

Lecture 0 Git简介 & 软件的安装与配置

金融科技协会 2019年9月19日

一 计算机操作系统简介

1. 操作系统

操作系统(operating system, 简称OS), 是扮演着人与计算机物理硬件之间的中间人的程序。操作系统不是铁板一块, 而是由许多组件构成, 如内核等, 但我们可以简单将其理解为一个整体。

三大常见操作系统: ①Windows, 微软公司推出的操作系统; ②UNIX, 诞生于上世纪70年代的一款操作系统, 苹果公司的MacOS系统即为其中一个版本; ③Linux, 一种大多数服务器使用的开源操作系统, 常见的版本有CentOS、Ubuntu等。

UNIX操作系统与Linux操作系统在很多方面十分类似, 他们统称为类UNIX操作系统。需要注意, 类UNIX操作系统的路径分隔符为斜杠 “/”, Windows系统的路径分隔符为反斜杠 “\”。

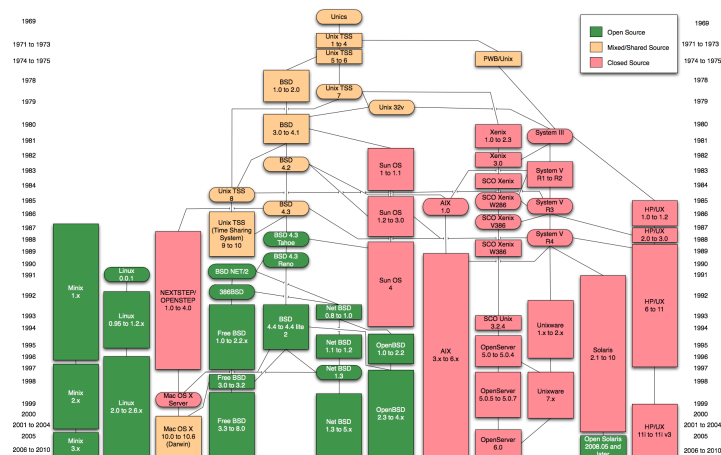


图 1: Unix History-From Wiki

2. Shell: 命令解析器

Shell是操作系统的必备组件。Shell充当人与计算机之间的“翻译官”: 人输入由自然语言构成的命令, shell将其翻译为计算机可读懂的二进制命令, 让计算机执行, 并将计算

机的执行结果翻译为人类可读懂的自然语言呈现给用户。

Shell有许多不同版本，常见的有shell有：①Bash，全称GNU Bourne-Again Shell，大部分UNIX操作系统使用，例如Mac OS；②Cmd，命令提示符，Windows系统专用；③PowerShell，Windows新一代shell。

简单来说：在windows中，shell是命令提示符；在mac中，shell是终端。它们语法稍有不同，例如which与where，dir与ls等。

3. 最常用的4个Bash命令

- cd: 切换当前工作目录至xxx，cd .. 表示返回上一层目录；
- pwd: 显示当前工作目录；
- ls: 显示当前工作目录中所有文件及文件夹名称，ls -a 显示包括隐藏的文件；
- clear: 清除屏幕，cmd是cls。

二 Git与Github的简单使用

1. 什么是Git?

Git is one of the most popular version control system(VCS).

Git是一款免费的、开源的分布式版本控制系统。什么是版本控制？简单来说，版本控制是指将每次对项目的修改，包括修改内容、修改者、修改时间等内容储存起来，以便track或roll back。总而言之，Git能够帮助个人进行项目管理，能够帮助项目成员之间更好地进行协作。

2. Git的安装

MacOS自带Git，Windows电脑需要在官网自行下载安装。Git官网地址为：<https://git-scm.com/>。安装时一路默认即可。

3. Git的初步配置

初次使用Git需要设置用户名和邮箱，即告诉Git：我是谁：

```
git config --global user.name "你的姓名"
git config --global user.email "你的邮箱"
```

