|  |
| --- |
|  |
| **Git&GitHub** |
|  |

|  |
| --- |
| zhao ying  2078-5-31 |

目录

[摘要 I](#_Toc515524687)

[1. Git简介 1](#_Toc515524688)

[2.Git安装及设置 3](#_Toc515524689)

[2.1 Centos上安装Git 3](#_Toc515524690)

[2.2 在Windows上安装Git 3](#_Toc515524691)

[2.3 初次运行Git前的设置 4](#_Toc515524692)

[3. Git仓库基本操作 5](#_Toc515524693)

[3.1 获取Git仓库 5](#_Toc515524694)

[3.1.1克隆现有的仓库 5](#_Toc515524695)

[3.1.2在工作目录中初始化新仓库 6](#_Toc515524696)

[3.2 记录每次更新到仓库 6](#_Toc515524697)

[3.3远程仓库的管理和使用 9](#_Toc515524698)

[4. GitHub简介 12](#_Toc515524699)

[4.1 GitHub用户申请及设置 12](#_Toc515524700)

[4.2 创建GitHub仓库 14](#_Toc515524701)

[4.3使用仓库 15](#_Toc515524702)

[5. Git与GitHub的联动使用 22](#_Toc515524703)

[5.1 克隆一个GitHub仓库并实现Git与GitHub联动使用 24](#_Toc515524704)

[5.1.1克隆现有的仓库 24](#_Toc515524705)

[5.1.2 记录每次更新到本地及远程仓库 25](#_Toc515524706)

[5.1 本地Git仓库内容上传至GitHub仓库 26](#_Toc515524707)

图目录

[Figure 1．SVN VS Git 2](#_Toc515524719)

[Figure 2．Git项目的三个工作区域 3](#_Toc515524720)

[Figure 3．Git for Windows 4](#_Toc515524721)

[Figure 4． Windows上Git安装成果的标志 4](#_Toc515524722)

[Figure 5．通过git clone命令克隆一个现存GitHub仓库 5](#_Toc515524723)

[Figure 6．初始化新仓库 6](#_Toc515524724)

[Figure 7．初始化成仓库后文件夹状态 6](#_Toc515524725)

[Figure 8．git add命令使用举例 7](#_Toc515524726)

[Figure 9．git commit 命令使用举例 7](#_Toc515524727)

[Figure 10．git status命令使用举例 7](#_Toc515524728)

[Figure 11．修改已追踪文件的提示 7](#_Toc515524729)

[Figure 12．git diff使用方法 8](#_Toc515524730)

[Figure 13．再次修改被追踪文件内容并提交 8](#_Toc515524731)

[Figure 14．git log查看提交历史 9](#_Toc515524732)

[Figure 15．git log 其他使用方法 9](#_Toc515524733)

[Figure 16．Git基本操作实例 11](#_Toc515524734)

[Figure 17．图解Git基本操作之间的关系 12](#_Toc515524735)

[Figure 18．GitHub官网页面 13](#_Toc515524736)

[Figure 19．GitHub设置 13](#_Toc515524737)

[Figure 20．创建Git仓库的方式 14](#_Toc515524738)

[Figure 21. new repository表单 15](#_Toc515524739)

[Figure 22．GitHub上一个名为“hello-world”仓库的主页面 16](#_Toc515524740)

[Figure 23. Create new file 17](#_Toc515524741)

[Figure 24．Upload files 17](#_Toc515524742)

[Figure 25．分支工作原理示意图 18](#_Toc515524743)

[Figure 26．建立一个名为“feature”的分支 18](#_Toc515524744)

[Figure 27．利用分支创建一个新的文件 19](#_Toc515524745)

[Figure 28．Feature分支create pull request 19](#_Toc515524746)

[Figure 29 Merge pull request 20](#_Toc515524747)

[Figure 30. create pull request的两种情况 21](#_Toc515524748)

[Figure 31．对blctrl/test项目进行Fork 21](#_Toc515524749)

[Figure 32．修改fork下来的仓库副本并提交pull request 22](#_Toc515524750)

