# AngularJS官网

<http://www.angularjs.net.cn/>

# Angular CLI 安装和使用

### 一、背景介绍：

两个概念：

1. 关于Angular版本，Angular官方已经统一命名Angular 1.x统称为Angular JS；Angular 2.x及以上统称Angular；
2. CLI是Command Line Interface的简写，是一种命令行接口，实现自动化开发流程，比如：ionic cli、vue cli等；它可以创建项目、添加文件以及执行一大堆开发任务，比如测试、打包和发布。

### 二、安装Angular CLI

**1. 首先确认安装了node.js和npm**

// 显示当前node和npm版本

|  |
| --- |
| $ node -v  $ npm -v |

// node 版本高于6.9.3 npm版本高于3.0.0

**2. 全局安装typescript（可选）**

|  |
| --- |
| $npm install -g typescript |

// 新建项目的时候会自动安装typescript(非全局)所以这里也可以不用安装。

**3. 安装Angular CLI**

|  |
| --- |
| $ npm install -g @angular/cli |

经过不算漫长的等待，你的Angular CLI就装好了。确认一下：

|  |
| --- |
| $ ng v  // 出现下面画面说明安装成功，如果不成功你可能需要uninstall一下，再重新来过  $ ng v  \_ \_ \_\_\_\_ \_ \_\_\_  / \ \_ \_\_ \_\_ \_ \_ \_| | \_\_ \_ \_ \_\_ / \_\_\_| | |\_ \_|  / △ \ | '\_ \ / \_` | | | | |/ \_` | '\_\_| | | | | | |  / \_\_\_ \| | | | (\_| | |\_| | | (\_| | | | |\_\_\_| |\_\_\_ | |  /\_/ \\_\\_| |\_|\\_\_, |\\_\_,\_|\_|\\_\_,\_|\_| \\_\_\_\_|\_\_\_\_\_|\_\_\_|  |\_\_\_/  @angular/cli: 1.1.1  node: 7.10.0  os: darwinx64 |

期间可能会失败（先看错误信息），可能由于node-sass这个包被墙了，所以解决办法有两个：要么用梯子，要么用淘宝镜像。

|  |
| --- |
| // 淘宝镜像解药  SASS\_BINARY\_SITE=https://npm.taobao.org/mirrors/node-sass/ npm install node-sass |

[这里可以参考](https://github.com/lmk123/blog/issues/28)

当然你也可以所有的东西都从淘宝源来安装，整个过程会快不少：

|  |
| --- |
| // 安装cnpm  npm install -g cnpm --registry=https://registry.npm.taobao.org |

### 三、新建Angular项目

**1. 新建Angular项目：**

|  |
| --- |
| $ ng new my-appng new my-app –skip-install |

这里要等很久啊，大概要下载141M东西。  
如果你已经建好了项目文件夹就可以使用ng init my-app来新建项目，ng init和ng new的区别是ng new会帮我们创建一个和项目名称相同的文件夹。

趁着它在下载，来看一下运行ng new之后Angular cli已经帮我们干了什么：

|  |
| --- |
| $ng new helloKeriy  installing ng  create .editorconfig  create README.md  create src/app/app.component.css // 使用HTML模板、CSS样式和单元测试定义AppComponent组件。 它是根组件，随着应用的成长它会成为一棵组件树的根节点。  create src/app/app.component.html  create src/app/app.component.spec.ts  create src/app/app.component.ts // 定义AppModule，这个根模块会告诉Angular如何组装该应用  create src/app/app.module.ts  create src/assets/.gitkeep // 这个文件夹下你可以放图片等任何东西，在构建应用时，它们全都会拷贝到发布包中。  create src/environments/environment.prod.ts  create src/environments/environment.ts  create src/favicon.ico // 每个网站都希望自己在书签栏中能好看一点。 请把它换成你自己的图标。  create src/index.html // 宿主页面  create src/main.ts  create src/polyfills.ts  create src/styles.css // 公共样式  create src/test.ts // 这是单元测试的主要入口点  create src/tsconfig.app.json  create src/tsconfig.spec.json  create src/typings.d.ts  create .angular-cli.json // Anguar 编译依赖  create e2e/app.e2e-spec.ts // e2e 端对端测试目录  create e2e/app.po.ts  create e2e/tsconfig.e2e.json  create .gitignore  create karma.conf.js  create package.json // Angular 的依赖包  create protractor.conf.js  create tsconfig.json // TypeScript 编译器的参数  create tslint.json  Successfully initialized git.  Installing packages for tooling via npm.  Installed packages for tooling via npm.  Project 'helloKeriy' successfully created. |