查看用户名和邮箱地址：

```
git config user.name  
git config user.email
```

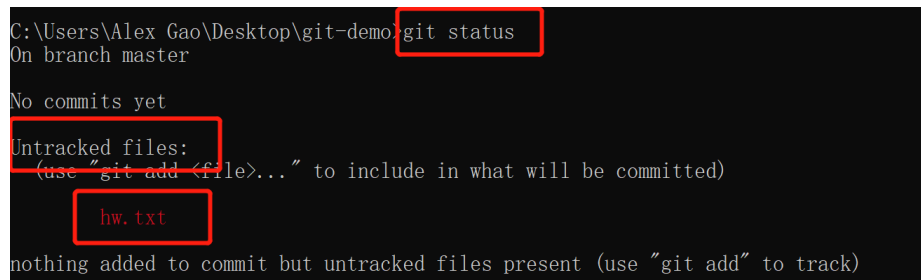
4. 在本地新建仓库并进行项目的版本管理

Git通过Repository(简称Repo)进行版本管理，一个项目就是一个Repo。

Step 1: 在桌面新建文件夹git-demo，并在其中新建名为hw.txt的文本文件(注意，需打开显示后缀名的系统选项)，键入内容“第一次提交”并保存。

Step 2: 打开shell，依次执行：cd Desktop，cd git-demo，git init——即进入项目的根目录，并为项目创建Repo。此时，我们若打开“查看隐藏文件”的选项，可以看到文件夹内多了一个名为.git的隐藏文件夹，表明Git已经为该项目创建了Repo。


Step 3: 在shell中执行git status，查看“工作区(working tree)”，即Repo文件夹的状态。发现，所有的文件或文件夹为红色，被标准为Untracked files，是Untracked状态，说明该文件或文件夹虽然在工作区内，但是不参与版本控制；此后还会遇到一种名为Modified的状态，该状态说明文件或文件夹已参与版本控制，但被修改。



```
C:\Users\Alex Gao\Desktop\git-demo>git status  
On branch master  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    hw.txt  
  
nothing added to commit but untracked files present (use "git add" to track)
```

图 2: Untracked files

Step 4: 在shell中执行git add .，将所有Untracked或Modified文件或文件夹导入缓存区(staging area)；再次执行git status，发现红色变绿色，显示Changes to be committed，说明所有修改已经进入缓存区，可以准备提交了。



```
C:\Users\Alex Gao\Desktop\git-demo>git add .  
  
C:\Users\Alex Gao\Desktop\git-demo>git status  
On branch master  
  
No commits yet  
  
Changes to be committed:  
  (use "git rm --cached <file>..." to unstage)  
    new file:   hw.txt
```

图 3: Changes to be committed

Step 5: 执行git commit -m [备注], 这里执行git commit -m "first commit", 完成第一次提交(commit)。注意, 备注不要包含中文。

```
C:\Users\Alex Gao\Desktop\git-demo>git commit -m "first commit"
[master (root-commit) 84c7384] first commit
1 file changed, 1 insertion(+)
create mode 100644 hw.txt
```

图 4: "first commit"

Step 6: 再次执行git status, 发现nothing to commit, working tree clean。

Step 7: 执行git log, 查看提交日志。特别注意commit后的一串密码, 是SHA1码, 表示此次commit。

```
C:\Users\Alex Gao\Desktop\git-demo>git status
On branch master
nothing to commit, working tree clean

C:\Users\Alex Gao\Desktop\git-demo>git log
commit 84c7384aff24b50c1dcf32acbac3ecdbed91f36 (HEAD -> master)
Author: Alex Gao <gjw2014sis@163.com>
Date: Tue Sep 17 20:25:50 2019 +0800

    first commit
```