[Figure 33．仓库拥有者收到Pull request 22](#_Toc515524751)

[Figure 34．本地工作目录、Git本地仓库与远程仓库 23](#_Toc515524752)

[Figure 35．克隆已存在的仓库到本地 25](#_Toc515524753)

[Figure 36．将本地改动推送至远程仓库的命令 26](#_Toc515524754)

[Figure 37．一般文件夹内容的推送流程 28](#_Toc515524755)

[Figure 38．推送改动至远程仓库时错误提示远程仓库已被改动 29](#_Toc515524756)

# 摘要

版本控制系统可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。

Git是一种分布式版本控制系统。在本地安装Git后通过命令行建立仓库，只利用本地仓库即可实现对本地项目/文件的版本控制。通过添加远程仓库可以实现与外界的工作交互。

GitHub 是最大的 Git 仓库托管平台，GitHub 托管着很多开源项目，其中不乏知名开源项目Ruby on Rails、jQuery、python 等。只利用GitHub也可实现对项目/文件的版本控制、问题追踪、代码审查等。此外，GitHub提供的包括fork在内的强大功能，可以实现对他人的项目做贡献或进行团队开发等。

Git与GitHub联动使用，可以在无网络时将改动更新至本地Git仓库，在有网络时集中将本地更新提交至GitHub远程仓库。

本文档详细介绍了Git的使用方法，GitHub的使用方法，在此基础之上介绍了Git与GitHub的联动使用方法。

关键词：Git、GitHub、仓库、版本库、版本控制

# 1. Git简介

版本控制是一种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系统。版本控制系统一般可分为：本地版本控制系统、集中式版本控制系统以及分布式版本控制系统。

本地版本控制系统，大多都是采用某种简单的数据库来记录文件的历次更新差异。

集中式版本控制系统（ Centralized Version Control Systems，简称 CVCS ），诸如 CVS，Subversion 以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。

分布式版本控制系统不需要集中式的仓库，每个人都工作在通过克隆建立的本地仓库中。每个人都拥有一个完整的仓库，查看提交日记、提交、创建里程碑和分支、合并分支、回退等所有操作都直接在本地完成而不需要网络连接。每个人都是本地仓库的主人，不再有谁能提交谁不能提交的限制，加上多样的协同工作模型（仓库间推送、拉回，以及补丁文件传送等），分布式版本系统让开源项目的参与度有爆发式增长。

Git是一种分布式版本控制系统。Git 和其他版本控制系统的主要差别在于，Git 只关心文件数据的整体是否发生变化，而大多数其他系统则只关心文件内容的具体差异。Git像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。为提高性能，若文件没有变化，Git 不会再次保存，而只对上次保存的快照作一链接。

Git 在本地磁盘上就保存着所有当前项目的历史更新，因此，在Git中的绝大多数操作都只需要访问本地文件和资源，不用连网，且处理速度极快。Git与Subversion（集中式版本控制系统）的区别如Figure 1所示。