这里也可以使用淘宝源了安装：

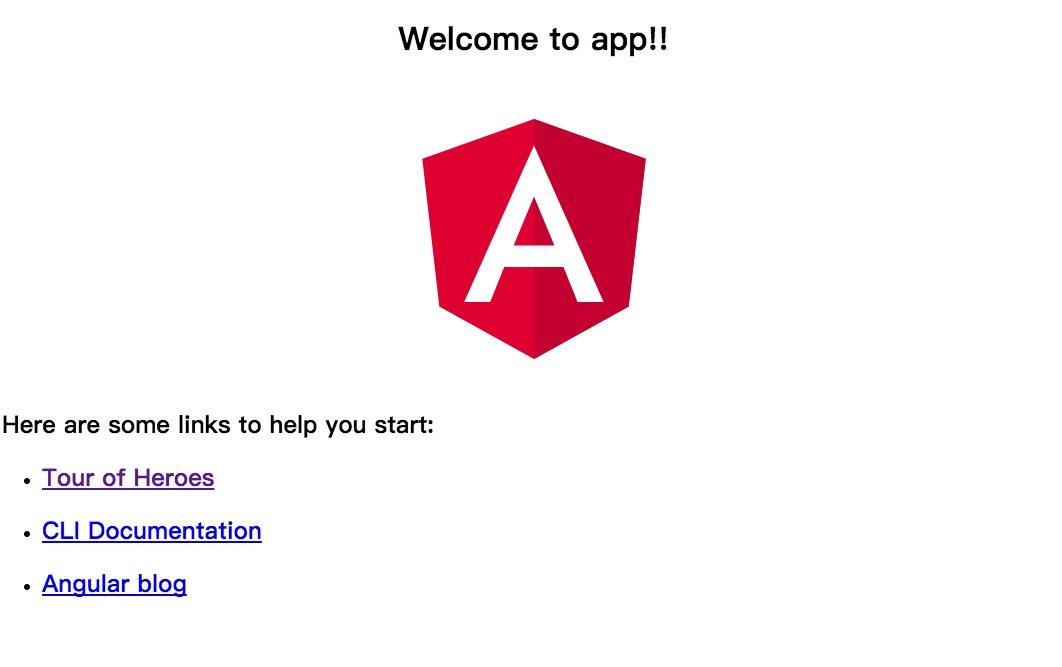
|  |
| --- |
| $ng new helloKeriy --skip-install // 先跳过npm安装  $cdhelloKeriy  $cnpm install // 使用淘宝源安装 |

那么，这时候Angular cli帮你干了以下这么多事情：

* 创建helloKeriy目录
* 应用程序相关的源文件和目录将会被创建
* 应用程序的所有依赖 (package.json中配置的依赖项) 将会被自动安装
* 自动配置项目中的TypeScript开发环境
* 自动配置Karma单元测试环境
* 自动配置Protractor (end-to-end) 测试环境
* 创建environment相关的文件并初始化为默认的设置  
  **2. 成果展示**  
  安装完成之后就可以启动项目了：

|  |
| --- |
| cd helloKeriy  ng serve -open |

ng serve命令会启动开发服务器，监听文件变化，并在修改这些文件时重新构建此应用。  
使用--open（或-o）参数可以自动打开浏览器并访问http://localhost:4200/。  
接下来你将看到：