图 5: git log

Step 8: 打开hw.txt, 加入第二行: 第二次提交, 并保存。

Step 9: 执行git status, 发现出现红色的modified状态的文件, 这是因为我们刚刚修改了Repo中的文件, 被Git察觉到了。

Step 10: 执行git add ., 再执行git commit -m "second commit"进行第二次提交。

Step 11: 执行git log, 查看提交日志。

```
C:\Users\Alex Gao\Desktop\git-demo>git log
commit 5816b0381a537c0dd9640735799d8bb19444553a (HEAD -> master)
Author: Alex Gao <gjw2014sis@163.com>
Date: Tue Sep 17 20:38:52 2019 +0800

    second commit

commit 84c7384aff24b50c1dcf32acbac3ecdbed91f36
Author: Alex Gao <gjw2014sis@163.com>
Date: Tue Sep 17 20:25:50 2019 +0800

    first commit
```

图 6: 两段提交记录

5. Branch: 分支与版本回退*

1) 分支的概念与使用

branch, 即分支, 可以将其想象为一个插在某个commit上的旗子(flag)。Git会在我们第一次做commit动作时, 创建一个branch。使用`git branch`查看所有的branch。发现, 这个创建的默认branch叫做master, 即主分支。

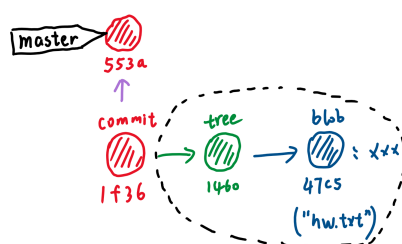


图 7: master分支

使用`git branch [分支名]`创建新的分支。这里我们执行`git branch dev`, 在第二个commit处创建名为dev的分支。效果如下:

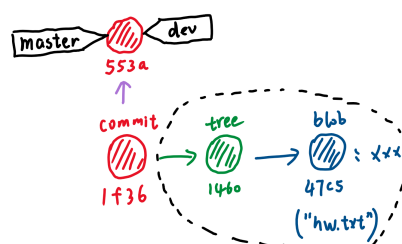


图 8: master与dev

再次使用`git branch`查看所有的branch。发现, 出现了一个新的名为dev的分支, 但“星号”仍在master分支上。“星号”表示“当前分支”, 即“Head”。星号在master上, 说明当前所在分支为master分支:

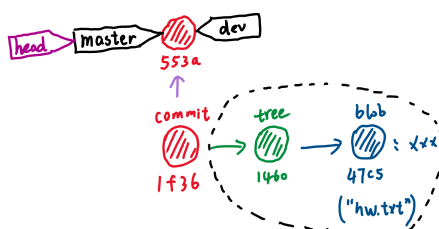


图 9: 当前分支为master

使用`git checkout [分支名]`切换分支。这里我们执行`git checkout dev`, 切换到刚刚创建的dev分支上。效果如下:

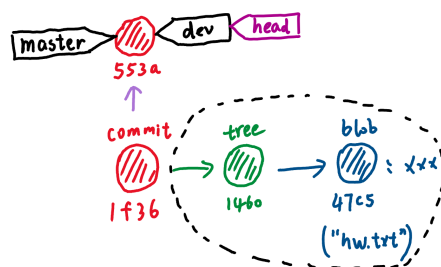


图 10: 当前分支切换为dev

修改文本文件hw.txt，加入第三行：第三次提交，并执行Step9与Step10，注意备注为third commit on dev branch。效果如下图：

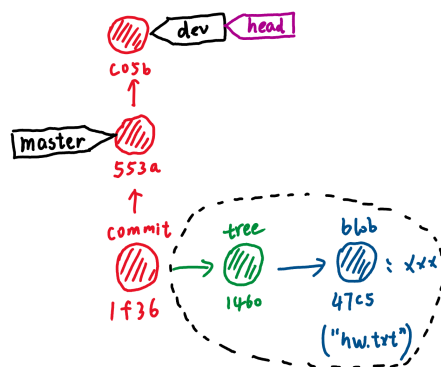


图 11: third commit on dev branch

此次提交发生在dev分支上，因此master这个旗子的位置不变，dev这个旗子被head带着到了第三次commit上。执行git checkout master，效果如下：

