**Git**

**一、Git基础**

**1、Git介绍**

Git是目前世界上最先进的分布式版本控制系统。

**2、Git与Github**

**2.1、两者区别**

Git是一个分布式版本控制系统，简单的说其就是一个软件，用于记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的软件。

Github（https://www.github.com）是一个为用户提供Git服务的网站，简单说就是一个可以放代码的地方（不过可以放的当然不仅是代码）。Github除了提供管理Git的web界面外，还提供了订阅、关注、讨论组、在线编辑器等丰富的功能。Github被称之为全球最大的基友网站。

**2.2、Github注册**

打开Github官网：https://github.com/，点击右上角的“Sign up”按钮。

选择免费的账户类型：

提示我们需要验证邮箱：

打开邮箱中收到的邮件，点击按钮进行验证：

至此，Github帐号注册完毕，我们将在后面会使用到本次注册的帐号。

**3、Git安装**

①下载得到安装包，并运行

②选择软件的安装位置

③选择需安装的组件（默认即可，直接下一步）

④选择使用的编辑器（默认即可，直接下一步）

⑤环境变量调节

⑥使用OpenSSH，直接下一步即可

⑦使用OpenSSL库

⑧配置命令行会话（默认即可）

⑨配置终端（默认即可）

⑩配置额外的选项（默认即可）

安装完成

在桌面空白处右键鼠标，若出现“Git GUI Here”、“Git Bash Here”则安装成功：

**二、Git的使用**

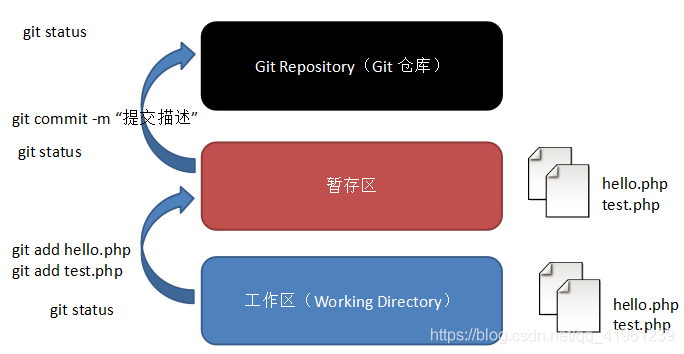
**1、本地仓库**

**1.1、工作流程**

Git本地操作的三个区域：



**工作流程：**



**1.2、本地仓库操作**

什么是仓库呢？仓库又名版本库，英文名repository，我们可以简单理解成是一个目录，用于存放代码的，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除等操作Git都能跟踪到。

**①在安装好后首次使用需要先进行全局配置**

桌面空白处右键，点击“Git Bash Here”以打开Git命令行窗口

$ git config --global user.name "用户名"

$ git config --global user.email "邮箱地址"

**②创建仓库**

当我们需要让Git去管理某个新项目/已存在项目的时候，就需要创建仓库了。**注意，创建仓库时使用的目录不一定要求是空目录，选择一个非空目录也是可以的，但是不建议在现有项目上来学习Git，否则造成的一切后果概不负责！**