上面展示的是我们的根组件，我们科室适当修改其中的一些信息，显示自己喜欢的内容。  
ng serve命令提供了很多参数，可以适当参考。  
以下参数仅供参考：

* --dry-run: boolean, 默认为false, 若设置dry-run则不会创建任何文件
* --verbose: boolean, 默认为false
* --link-cli: boolean, 默认为 false, 自动链接到@angular/cli包
* --skip-install: boolean, 默认为false, 表示跳过npm install
* --skip-git: boolean, 默认为false, 表示该目录不初始化为 git 仓库
* --skip-tests: boolean, 默认为false, 表示不创建tests相关文件
* --skip-commit: boolean, 默认为false, 表示不进行初始提交
* --directory: string, 用于设置创建的目录名，默认与应用程序的同名
* --source-dir: string, 默认为'src', 用于设置源文件目录的名称
* --style: string, 默认为'css', 用于设置选用的样式语法 ('css', 'less' or 'scss')
* --prefix: string, 默认为'app', 用于设置创建新组件时，组件选择器使用的前缀
* --mobile: boolean, 默认为false,表示是否生成 Progressive Web App 应用程序
* --routing: boolean, 默认为false, 表示新增带有路由信息的模块，并添加到根模块中
* --inline-style: boolean, 默认为false, 表示当创建新的应用程序时，使用内联样式
* --inline-template: boolean, 默认为false, 表示当创建新的应用程序时，使用内联模板

### 四、Angular CLI简单使用

**1. 新建组件**

$ng generate component great-angular

installing component

create src/app/great-angular/great-angular.component.css

create src/app/great-angular/great-angular.component.html

create src/app/great-angular/great-angular.component.spec.ts

create src/app/great-angular/great-angular.component.ts

update src/app/app.module.ts

如你所见，Angular cli帮我们干了如下事情：

src/app/great-angular 目录被创建  
great-angular 目录下会生成以下四个文件：  
CSS 样式文件，用于设置组件的样式  
HTML 模板文件，用于设置组件的模板  
TypeScript 文件，里面包含一个 GreatAngular 组件类和组件的元信息  
Spec 文件，包含组件相关的测试用例  
GreatAngular 组件会被自动地添加到最近模块 @NgModule装饰器的 declarations 属性中。

**2. 其他命令**

Angualr CLI提供了许多常用命令供我们选择：

ng generate class my-new-class// 新建 class

ng generate component my-new-component // 新建组件

ng generate directive my-new-directive // 新建指令

ng generate enum my-new-enum// 新建枚举

ng generate module my-new-module // 新建模块

ng generate pipe my-new-pipe // 新建管道

ng generate service my-new-service // 新建服务

nggenerate guard my-new-guard// 创建guard

nggenerate interface my-new-interface // 创建接口

当然选择。。简写：

ng g cl my-new-class // 新建 class

ng g c my-new-component // 新建组件

ng g d my-new-directive // 新建指令

ng g e my-new-enum// 新建枚举

ng g m my-new-module // 新建模块

ng g p my-new-pipe // 新建管道

ng g s my-new-service // 新建服务

ng g interface my-new-interface // 创建接口

ng g guard my-new-guard // 创建guard

[CLI git文档](https://github.com/angular/angular-cli)

**3. 单元测试**  
Angular默认帮我们集成了``karma`测试框架，我们只需要：

$ngtest

**4. 端到端测试**

$ng e2e

关于Angular测试部分详细可以参考[官方文档](https://angular.cn/docs/ts/latest/guide/testing.html)。  
**5. 构建应用程序**

$ng built

其中过程应该是这样的：  
Angular CLI 从 .angular-cli.json 文件中加载配置信息  
Angular CLI 运行 Webpack 打包项目相关的 JavaScript、 CSS 等文件  
打包后的资源，将被输出到配置文件中 outDir 所指定的目录，默认是输出到 dist 目录。