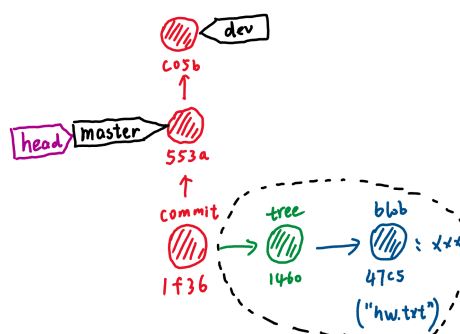


图 12: 切换到master，实现版本回退

此时打开hw.txt文件，发现文件的内容回退到了第二次提交后，即没有“第三次提交”这一行。若此时再修改hw.txt，加入：第四次提交，然后重复之前的步骤进行commit，效果如下：

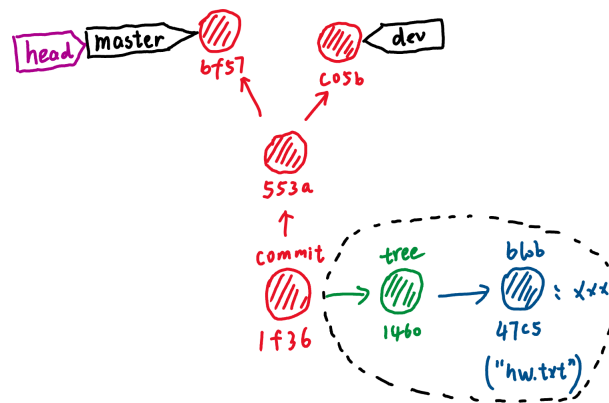


图 13: 在master分支上进行第四次commit

最后两点：

①分支合并: `git merge [另一分支名]`;

②Detached HEAD，即头部分离：我们不仅可以checkout一个分支，还可以直接checkout一个commit，例如：`git checkout 84c7384aff24b50c1dcf32acbacc3ecdbe91f36`，效果如下：

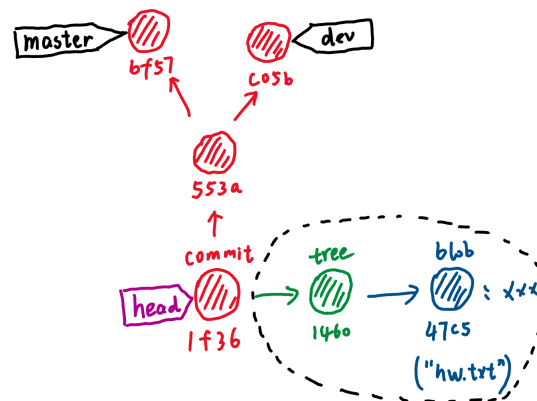


图 14: Detached HEAD

2) 分支的作用

分支是一个流程，是由无数次的修改提交(commit)构成一个时间轴。

而当你想在Repo中为你的项目加入一些新特性，而又isn't sure whether you should keep it的时候，你需要一块专门的空间来做experimental try，在这个空间里，你做的任何改动不会影响“大局”，即“master”，你可以放心对文件进行修改，直到修改到满意为止。这个空间就是你新建的分支(branch)，可以最终通过merge，将该分支合并到master上，相当于把原来在“试验田”中的新特性正式加到你的项目中。

对于单人开发，可能只需要master和develop两个分支，平时的开发在develop分支进行，开发测试完成后，在发布前将develop合并到master分支即可。

对于团队开发，可以为每个开发者创建一个分支，每个人在自己的分支上开发自己的部分，然后逐步merge到master。

因此分支相当于生产零部件，master相当于主车间里的主体，最终零部件都是要进入主车间进行装配的。

6. Github

1) 基本概念

Github是一个将代码保存在云端的网站。

Github是基于Git版本控制的代码托管平台。即，Github上上传的所有项目代码均是基于Git进行版本控制的。但Github的功能除了Git为它提供的版本控制外，还有其他许多强大的功能，比如项目协作等。