**注意：为了避免在学习或使用过程中出现各种奇葩问题，请不要使用包含中文的目录名（父目录亦是如此）。**

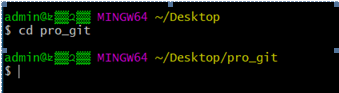
a. 创建空目录—或者手动创建文件夹即可

**mkdir 文件名**



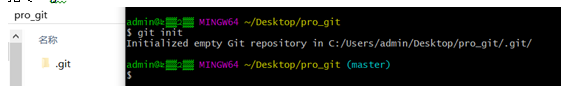
b. 在命令行中进入项目目录pro\_git

**cd 文件名**



c. Git仓库初始化（让Git知道，它需要来管理这个目录）

指令：**git init**



表现：执行之后会在项目目录下创建“.git”的隐藏目录，这个目录是Git所创建的，不能删除，也不能随意更改其中的内容。

拓展：

**③Git常用指令操作**

查看当前状态：git status 【非必要】

**添加到缓存区：git add 文件名**

新建好文件然后可以进行简单的文件提交到远程仓库

说明：git add指令，可以添加一个文件，也可以同时添加多个文件。

语法1：git add 文件名

语法2：git add 文件名1 文件名2 文件名3 …

语法3：git add . 【添加当前目录到缓存区中】

**提交至版本库：git commit -m “注释内容”**

在后续对于文件（可以操作1个或多个）操作之后，重复使用git add与git commit指令即可。

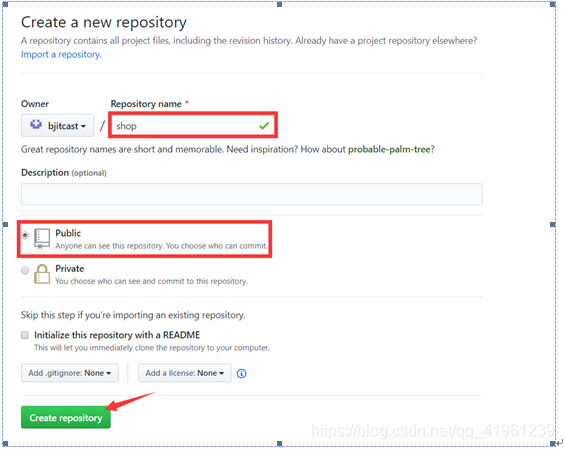
**2、远程仓库**

线上仓库的操作学习以Github为例。

**2.1、线上仓库创建**

打开创建仓库页面：https://github.com/new

圈出的部分为必填项，其余根据实际需要选择性补充：

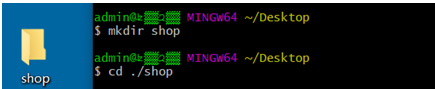


**注意：仓库名要求在当前帐号下唯一。**

**2.2、两种常规使用方式**

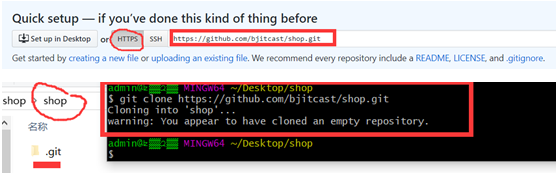
**2.2.1、基于http/https协议**

1. 创建空目录，名称就称为shop



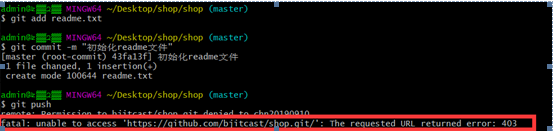
b. 使用clone指令克隆线上仓库到本地

**语法：git clone 线上仓库地址**



c. 在仓库上做对应的操作（提交暂存区、提交本地仓库、提交线上仓库、拉取线上仓库）

**提交到线上仓库的指令：git push**



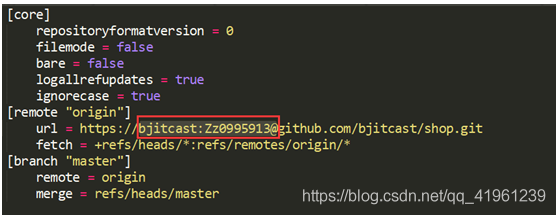
在首次往线上仓库shop提交内容的时候出现了403的致命错误，原因是不是任何人都可以往线上仓库提交内容，必须需鉴权。

(如果不加可以的话那就不加)