### [使用git将项目上传到github（最简单方法）](https://www.cnblogs.com/cxk1995/p/5800196.html)

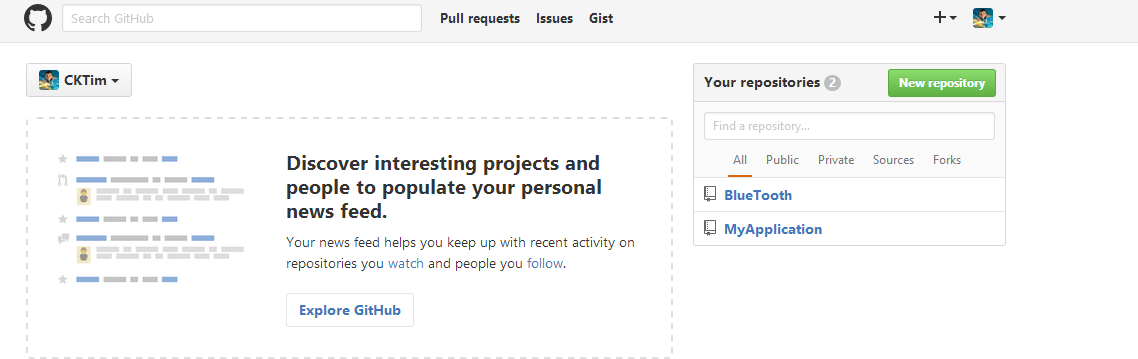
**首先你需要一个github账号，所有还没有的话先去注册吧！**

**https://github.com/**

**我们使用git需要先安装git工具，这里给出下载地址，下载后一路直接安装即可：**

**https://git-for-windows.github.io/**

**1.进入Github首页，点击New repository新建一个项目**



**2.填写相应信息后点击create即可**

**Repository name: 仓库名称**

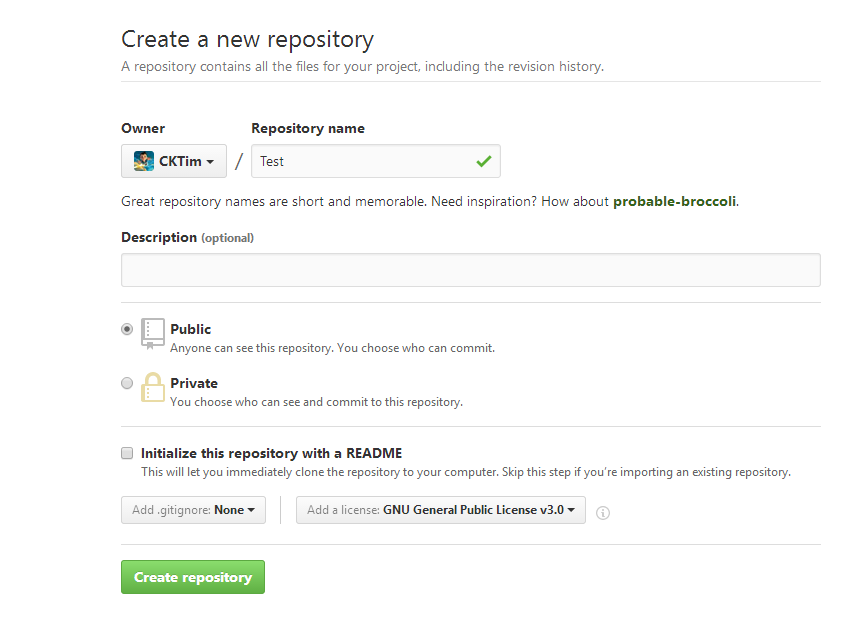
**Description(可选): 仓库描述介绍**

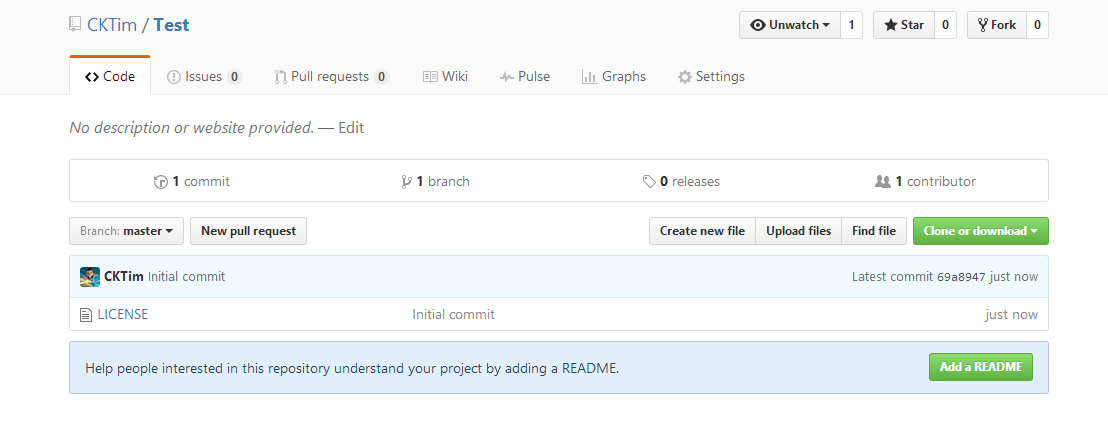
**Public, Private : 仓库权限（公开共享，私有或指定合作者）**

