WEEK1 NOTE

你应该已经在 Week0 中完成了关于 Linux 环境的安装，那么现在来尝试下在 Linux下进行开发吧😳。

你将会学到：

工具的选择

这里使用 vscode 作为编辑器，gcc 作为编译器。

> ⚠ **Warning**
>
> 你可能会想，为什么不使用“现代化”的IDE，而是要麻烦的自己操作。作为学习 c 的程序员，搞明白 c 语言底层逻辑、与系统的交互等等是很重要的，作为一个系统级语言，光跑起来是不够的，要明白是如何跑的。你以后可能在写代码的时候发生许多“玄学问题”，如果没有底层能力将会十分煎熬。
>
> 下面这段出自 learn c the hard way
>
> IDE，或者“集成开发工具”，会使你变笨。如果你想要成为一个好的程序员，它会是最糟糕的工具，因为它隐藏了背后的细节，你的工作是弄清楚背后发生了什么。如果你试着完成一些事情，并且所在平台根据特定的IDE而设计，它们非常有用，但是对于学习C编程（以及许多其它语言），它们没有意义。
>
> > 注
> >
> > 如果你玩过吉他，你应该知道TAB是什么。但是对于其它人，让我对其做个解释。在音乐中有一种乐谱叫做“五线谱”。它是通用、非常古老的乐谱，以一种通用的方法来记下其它人应该在乐器上弹奏的音符。如果你弹过钢琴，这种乐谱非常易于使用，因为它几乎就是为钢琴和交响乐发明的。
> >
> > 然而吉他是一种奇怪的乐器，它并不能很好地适用这种乐谱。所以吉他手通常使用一种叫做TAB（tablature）的乐谱。它所做的不是告诉你该弹奏哪个音符，而是在当时应该拨哪根弦。你完全可以在不知道所弹奏的单个音符的情况下学习整首乐曲，许多人也都是这么做的，但是如果你想知道你弹的是什么，TAB是毫无意义的。
> >
> > 传统的乐谱可能比TAB更难一些，但是会告诉你如何演奏音乐，而不是如果玩吉他。通过传统的乐谱我可以在钢琴上，或者在贝斯上弹奏相同的曲子。我也可以将它放到电脑中，为它设计全部的曲谱。但是通过TAB我只能在吉他上弹奏。

> IDE就像是TAB，你可以用它非常快速地编程，但是你只能够用一种语言在一个平台上编程。这就是公司喜欢将它卖给你的原因。它们知道你比较懒，并且由于它只适用于它们自己的平台，他们就将你锁定在了那个平台上。
>
> 打破这一循环的办法就是不用IDE学习编程。一个普通的文本编辑器，或者一个程序员使用的文本编辑器，例如Vim或者Emacs，能让你更熟悉代码。这有一点点困难，但是终结果是你将会熟悉任何代码，在任何计算机上，以任何语言，并且懂得背后的原理。

vscode 的配置

我们为你导出了一个通用的基础配置，当然，我们同时也欢迎你自己去配置你喜欢的、你需求的东西。vscode 作为一个拥有相当丰富的插件库的应用，大部分的配置将会很愉悦（如果你想使用某插件有疑问， RTFM）。但记住，vscode 本质上还是一个文本编辑器，请不要试图当作 IDE 使用（如配置大量的“自动编译、执行”插件等等），你要做的是改善自己的打字体验。至于编译，交给 gcc 吧。

Vim

程序员们对自己正在使用的文本编辑器通常有着 非常强的执念。

现在最流行的编辑器是什么？Stack Overflow 的调查（这个调查可能并不如我们想象的那样客观，因为 Stack Overflow 的用户并不能代表所有程序员）显示，Visual Studio Code 是目前最流行的代码编辑器。而 Vim 则是最流行的基于命令行的编辑器。

Vim 有着悠久历史；它始于 1976 年的 Vi 编辑器，到现在还在 不断开发中。Vim 有很多聪明的设计思想，所以很多其他工具也支持 Vim 模式（比如，140 万人安装了 Vim emulation for VS code）。即使你最后使用 其他编辑器，Vim 也值得学习。