All in all, Github使用Git进行版本控制，同时提供其他强大功能。

local repo与remote repo

local repo，即本地仓库，指项目成员本地计算机上的代码仓库。remote repo，或central repo，指位于Github云端的中央代码仓库。有了“本地”和“远程”之分，则必然涉及“同步”的问题：

1. 将“远程”的修改同步到“本地”；
2. 将“本地”的修改同步到“远程”。

2) 下载他人的repo，并保持同步

Step 1: 在Github中复制该repo的地址，如下图所示：

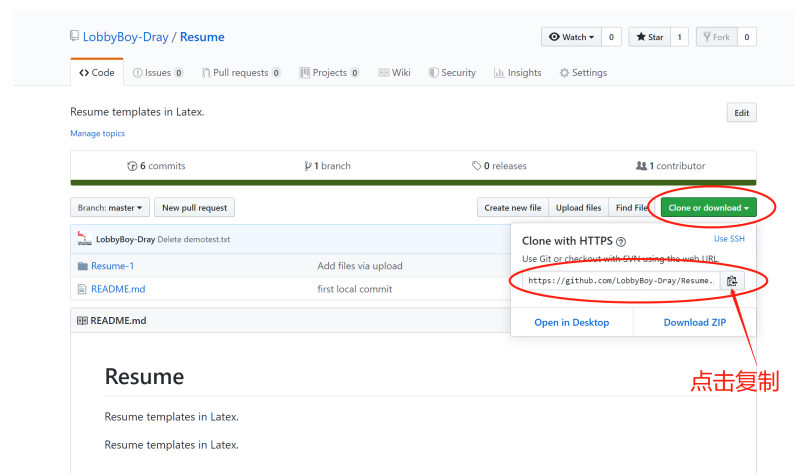


图 15: repo的地址，即repo url

Step 2: cd到目标地址，执行git clone [repo地址]。例如，下载到桌面，则先执行cd Desktop，再执行git clone https://github.com/LobbyBoy-Dray/Resume.git，效果如下图：

```
C:\Users\Alex Gao\Desktop>git clone https://github.com/LobbyBoy-Dray/Resume.git
Cloning into 'Resume'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 20 (delta 2), reused 7 (delta 1), pack-reused 0
Unpacking objects: 100% (20/20), done.
C:\Users\Alex Gao\Desktop>
```

图 16: git clone

假如repo的作者在云端对该repo进行了更新，我们需要同步更新。

Step 3: cd到该repo的根目录，这里我们执行cd Resume进入该repo的根目录，再执行git pull origin master。效果如下：

```
C:\Users\Alex Gao\Desktop\Resume>git pull origin master
From https://github.com/LobbyBoy-Dray/Resume
* branch          master      -> FETCH_HEAD
Already up to date.
```