**Initialize this repository with a README: 添加一个README.md**

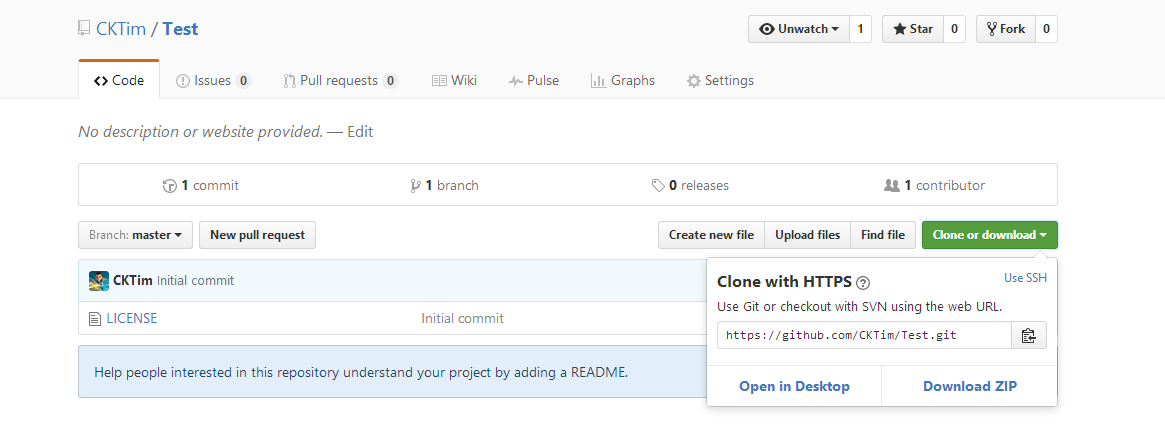
**gitignore: 不需要进行版本管理的仓库类型，对应生成文件.gitignore**

**license: 证书类型，对应生成文件LICENSE**

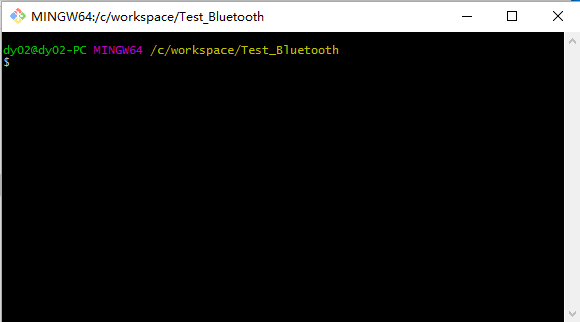




**4.点击Clone or dowload会出现一个地址，copy这个地址备用。**

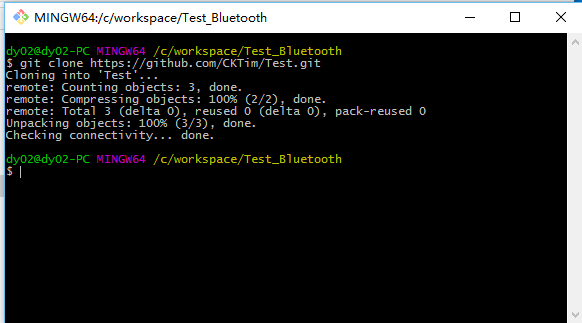


**5.接下来就到本地操作了，首先右键你的项目，如果你之前安装git成功的话，右键会出现两个新选项，分别为GitGuiHere,Git Bash Here,这里我们选择Git Bash Here，进入如下界面，Test\_Bluetooth即为我的项目名。**