> **Tip**
>
> **Vim的哲学**
>
> 在编程的时候，你会把大量时间花在阅读/编辑而不是在写代码上。所以，Vim 是一个*多模态*编辑 器：它对于插入文字和操纵文字有不同的模式。Vim 是可编程的（可以使用 Vimscript 或者像 Python 一样的其他程序语言），Vim 的接口本身也是一个程序语言：键入操作（以及其助记名） 是命令，这些命令也是可组合的。Vim 避免了使用鼠标，因为那样太慢了；Vim 甚至避免用 上下左右键因为那样需要太多的手指移动。
>
> 这样的设计哲学使得 Vim 成为了一个能跟上你思维速度的编辑器。

对于Vim我们这里做基础要求：能够最低限度使用 vim 进行编辑。也就是当你 ssh 没有图形界面的时候，你可以使用 vim 进行修改。

对于vim的安装我们建议STFW，但是这里会留下关于vim的拓展资料
拓展资料：

- `vimtutor` 是一个 Vim 安装时自带的教程
- Vim Adventures 是一个学习使用 Vim 的游戏
- Vim Tips Wiki
- Vim Advent Calendar 有很多 Vim 小技巧
- Vim Golf 是用 Vim 的用户界面作为程序语言的 code golf
- Vi/Vim Stack Exchange
- Vim Screencasts
- Practical Vim（书籍）

Git

你是否有这种情况：写代码的时候发现自己很大一段都写错了，想回到曾经的版本发现单纯的撤回完全做不到。或者团队开发项目的时候不知道怎么同步每个人的代码。那么这个时候，你就会想到 Git 的好处了。

什么是 Git

版本控制系统 (VCSs) 是一类用于追踪源代码（或其他文件、文件夹）改动的工具。顾名思义，这些工具可以帮助我们管理代码的修改历史；不仅如此，它还可以让协作编码变得更方便。VCS 通过一系列的快照将某个文件夹及其内容保存了起来，每个快照都包含了文件或文件夹的完整状态。同时它还维护了快照创建者的信息以及每个快照的相关信息等等。

为什么说版本控制系统非常有用？即使您只是一个人进行编程工作，它也可以帮您创建项目的快照，记录每个改动的目的、基于多分支并行开发等等。和别人协作开发时，它更是一个无价之宝，您可以看到别人对代码进行的修改，同时解决由于并行开发引起的冲突。

现代的版本控制系统可以帮助您轻松地（甚至自动地）回答以下问题：

- 当前模块是谁编写的?
- 这个文件的这一行是什么时候被编辑的？是谁作出的修改？修改原因是什么呢?
- 最近的1000个版本中，何时/为什么导致了单元测试失败?

尽管版本控制系统有很多， 其事实上的标准则是 **Git** 。而这篇 XKCD 漫画 则反映出了人们对 Git 的评价：

找不到“./material/git.png”。

我该如何使用 Git

Git的命令行接口(基础部分)

- `git help <command>`: 获取 git 命令的帮助信息
- `git init`: 创建一个新的 git 仓库，其数据会存放在一个名为 `.git` 的目录下
- `git status`: 显示当前的仓库状态
- `git add <filename>`: 添加文件到暂存区
- `git commit`: 创建一个新的提交
  - 如何编写 良好的提交信息!
  - 为何要 编写良好的提交信息
- `git log`: 显示历史日志
- `git log --all --graph --decorate`: 可视化历史记录（有向无环图）
- `git diff <filename>`: 显示与暂存区文件的差异
- `git diff <revision> <filename>`: 显示某个文件两个版本之间的差异
- `git checkout <revision>`: 更新 HEAD 和目前的分支