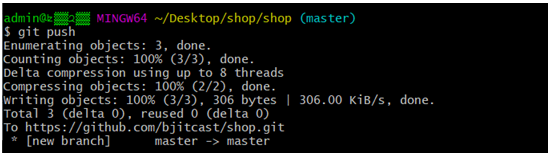
需要修改“.git/config”文件内容：



例如：



在设置好用户名密码之后再次尝试push指令：



如果看到类似上述效果（没有fatal错误）则表示提交成功。

【验证】此时可以观察浏览器，刷新线上仓库的地址：

**拉取线上仓库：git pull**

**提醒：**

**在每天工作的第一件事就是先git pull拉取线上最新的版本；每天下班前要做的是git push，将本地代码提交到线上仓库。**

**另一种上传代码的方式**

远程仓库早已创建完毕，新创建的文件需要关联之前的远程文件

**1.初始化项目 git init**

初始化成功后你会发现项目里多了一个隐藏文件夹.git

**2.执行指令：git add .**

将所有文件添加到仓库

**3.执行指令：git commit -m "提交文件"**

双引号内是提交注释。

提交文件

**4 关联github仓库**

**git remote add origin 仓库地址**

**5.上传本地代码：git push -u origin master**

**2.2.2、基于ssh协议（推荐）**

该方式与前面https方式相比，只是影响github对于用户的身份鉴权方式，对于git的具体操作（如提交本地、添加注释、提交远程等操作）没有任何影响。

生成公私玥对指令（需先自行安装OpenSSH）：ssh-keygen -t rsa -C “注册邮箱”

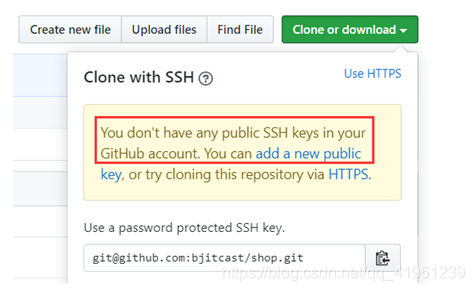
步骤：

**①生成客户端公私玥文件**

**②将公钥上传到Github**

实际操作：

①打开提示



②创建公私玥对文件



③上传公钥文件内容（id\_rsa.pub）



填写完毕之后保存即可。

④执行后续git操作，操作与先前一样

a. clone线上仓库到本地（git clone）

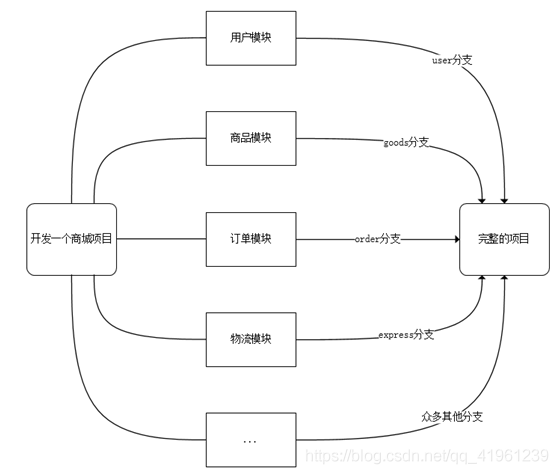
b. 修改文件后添加缓存区、提交本地仓库、提交线上仓库

在push的时候并没有提示要求我们输入帐号密码，因为公私玥已经实现了用户身份鉴权。

线上仓库的效果：

**2.3、分支管理**

什么是分支？



在版本回退的章节里，每次提交后都会有记录，Git把它们串成时间线，形成类似于时间轴的东西，这个时间轴就是一个分支，我们称之为master分支。

在开发的时候往往是团队协作，多人进行开发，因此光有一个分支是无法满足多人同时开发的需求的，并且在分支上工作并不影响其他分支的正常使用，会更加安全，Git鼓励开发者使用分支去完成一些开发任务。