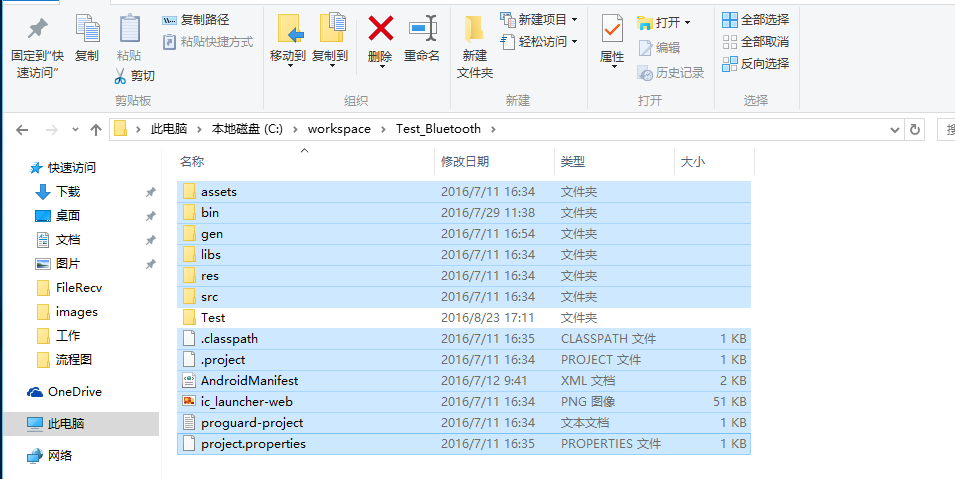


**6.接下来输入如下代码（关键步骤），把github上面的仓库克隆到本地**

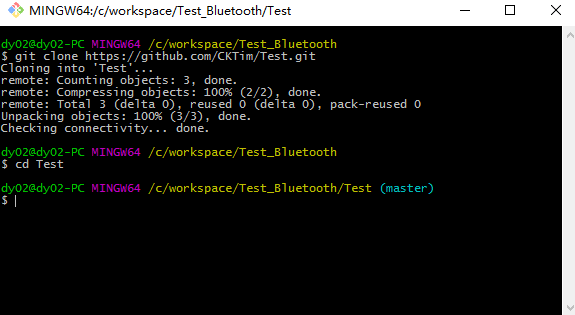
**git clone https://github.com/CKTim/BlueTooth.git（https://github.com/CKTim/BlueTooth.git替换成你之前复制的地址）**



**7.这个步骤以后你的本地项目文件夹下面就会多出个文件夹，该文件夹名即为你github上面的项目名，如图我多出了个Test文件夹，我们把本地项目文件夹下的所有文件（除了新多出的那个文件夹不用），其余都复制到那个新多出的文件夹下，**



**8.接着继续输入命令 cd Test，进入Test文件夹**

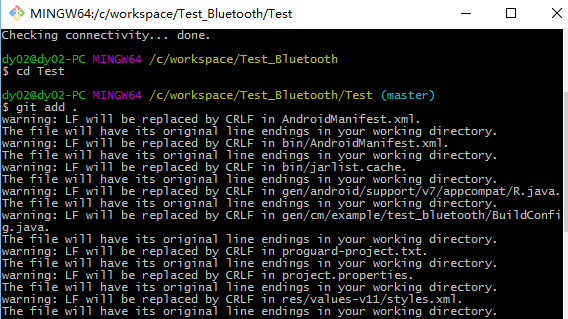


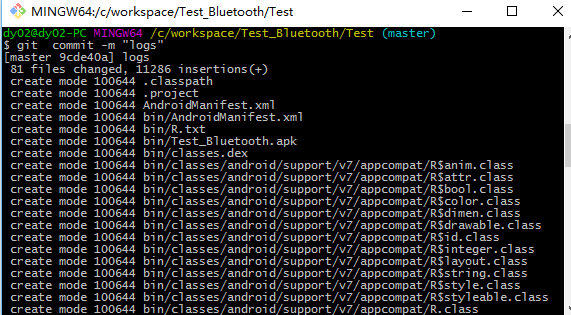
**9.接下来依次输入以下代码即可完成其他剩余操作：**

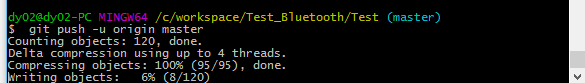
**git add .        （注：别忘记后面的.，此操作是把Test文件夹下面的文件都添加进来）**

**git commit  -m  "提交信息"  （注：“提交信息”里面换成你需要，如“first commit”）**

**git push -u origin master   （注：此操作目的是把本地仓库push到github上面，此步骤需要你输入帐号和密码）**







### git更新代码到本地

正规流程

git status（查看本地分支文件信息，确保更新时不产生冲突）

git checkout – [file name] （若文件有修改，可以还原到最初状态; 若文件需要更新到服务器上，应该先merge到服务器，再更新到本地）

git branch（查看当前分支情况）

git checkout remote branch (若分支为本地分支，则需切换到服务器的远程分支)

git pull

若命令执行成功，则更新代码成功！

快速流程

上面是比较安全的做法，如果你可以确定什么都没有改过只是更新本地代码

1. git pull (一句命令搞定)

git branch 看看分支

gitchechoutaaa切换分支aaa

gitbranckaaa创建aaa分支

gitchechout -b aaa本地创建aaa分支，同时切换到aaa分支。只有提交的时候才会在服务端上创建一个分支

**Angular6-hmr开发配置**

### 创建HMR的环境

在environments文件夹中新建 src/environments/environment.hmr.ts，内容分别配置为：

exportconst environment = {

production: false,

hmr: true

};