> **下面举一个clone仓库的例子**
>
> - **安装**
>
> ```
> # Debian , Ubuntu
> sudo apt install git
> ```
>
> - **配置**
> 前面提到 `git` 可以回答**当前模块是谁编写的，是谁作出的修改**等问题。这就需要提前配置好信息——告诉 `git` 你是谁
>
> ```
> git config --global user.name "Zhang San"             # your name
> git config --global user.email "zhangsan@foo.com"     # your email
> ```
>
> 完成以上配置后你的每一次 `commit` 都会以名称为 `Zhang San` 邮箱为 `zhangsan@foo.com` 的信息提交
>
> `git config` 可以配置的内容众多，可以自行上网搜索或者使用以下方法中的任意一种获得任何git命令的手册页(manpage)（将verb替换为config即为git config的手册）

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

- **克隆仓库**
  克隆仓库即下载对应仓库内容到当前目录，下面介绍使用方法

  第一步：
  使用 `cd` 命令移动到你想要存储仓库的位置

  ```
  cd 路径
  ```

  第二步：
  获取仓库url
  仓库绿色Code按钮展开后就会给出，复制选中https后给出的url
  （如图中所示，不同仓库的url内容不同，改图仅作参考）

  找不到“./material/git clone url.png”。

  第三步：
  使用 `git clone` 命令进行复制

  ```
  git clone <url>
  ```

  若是继续以第二步中图为例即为

  ```
  git clone https://github.com/camera-2018/git-example.git
  ```

大功告成！

显然，上面列举的只是最基本最基本的内容，对于真正想使用 Git 的人，其实最推荐的是先学会再使用。但显然我们并没有那么多耐心。所以，当你感觉缺某部分的时候，STFW吧。

当然，从零开始摸索是十分痛苦且迷茫的，所以下面会给出一本书和一个游戏教程

> **Note**
>
> 推荐学习：

Book: Pro Git
Game: Learn Git Branching

Important

小作业，学习如何回退到自己的某一个提交，看到这里就开一个仓库练手吧

Caution

为什么这里没有如何创建一个仓库的内容？因为 Github 和 Git 创建的仓库远程默认名称不同，第一次用你可能会晕（当然，最好的方法当然是尝试下），所以这次课程先会建议你先在 Github 上面创建仓库，然后 clone 下来（但我们仍推荐做全部的尝试，因为提前的尝试可以减少未来遇见的恐慌）。

Warning

记住，在项目中每做一部分就要提交，不然你一定会后悔的（等项目炸了就老实了）

Tip

Github:

注意区分 Git 和 Github: Git 是一个版本管理系统，而 Github 是一个在线的，基于 Git 的代码托管平台。最基础的，你可以认为 Github 远程存储你的 Git 仓库，并且可以与他人共享。当然，Github 现在是一个庞大的开源平台，你可以在上面发掘很多有用的东西。

我们的任务更新后续都会发表在 Github 上面，需要你自己 clone 仓库（你将会在 **lab** 中看见），这个操作并不需要账号。但我们仍然希望你能接触这样一个开源社区，尝试注册 账号并且使用它。

对英文怯魅!

▶ 拓展

TAR

tar 是一个常用的命令行工具，可以管理 tar 文件（也就是压缩包）。我们这里简单给你提供两个常用命令。

```
tar -cvf archive_name.tar /path/to/directory   // 压缩
tar -xvf archive_name.tar -C /target/directory  // 解压
```

但有时候 `.tar` 文件还会出现 `.gz`、 `.bz2`、 `.xz` 等不明后缀，这该怎么办呢?
其实只要多加一个参数即可

```
tar -xzvf archive_name.tar.gz                          //解压.gz
tar -xjvf archive_name.tar.bz2                         //解压.bz2
tar -xJvf archive_name.tar.xz                          //解压.xz
```

Important

如果你希望了解更多，请 RTFM.

在终端中输入 man tar 来查看

Tip

Why CLI?

