，调用哪个编译器，需要链接哪些库文件，需要执行哪个程序来删除已经编译的二进制文件等规则。通过这些规则， make 程序可以进行程序构建、安装和删除等操作。所有这些规则包含在一个名为 Makefile 的文件中。在 Makefile 文件中，程序员需要按照 Makefile 的语法规则指定命令执行的顺序和一系列的文件依赖关系[6]。

这里，我们用一个非常简单的程序 sl 来介绍以下 Makefile 的基本用法[7]。大家可以在 GitHub 上下载该程序的源代码[8]，然后使用 make 来进行编译。出去代码仓库中的一些 README 之类的文件，编译该程序只需要保留 2 个源文件（sl.c和 sl.h）和一个 Makefile 文件。我们来看一下它的 Makefile文件（Listing 1）。

第 1-7 行都是注释，使用 # 开头。这是一个 C 语言写的程序，所以在第 9 行指定了需要使用的编译器是 gcc ，第 10 行设置了编译器选项。这两个都是 Makefile 中的变量。后面的内容都指定了一条规则。例如，第 12 行 all 规则，表示 make 程序需要编

[6] 对于多文件编程不熟悉的同学，建议阅读《C 语言程序设计：现代方法》的第 15 章的内容。

[7] 这里我确实没有写错，是 sl 。

[8] https://github.com/mtoyoda/sl

Code Block 1: sl 的 Makefile 文件

```
1  #=============================================
2  #    Makefile: makefile for sl 5.1
3  #       Copyright 1993, 1998, 2014
4  #                 Toyoda Masashi
5  #                 (mtoyoda@acm.org)
6  #       Last Modified: 2014/03/31
7  #=============================================
8
9  CC=gcc
10 CFLAGS=-O -Wall
11
12 all: sl
13
14 sl: sl.c sl.h
15         $(CC) $(CFLAGS) -o sl sl.c -lncurses
16
17 clean:
18         rm -f sl
19
20 distclean: clean
```

译的所有目标程序。当执行 make 或者 make all 命令时，就会编译 sl 这个目标程序。为了编译 sl， make 需要找到编译 sl 的规则，就在第 14-15 行。第 14 行说明 sl 编译程序需要两个源文件，在下一行规则说明了如何编译 sl 。这里， $(CC)和 $(CFLAGS)是在第 9-10 行定义的变量，我们分别用 gcc 和 -O -Wall 替换，于是就得到了完整的编译命令：

```
gcc -O -Wall -o sl sl.c -lncurses
```

该命令的最后一个参数 -lncurses指定了需要链接的库文件，名为 ncurses。所以,为了编译这个源代码,我们需要安装 ncurses 开发库[9]：

```
$ sudo apt install libncurses5-dev
```

注意，在安装开发库时，一定要安装 -dev结尾的包，该包是用于程序开发的包，包含了所需要的头文件、库文件和运行时文件等。在不确定一个包的名字时，可以使用包搜索的命令来搜索以下有哪些包：

```
$ apt search ncurses
```

使用关键字 ncurses 来搜索，会列出所有包含 ncurses 的包。注意，这里并没有使用 sudo ，因为我们不需要获取系统文件的写入权限。

安装完 ncurses ，我们就可以在 Shell 中输入 make 命令，就会编译 sl 这个程序了。第 17 行的 clean 规则指定了如何删除编译好的二进制文件，因为没有依赖项，所以冒号后面是空的，具体的命令在下一行，即第 18 行，用到了我们熟悉的 rm 命令。

[9] ncurses 是 Linux/Unix 系统下基于文本的用户界面库。我们等一下在安装 CMake 时又会遇到它。

`Makefile` 功能非常强大，以上的例子仅仅是 `Makefile` 的最基本应用。由于 `Makefile` 的强大功能是建立在复杂规则的编写上的，人工写起来比较麻烦，因此，软件工具开发者又提出了很多的对开发者更友好工具，比如 CMake，Basel 等。比较尽管我们在实际的开发中很少需要自己写 `Makefile` 文件，但是，了解 `Makefile` 的简单对后面的程序开发有帮助的。

下面介绍目前开源软件最常使用的一种程序构建工具：CMake。

CMake

CMake[10]是一个开源的、跨平台的工具集，用于软件构建、测试和软件包打包。CMake 的跨平台特性可以让软件开发者仅需要维护一份 CMake 的代码组织文件，就可以在不同平台根据编译器生成不同的平台相关的工程文件，例如在 Linux/Unix 下，可以生成 `Makefile`；在 Windows 下就可以生成 VisualStudio 的 `solution`文件，在 macOS 下可以生成 Xcode 工程文件。对于 C/C++ 开发者来说，CMake 是目前最合适的代码构建工具。

[10] `https://cmake.org/`

可以通过 `apt` 快速安装 CMake 工具：

```
$ sudo apt install cmake
```

安装完成后，就可以使用 `cmake` 命令来使用 CMake 工具了。

此外，在 Shell 中除了使用命令方式外，还可以安装一个 Curses GUI：

```
$ sudo apt install cmake-curses-gui
```