将 src/environments/environment.prod.ts 中设置属性hmr: false ，

exportconst environment = {

production: true,

hmr: false

};

将 src/environments/environment.ts 中设置属性hmr: false ，

exportconst environment = {

production: false,

hmr: false

};

 将angular.json文件中配置HMR环境，并在构建和服务中添加配置以启用HMR。

  Tip:<project-name>这里表示在angular.json中添加此配置的项目的名称。

  "build": {

    "configurations": {

      ...

      "hmr": {

        "fileReplacements": [

          {

            "replace": "src/environments/environment.ts",

            "with": "src/environments/environment.hmr.ts"

          }

        ]

      }

    }

  },

  ...

  "serve": {

    "configurations": {

      ...

      "hmr": {

        "hmr": true,

        "browserTarget": "<project-name>:build:hmr"

      }

    }

  }

在 src/tsconfig.app.json中添加types必要的类型

{

  ...

  "compilerOptions": {

    ...

    "types": ["node"]

  },

}

在终端中执行以下命令，即可将项目以hmr环境运行

ng serve --configuration hmr

或者在package.json文件中添加script对象，用来便捷启动程序：

"scripts": {

  ...

  "hmr": "ng serve --configuration hmr"

}

### 添加@ angurlarclass/HMR依赖

为了使HMR工作，我们需要安装依赖项并配置我们的应用程序来使用它，

$ npm install --save-dev @angularclass/hmr

创建 src/hmr.ts文件， 内容如下:

import { NgModuleRef, ApplicationRef } from'@angular/core';

import { createNewHosts } from'@angularclass/hmr';

exportconsthmrBootstrap= (module:any, bootstrap: () =>Promise<NgModuleRef<any>>) => {

  letngModule:NgModuleRef<any>;

  module.hot.accept();

  bootstrap().then(mod=>ngModule= mod);

  module.hot.dispose(() => {

    constappRef:ApplicationRef=ngModule.injector.get(ApplicationRef);

    const elements =appRef.components.map(c=>c.location.nativeElement);

    constmakeVisible=createNewHosts(elements);

    ngModule.destroy();

    makeVisible();

  });

};

将原src/main.ts中的内容替换如下，hmr就可以正常工作啦！！！ :

import { enableProdMode } from'@angular/core';

import { platformBrowserDynamic } from'@angular/platform-browser-dynamic';

import { AppModule } from'./app/app.module';

import { environment } from'./environments/environment';

import { hmrBootstrap } from'./hmr';

if (environment.production) {

  enableProdMode();

}

const bootstrap = () =>platformBrowserDynamic().bootstrapModule(AppModule);

if (environment.hmr) {

  if (module[ 'hot' ]) {

    hmrBootstrap(module, bootstrap);

  } else {

    console.error('HMR is not enabled for webpack-dev-server!');

    console.log('Are you using the --hmr flag for ng serve?');

  }

} else {

  bootstrap().catch(err=>console.log(err));

}

### 以上步骤配置完成，终端执行hmr命令，

$ npm run hmr

当在控制台中看到以下提醒时，说明hmr已经在工作啦～！

NOTICE Hot Module Replacement (HMR) is enabled for the dev server.

# Git恢复之前版本的两种方法reset、revert（图文详解）

## ****一、问题描述****

在利用github实现多人合作程序开发的过程中，我们有时会出现**错误提交**的情况，此时我们希望能撤销提交操作，让程序回到提交前的样子，本文总结了两种解决方法：回退（reset）、反做（revert）。

## ****二、背景知识****

#### ****git的版本管理，及HEAD的理解****

使用git的每次提交，Git都会自动把它们串成一条时间线，这条时间线就是一个分支。如果没有新建分支，那么只有一条时间线，即只有一个分支，在Git里，这个分支叫主分支，即master分支。有一个HEAD指针指向当前分支（只有一个分支的情况下会指向master，而master是指向最新提交）。每个版本都会有自己的版本信息，如特有的版本号、版本名等。如下图，假设只有一个分支：