1分支相关指令：

2查看分支：git branch

3创建分支：git branch 分支名

4切换分支：git checkout 分支名

5删除分支：git branch -d 分支名

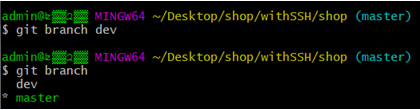
6合并分支：git merge 被合并的分支名

查看分支：

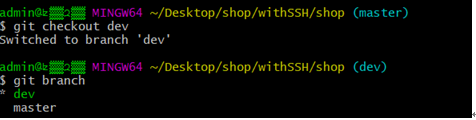
IMG_256

注意：当前分支前面有个标记“\*”。

创建分支：

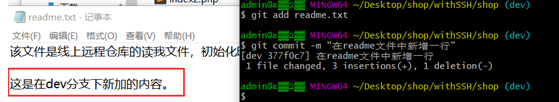


切换分支：

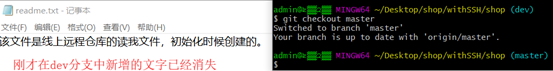


合并分支：

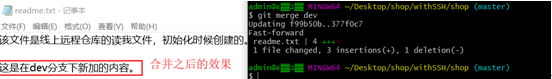
现在先在dev分支下的readme文件中新增一行并提交本地



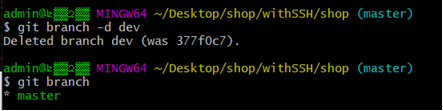
切换到master分支下观察readme文件



将dev分支的内容与master分支合并：



删除分支：



注意：在删除分支的时候，一定要先退出要删除的分支，然后才能删除。

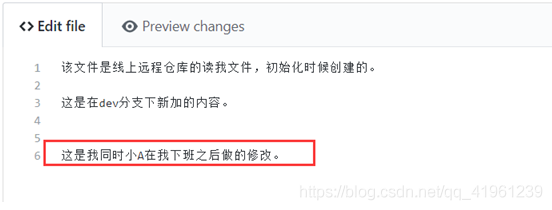
合并所有分支之后，需要将master分支提交线上远程仓库中：



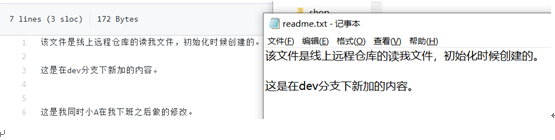
**2.4、冲突的产生与解决**

案例：模拟产生冲突。

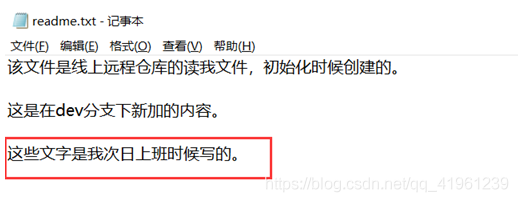
①同事在下班之后修改了线上仓库的代码



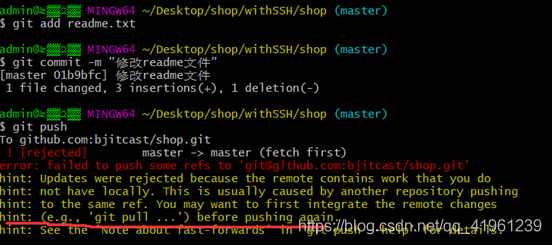
注意：此时我本地仓库的内容与线上不一致的。



②第二天上班的时候，我没有做git pull操作，而是直接修改了本地的对应文件的内容