你可能会问为什么要使用看上去繁琐的 CLI（Command Line interface, 命令行界面） 而不是更加友好的 GUI（Graphical User Interface， 图形用户界面） 呢？显然，前者能干的事情更多，作为计算机的直接操控者，CLI 会给你提供更加直接的操作（想想修改系统某处的时候，你是会选择手忙脚乱的翻 GUI,还是几行命令解决？），更何况，CLI 其实更加方便。当你熟练使用，并且配置舒服后，你会喜欢上 CLI 的。

选做：如果你喜欢一些花里胡哨的，试试自己配置 oh-my-zsh 并且安装喜欢的插件，它会改善你的体验的。

▶ 拓展

C 语言

别担心，我们这节课并不会重点讲 C 语言。我们现在只需要对在 Linux 下的 C 语言有一个了解，知道如何让他跑起来就可以了（事实上，这里面同样有很大的学问）。

Important

什么是编译?

你有没有想过你写的 c 语言程序是如何运行的？你现在可能还没有抽象这个概念（我们以后将会告诉你），但一个事实是，计算机拥有不同的层级。你现在在 c 语言这个层次编写相对易懂的代码，然后 编译器（这里是 gcc）会把他转换成汇编从而变成机器语言（这里面其实有很多的学问），然后你就会获得一个可执行的文件了。

⚠ Caution

希望看到这里的时候，你已经完成了环境的配置。接下来，我们将会尝试在你的环境下跑 c 程序。

```c
#include <stdio.h>  //这是头文件，你现在可以理解为给你的程序提供一些函数（操作）

int add(int a, int b) {     //这是一个自定义函数，功能很简单就是 a+b
    return a + b;
}

int main() {   // 这是主函数，你的代码从这里开始执行（真的是从这里开始吗，尝试 STFW）

    int a, b;        // 这里定义了 a, b 两个变量

    scanf("%d %d", &a, &b);      // 这是输入数据到 a, b 里面

    int sum = add(a, b);     // 执行函数，sum 获取 a + b 的值

    printf("The sum of %d and %d is %d\n", a, b, sum); // 输出，打印结果

    return 0;   // 返回值 0
}
```

如果你以前没有学过 c 语言（甚至没有学过任何一个编程语言），上面的代码你无需完全了解，这不是我们这节课的目的。现在把他丢到 vscode 里面，检查你的语法高亮（记得保存），然后我们准备让他跑起来。

GCC

gcc - GNU project C and C++ compiler

我们列举 gcc 的一些功能的参数如下：

-o 指定输出文件名

-Wall 开启警告

-O(2/3, ...) 优化

尝试在你的终端中（记得到你保存程序的文件里面）输入：

```
gcc -o xxx xxx.c
./xxx
```

上面的指令会进行编译，下面的就是运行了，尝试输入两个整数看看结果吧。

> **Tip**
>
> 我们还是推荐你去阅读 gcc 的手册去了解更多, 输入 man gcc 查看。当然，你也可以 STFW。
>
> 🤓我们这节的 lab 会让你进行一个简易的多文件编译，提前更多地了解 gcc 会让你更轻松

附加部分 Make

你是不是已经开始感觉 手打 gcc 是一件有点折磨的事情了，想象一下，有一个有上百个文件的项目，你要编译他们，要做很多其他的工作，是不是很麻烦？所以我们这里想你介绍 make ，追求一个命令干活。

> **Tip**
>
> 更详细的内容应该在后面会讲到，现在只是引出

尝试输入命令

```
make l1
```

他会自动处理 l1.c 并且生成 l1 可执行文件。

当然，这只是 make 的默认规则，事实上，make 有更多强大的可操作性，你可以编写自己的 Makefile 文件，制定规则，然后让 Makefile 给你 自动化实现。

尝试下在你的目录下面创建一个名为 Makefile 的文件，输入

```
CFLAGS=-Wall -g

SRCS = l1.c
TARGETS = $(SRCS:.c=)

all: $(TARGETS)

%: %.c
 $(CC) $(CFLAGS) $< -o $@

clean:
 rm -f $(TARGETS)

.PHONY: all clean
```

然后输入 make 和 make clean， 看看效果

> **Tip**
>
> 想办法让这个 Makefile 可以编译多个文件而不只是 l1.c