Curses GUI 在 shell 环境下提供了一个更为友好直观的用户界面，控制各个编译选项更加方便。在安装后，使用 `ccmake` 命令可以打开 Curses GUI。对于喜欢图形界面的开发者，也可以安装 `cmake-gui` 来使用窗口 GUI 工具来使用 CMake。

在 Linux 下，CMake 的一般用法是：

```
$ cmake <CMakeLists.txt所在目录>
```

`CMakeLists.txt`是 CMake 的主配置文件，每一个使用 CMake 来构建的工程都需要一个 `CMakeLists.txt` 文件。在 CMake 根据 `CMakeLists.txt` 文件中的配置处理完成后，会生成一个 Makefile 文件，可以使用 `make` 来完成最终的程序构建[11]。

[11] 也可以使用 cmake –build 来进行构建。在 Linux 下，一般使用 make 会更加方便一些。

下面通过一个简单的示例来讲解 CMake 的用法，我们通过自己写一个 `CMakeLists.txt` 来构建前一节的 `sl` 程序。Ubuntu Linux 内置了一个图形界面下的编辑器：gedit。首先导航到 y `sl` 源代码所在的目录，然后可以在 shell 中输入 `gedit CMakeLists.txt` 来打开一个 gedit 窗口，并创建一个 CMakeLists.txt 的文件。

Code Block 2: 一个简单的 CMakeLists.txt

```
# cmake 所需的最低版本要求
cmake_minimum_required(VERSION 2.8)

# 设置项目名称
project(cmakesl)

# 寻找 curses 库
find_package(CURSES REQUIRED)

# 加入头文件目录
include_directories(${CURSES_INCLUDE_DIRS})

# 添加项目的输出程序和所需源代码
add_executable(cmakesl sl.c)

# 将 curses 库链接到程序中
target_link_libraries(cmakesl ${CURSES_LIBRARIES})
```

其中，每一条语句都进行了标注 [12]。在 CMakeLists.txt 中是不区分大小写的，但是为了方便方便阅读，我的习惯是将 CMake 命令写成小写的形式，参数、变量等写成大写的形式。和 Makefile 文件相比，CMake 的命令更“高级”，更符合人类的阅读方式，在逻辑上也更容易组织。使用 CMake 的一个难点在于找库命令，即 find_package。需要使用正确的库名才可以找到库文件所在位置。例如这里，寻找的库名不是 ncurses，而是 curses。这是因为，在早期的 Unix 系统上就是curses库，ncurses[13]是原来 curses 的改进版。所以，为了兼容多操作系统，使用 find_package(CURSES, REQUIRED) 会找到 curses 或者 ncurses，找到其中的一个库就会正确返回。其中 REQUIRED 参数表示这个库是必需的，如果找不到 cmake 就会失败，也就无法继续进行构建。

[12] 与很多脚本语言类似，CMake 使用“#”表示一行注释的开始。

[13] new curses

Git

介绍完代码构建工具后，我们在本课程笔记的最后介绍一个代码版本控制工具：Git。所谓的版本控制，就是在开发过程中，对每一次的源代码改动进行记录，在需要的时候可以回滚。Git 还拥有非常灵活的代码分支功能，非常适合开发团队的共同代码维护。目前，Git 的代码托管网站主要有：GitHub、BitBucket 和 GitLab。目前，这些托管网站都支持免费的公开或者私人代码仓库。其中 GitHub 是最活跃的开源代码社区。

在 Ubuntu 中，Git 可以通过 APT 快速安装：

```
$ sudo apt install git
```

一些历史：早期的版本控制工具都是集中式代码管理，例如 CVS, SVN 等，这些版本控制工具把代码库放在服务器上，在客户端仅保存当前修改的一个版本，如果需要操作早期版本，需要联网操作。

Git 是一种分布式版本管理工具，即每个客户端都具有一份完整的仓库备份，更加适合开源软件的开发。值得一提的是，Git 的创造者就是 Linux 之父 Linus Torvalds。

Git 的学习资料可以在官方网站[14]获得。Pro Git 是一份在线的电子书[15]，详尽地介绍了 Git 各种功能的使用方法。

本节，我们介绍如何进行 git 安装后的首次使用配置方法，然后在本地建立一个 Git 仓库。

获取帮助信息

在开始之前，首先找到一份救援手册是一个明智的选择。可以通过下面的几个命令中的任意一个来获得 Git 命令的帮助：

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

这三个命令是等价的，都会调出 <verb>命令的详尽手册页。例如，我们在使用前需要配置 Git 环境，需要用到 git config命令，这时，可以通过下面的命令获取帮助：

```
$ git config --help
```

如果仅仅想要获得一份简要的命令选项帮助，可以使用 -h 选项：

```
$ git config -h
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    --worktree            use per-worktree config file
    -f, --file <file>     use given config file
    --blob <blob-id>      read config from given blob object
......
```