## ****三、解决方法****

### ****方法一：git reset****

**原理：** git reset的作用是修改HEAD的位置，即将HEAD指向的位置改变为之前存在的某个版本，如下图所示，假设我们要回退到版本一：  
  
**适用场景：** 如果想恢复到之前某个提交的版本，且那个版本之后提交的版本我们都不要了，就可以用这种方法。

#### ****具体操作：****

**1. 查看版本号：**  
可以使用命令“git log”查看：  
  
也可以在github网站上查看：  
  
**2. 使用“git reset --hard 目标版本号”命令将版本回退：**  
  
再用“git log”查看版本信息，此时本地的HEAD已经指向之前的版本：  
  
**3. 使用“git push -f”提交更改：**  
此时如果用“git push”会报错，因为我们本地库HEAD指向的版本比远程库的要旧：  
  
所以我们要用“git push -f”强制推上去，就可以了：  
  
在github图形化界面上看，远程库的HEAD也已经指向目标版本：

#### ****回退成功！****

### ****方法二：git revert****

**原理：** git revert是用于“反做”某一个版本，以达到撤销该版本的修改的目的。比如，我们commit了三个版本（版本一、版本二、 版本三），突然发现版本二不行（如：有bug），想要撤销版本二，但又不想影响撤销版本三的提交，就可以用 git revert 命令来反做版本二，生成新的版本四，这个版本四里会保留版本三的东西，但撤销了版本二的东西。如下图所示：  
  
**适用场景：** 如果我们想撤销之前的某一版本，但是又想保留该目标版本后面的版本，记录下这整个版本变动流程，就可以用这种方法。

#### ****具体操作：****

举个例子，现在库里面有三个文件：[READ.md](http://READ.md)、text.txt、text2.txt。  
  
**1. 查看版本号：**  
可以通过命令行查看（输入git log）：  
如图，最近的两个版本分别叫：“add text.txt”（即新增了文件text.txt）、“add text2.txt”（新增了文件text2.txt）。这个时候我们不需要text.txt这个文件了，那就是说不想要“add text.txt”那个版本的操作，那可以通过反做“add text.txt”这个版本来实现。  
  
也可以通过github网站图形化界面查看版本号：  
  
**2.使用“git revert -n 版本号”反做，并使用“git commit -m 版本名”提交：**  
（1）反做，使用“git revert -n 版本号”命令。如下命令，我们反做版本号为8b89621的版本：

git revert -n 8b89621019c9adc6fc4d242cd41daeb13aeb9861

* 1

**注意：** 这里可能会出现冲突，那么需要手动修改冲突的文件。而且要git add 文件名。  
（2）提交，使用“git commit -m 版本名”，如：

git commit -m "revert add text.txt"

* 1

此时可以用“git log”查看本地的版本信息，可见多生成了一个新的版本，该版本反做了“add text.txt”版本，但是保留了“add text2.txt”版本：  
  
**3.使用“git push”推上远程库：**

git push

* 1

查看github上显示的远程库版本信息：  
  
此时查看仓库的文件，剩下两个：[READ.md](http://READ.md)、text2.txt

#### ****反做成功！****

# git commit之后，想撤销commit

写完代码后，我们一般这样

git add . //添加所有文件

git commit -m "本功能全部完成"

执行完commit后，想撤回commit，怎么办？

这样凉拌：

**git reset --soft HEAD^**

这样就成功的撤销了你的commit

注意，仅仅是撤回commit操作，您写的代码仍然保留。

## 说一下个人理解：

HEAD^的意思是上一个版本，也可以写成HEAD~1

如果你进行了2次commit，想都撤回，可以使用HEAD~2

## 至于这几个参数：

## --mixed

意思是：不删除工作空间改动代码，撤销commit，并且撤销git add . 操作

这个为默认参数,git reset --mixed HEAD^ 和 git reset HEAD^ 效果是一样的。

## --soft

不删除工作空间改动代码，撤销commit，不撤销git add .

## --hard

删除工作空间改动代码，撤销commit，撤销git add .

注意完成这个操作后，就恢复到了上一次的commit状态。

### 顺便说一下，如果commit注释写错了，只是想改一下注释，只需要：

git commit --amend

此时会进入默认vim编辑器，修改注释完毕后保存就好了。