图 17: git pull origin master

这里因为该repo并未发生改变，所以显示Already up to date，表示已为最新版。

3) 上传自己的repo，并推送更新

有两种方案：①在Github上新建repo，下载到本地；②在本地新建repo，上传至Github。这里仅介绍第一种方法。

Step 1: 在加号的下拉菜单中点击New repository。

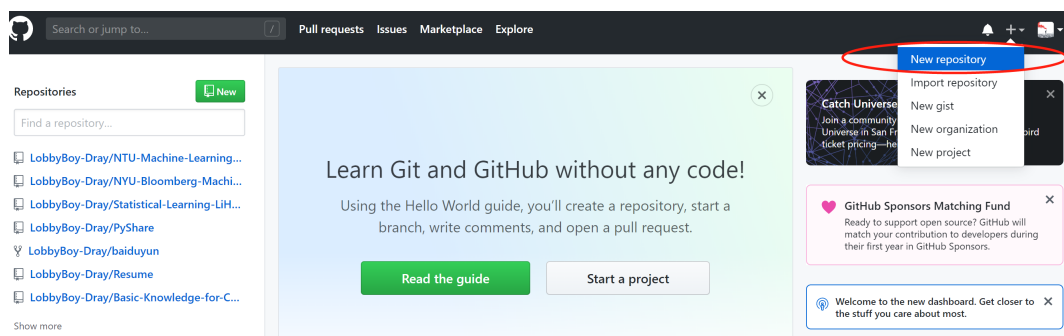


图 18: New repository

Step 2: 进入创建repo的界面，填写相关信息。

Step 3: 创建成功后自动跳转到该repo内部，将其clone到本地即可。

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: LobbyBoy-Dray / Repository name: Lyrics repo名称

Great repository names are short and memorable. Need inspiration? How about jubilant-winner.

Description (optional): This is a repo for my lyrics. repo简介

☒ Public
Anyone can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

☒ Initialize this repository with a README 是否创建readme文件，最好选中
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None ⓘ

Create repository 创建repo

图 19: create a repo

Step 4: 在本地对repo进行修改后，cd到该repo根目录，执行git push origin master即可。

4) 其他功能

- Fork
- Pull request
- Issue

三 Anaconda的安装与配置

1. 什么是Anaconda?



图 20: Anaconda

Anaconda是一个开源的Python发行版本，可以简单理解为：Anaconda = Python + conda + a lot of data science packages + tools like Jupyter Notebook

conda是一个包管理器和环境管理器(可以理解为五金店，买工具的地方)：在Python的使用，特别是数据处理和分析中，常常用到许多第三方包(类似工具)，而使用conda可以方便地帮助你安装、更新、卸载这些第三方包(因此类似于五金店)。

虽然Python官网上的Python附带pip(官方的包管理器)，但conda会更加贴心地提示一些包之间的依赖关系，避免许多麻烦。另外，conda可以进行环境管理：例如A项目使用了Python2，而B项目又要使用Python3，一台电脑同时安装两个版本Python会非常混乱，而conda可以帮你为每个项目建立一个Python环境，自己选择在此环境中使用哪个版本的Python，各环境彼此独立，互不干扰。Anaconda自带150多个科学计算包及其依赖项，如Numpy、Pandas、Scipy等，并集成了Spyder、Jupyter Notebook等工具。

2. Anaconda的安装

首先，从官网上根据自己的电脑配置下载相应的安装包。特别的，对于Win系统，Anaconda需要下载与自己电脑操作系统类型相匹配的安装包，即64位或32位。查看自己Win系统类型的方法如下：找到“此电脑”(或“我的电脑”)，右击选择属性即可。

注意：第一，安装路径最好全部都是英文，如C:\Users\DraymondGao\Anaconda3(不要用C:\Users\DraymondGao\编程\Anaconda3，其中“编程”不是英文)，否则可能报错。第二，到下面这一步时，两个都打勾：

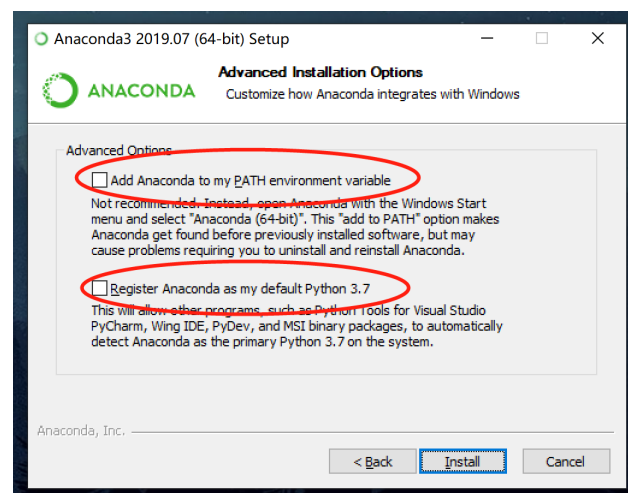


图 21: 两个都打勾

3. 包管理

推荐使用conda，而非pip。常用的命令：

- `conda list`: 查看安装的所有的包;
- `conda upgrade -all`: 将所有的包更新到最新版本;
- `conda install package_name`: 安装包;
- `conda remove package_name`: 卸载包;
- `conda update package_name`: 更新包。