配置 Git 环境

在首次使用 Git 之前，首先需要设置用户和邮件。这两个信息非常重要，因为 Git 在 commit 操作时需要用到。设置方法如下：

```
$ git config --global user.name "Qingqing Yang"
$ git config --global user.email qqyang@nit.zju.edu.cn
```

参照这个命令，把姓名和邮箱地址改成自己的就可以了。还可以设置默认的编辑器，这里我们就先不设置了，Git 会使用系统默认的编辑器。可以通过下面的命令查看 Git 的设置是否正确：

```
$ git config --list
```

[14] https://git-scm.com/

[15] 该书的中文版本可以通过这个链接浏览：https://git-scm.com/book/zh/v2

初始化 Git 仓库

设置完成后, 就可以在本地创建 Git 代码仓库了。选择一个目录, 导航到该目录下，运行 `git init` 命令。例如，在 `~Workspace` 下创建一个 `machine-vision` 的目录，然后在该目录下初始化一个 Git 仓库：

```
$ cd ~/Workspace
$ mkdir machine-vision && cd machine-vision
$ git init
```

该命令会返回一条信息，显示初始化了一个 empty git repository。但是当使用 `ls` 命令查看时，发现目录下依然空空如也。实际上，`git init`命令会在当前目录下创建一个 `.git`的目录，该目录内放置了本地仓库的配置信息。

思考：该怎么显示这个目录?

在进行代码作业之前，先看一下仓库的状态，这是一个好的习惯。可以使用 `git status` 查看当前仓库的状态：

```
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

该状态显示目前在 `master` 分支上，即主分支，还没有任何 commits，也没有什么需要 commit 的。在最后一行的括号内还给了提示，怎么进行创建并加入代码。十分人性化。

OK，现在我们使用文本编辑器创建一个文件，命名为 `README`。这里，我们依然使用 `gedit`：

```
$ gedit README
```

在 Ubuntu 下，会打开一个文本编辑器界面，输入任意的内容，然后保存关闭。再使用 `git status` 看以下当前的状态：

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README

nothing added to commit but untracked files present (use "git add" to track)
```

这个时候的提示已经不一样了,Git 已经检测到了 untraced files，表示这个文件已经在目录下，但是还没有进行“追踪”，即进行版本控制。现在进行最重要的一步操作，把这个修内容 commit 到 Git 仓库中，让 Git 可以对其进行版本控制：

```
$ git add README
```

然后再次查看仓库状态：

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README
```

这时候，Git 已经显示有 Changes to be committed:了，而且也提示了怎么样把刚才的操作撤销。现在，我们进行 commit 操作：

```
$ git commit -m "Add README"
master (root-commit) 5a74f0b] Add README
 1 file changed, 3 insertions(+)
 create mode 100644 README
```

这里，使用了直接添加 commit 说明的方式，用 -m 选项来加入本次 commit 的说明，在这里是引号内的内容。注意，加入 commit 说明是必需的，如果简单运行 git commit 命令，Git 会打开默认的文本编辑器来输入 commit 信息。因为我们还没有对文本编辑器进行特别的介绍，因此，这里采用命令的方式加入 commit 说明。Git 的提示信息会说明本次 commit 的总结信息，并给这个 commit 一个 SHA-1 校验码。

这时，我们可以再打开 README 文件进行一些编辑，再查看仓库状态时，可以看到如下信息：

```
$ git status
n branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README

no changes added to commit (use "git add" and/or "git commit -a")
```

Git 提示可以使用 git add 来添加刚才的修改,也可以 git commit -a命令来添加。这里，尝试一下 Git 提供的新命令：

```
$ git commit -a -m "Add more"
```

这样，就可以把对 README的新的修改 commit 到仓库中。

关于 Git 的使用暂时介绍到这里，但是 Git 的内容远远不止这些。要使用好 Git 需要多看资料，多实践，通过地实践训练才能熟悉日常的使用。

小结

阅读和参考资料

[1] “The Debian package management tools.”[Online]. Available: https://www.debian.org/doc/manuals/debian-faq/pkgtools.en.html.

[2] “CMake Cookbook（中文翻译）.”[16] [Online]. https://chenxiaowei.gitbook.io/cmake-cookbook/.

[3] “Pro Git.”[Online]. 中文版网址：https://git-scm.com/book/zh/v2. 该书详尽地介绍了 Git 各种功能的使用方法。

[16] 网上资料，没有确认过翻译的版权问题。如有该资料存在版权问题，请购买原版。

实践

- 修改 APT 源，并完成软件更新
- 安装 GCC，能够运行 gcc、 g++ 和 make。
- 安装 CMake 和 CMake Curses GUI,能够运行 cmake 和 ccmake。
- 安装并配置 Git。并在本地创建一个仓库。
- 探索：在 GitHub[17]上创建一个账号（如果已有账号则使用原账号），创建一个名为 mvia-course-project的仓库[18]，并克隆到本地。

[17] https://github.com/

[18] mvia 表示 Machine Vision in Action