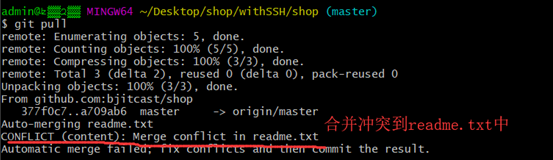
③需要在下班的时候将代码修改提交到线上仓库（git push）



提示我们要在再次push之前先git pull操作。

【解决冲突】

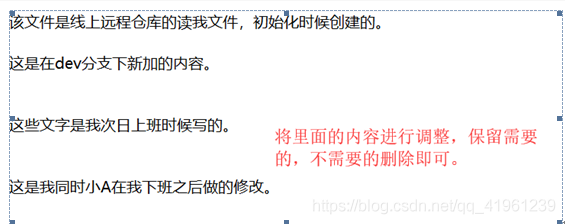
④先git pull



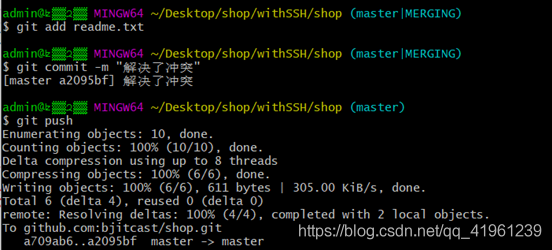
此时git已经将线上与本地仓库的冲突合并到了对应的文件中。

⑤打开冲突文件，解决冲突

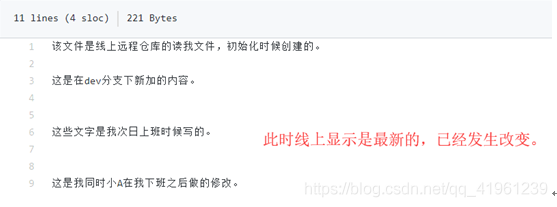
解决方法：需要和同事（谁先提交的）进行商量，看代码如何保留，将改好的文件再次提交即可。



⑥重新提交



线上效果：

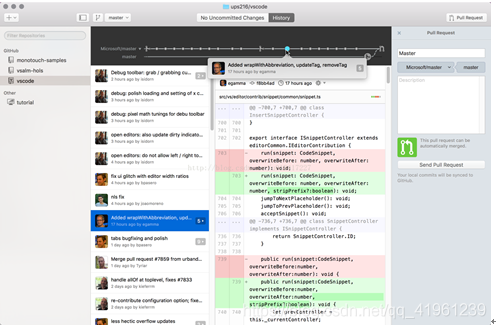


**新手上路小技巧：上班第一件事先git pull，可以在一定程度上避免冲突的产生。**

**三、Git实用技能**

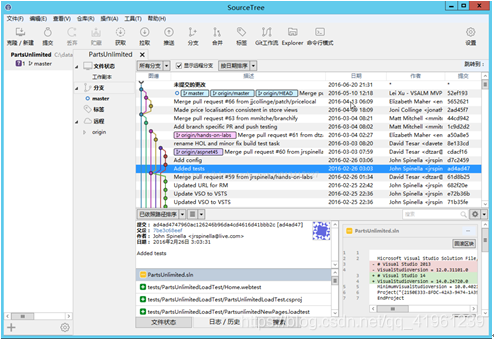
**1、图形管理工具**

**①Github for Desktop**



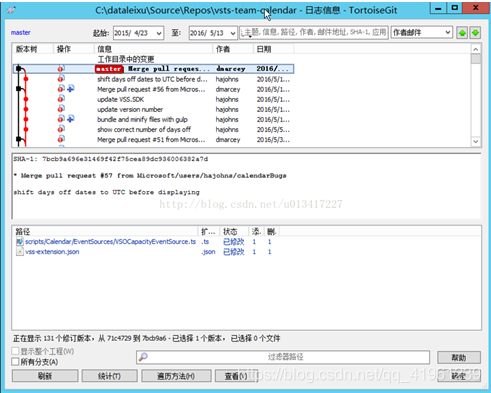
Github出品的软件，功能完善，使用方便。对于经常使用GitHub的开发人员来说是非常便捷的工具。界面干净，用起来非常顺手，顶部的分支时间线非常绚丽。

**②Source tree**



老牌的Git GUI管理工具了，也号称是最好用的Git GUI工具。功能丰富，基本操作和高级操作都非常流畅，适合初学者上手。

**③TortoiseGit**



对于熟悉SVN的开发人员来说，这个小乌龟图标应该是非常友善了。TortoiseGit 简称 tgit, 中文名海龟Git。它与其前辈TortoiseSVN都是非常优秀的开源版本控制客户端软件。