4. 环境管理

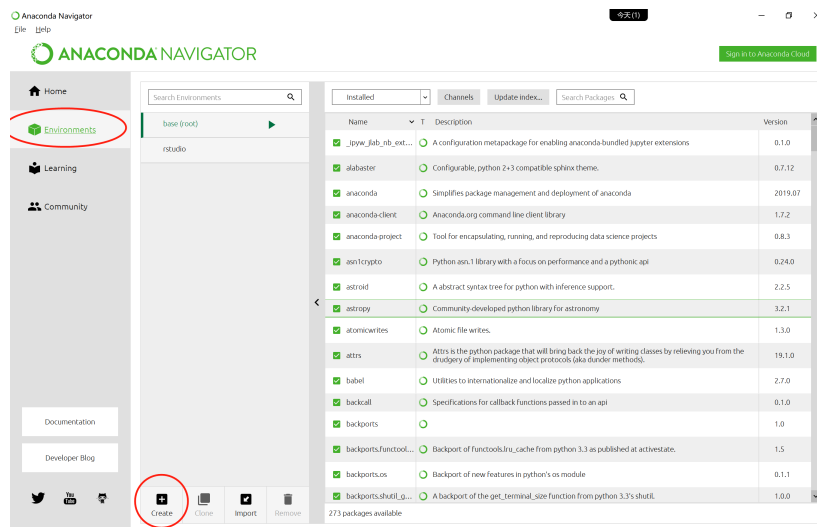


图 22: 环境管理，点击environments

使用Anaconda Navigator进行环境管理。若需要为新项目创建环境，则点击左下角的Create，跳出如下提示框：

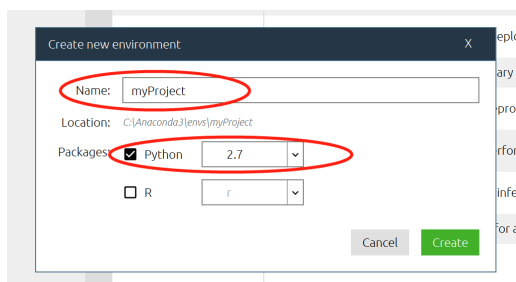


图 23: 可以为环境选择python版本

四 Visual Studio Code的安装与配置

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux.

VS Code是一款强大的轻量级代码编辑器。注意，macOS需要10.10+。下面是一些必备插件：

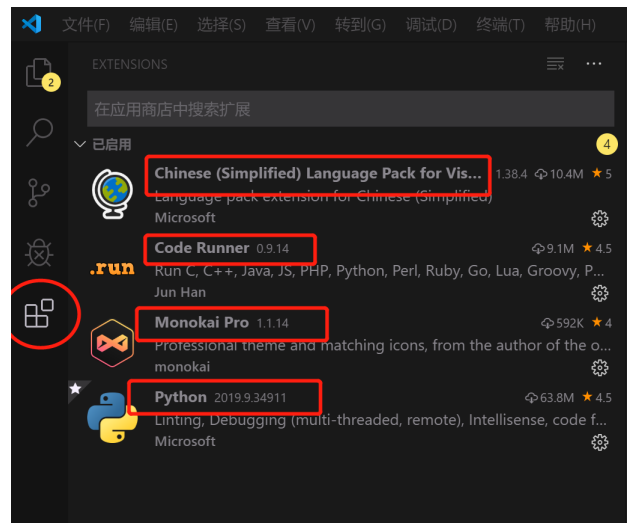


图 24: 常用插件

“VS code中python输出的中文显示为乱码”的解决方法：添加一个系统变量，详情见[第一种解决方案](<https://www.cnblogs.com/bestcode/p/9820744.html>)。