**2、忽略文件**

场景：在项目目录下有很多万年不变的文件目录，例如css、js、images等，或者还有一些目录即便有改动，我们也不想让其提交到远程仓库的文档，此时我们可以使用“忽略文件”机制来实现需求。

忽略文件需要新建一个名为.gitignore的文件，该文件用于声明忽略文件或不忽略文件的规则，规则对当前目录及其子目录生效。

注意：该文件因为没有文件名，没办法直接在windows目录下直接创建，可以通过命令行Git Bash来touch创建。

常见规则写法有如下几种：

1）/mtk/ 过滤整个文件夹

2）\*.zip 过滤所有.zip文件

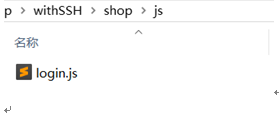
3）/mtk/do.c 过滤某个具体文件

4) !index.php 不过滤具体某个文件

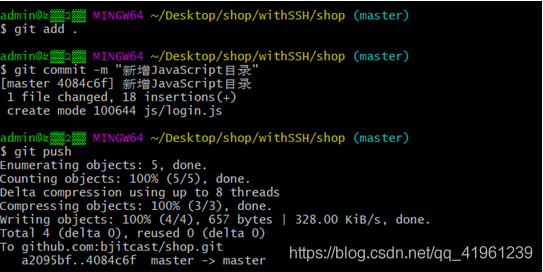
在文件中，以#开头的都是注释。

案例：

①先在本地仓库中新建一个js目录以及目录中js文件



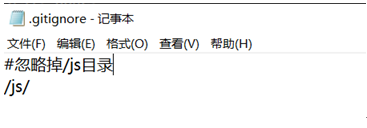
②依次提交本地与线上



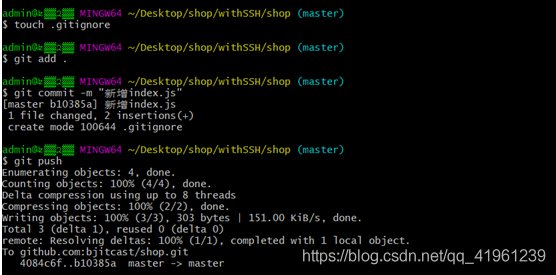
③新增.gitignore文件

IMG_256

④编写文件中的规则（根据需要编写）



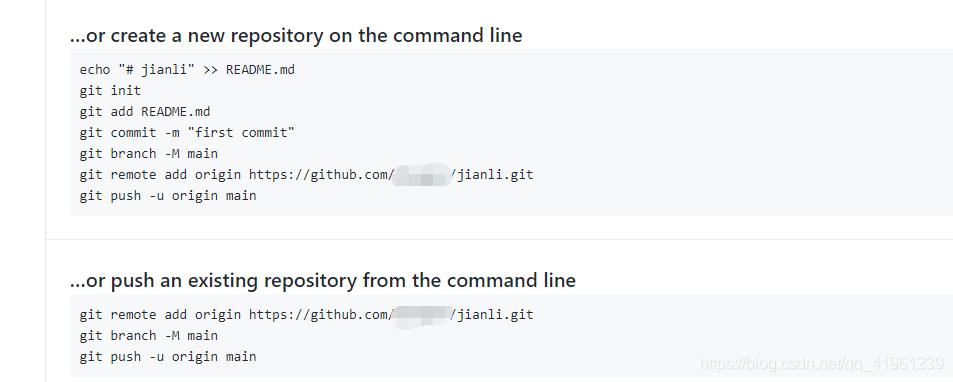
⑤再次提交本地与线上



观察线上仓库js目录中是否有新增index.js文件：



如果本地项目想上传到GitHub之前的项目上面，需要关联之前的连接



即可