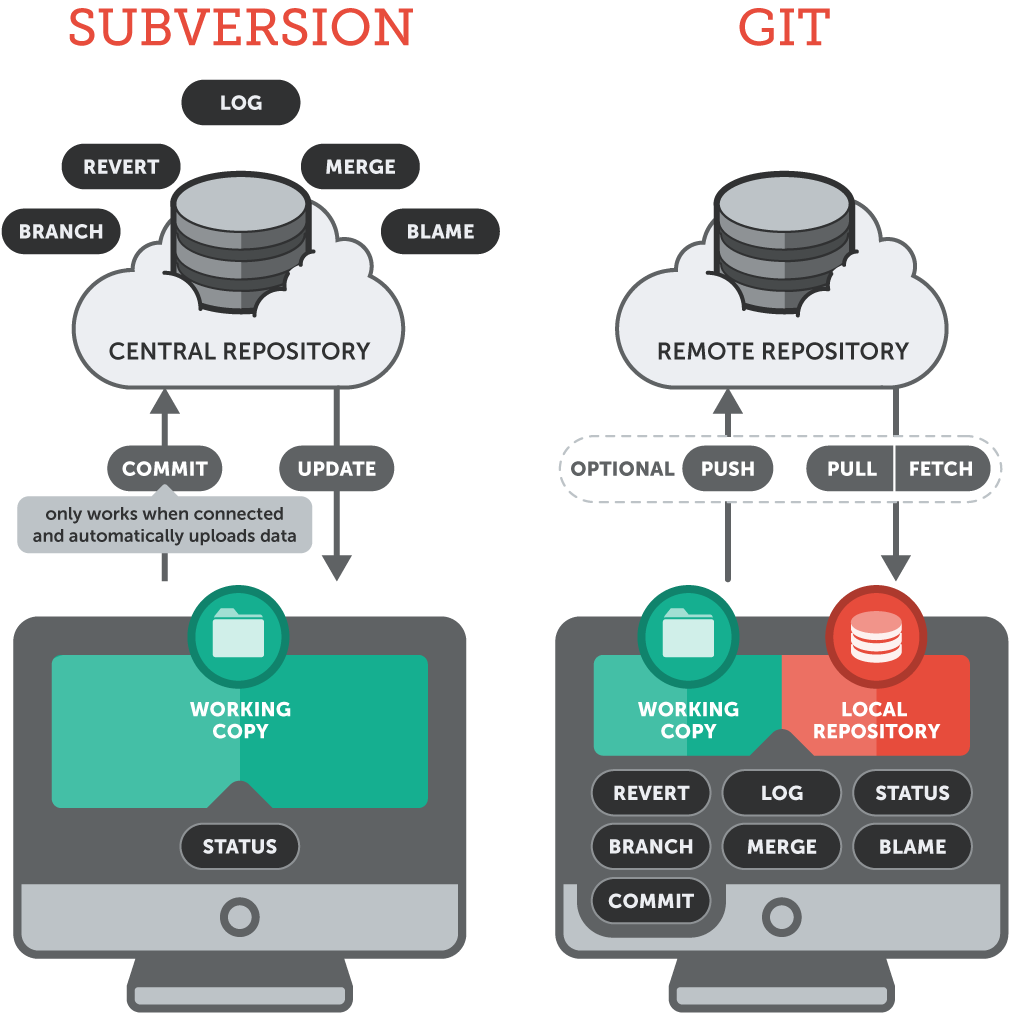


Figure 1．SVN VS Git

Git 使用 SHA-1 算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个 SHA-1 哈希值，作为指纹字符串。该字串由 40 个十六进制字符（0-9 及 a-f）组成（例如，24b9da6552252987aa493b52f8696cd6d3b00373）。Git 的工作完全依赖于这类指纹字串，所以会经常看到这样的哈希值。实际上，所有保存在 Git 数据库中的东西都是用此哈希值来作索引的，而不是靠文件名。

对于任何一个文件，在 Git 内都只有三种状态：已提交（committed），已修改（modified）和已暂存（staged）。已提交表示该文件已经被安全地保存在本地数据库中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。由此Git 管理项目时，文件流转的三个工作区域：Git 的工作目录，暂存区域，以及本地仓库，如Figure 2所示。

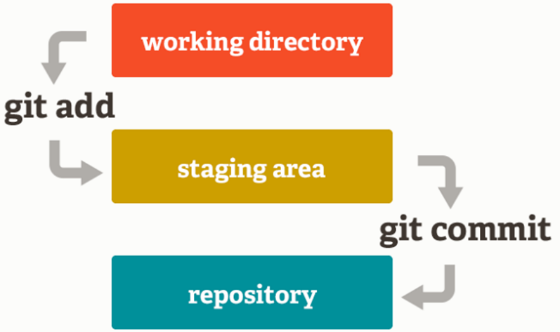


Figure 2．Git项目的三个工作区域

# 2.Git安装及设置

本章节主要介绍两种系统上的安装：Centos上通过编译源代码安装，以及Windows上的安装。

## 2.1 Centos上安装Git

安装之前，输入$git –version用于检测是否已安装Git，如出现Git版本，则说明已经安装完毕，否则按照以下流程进行安装：

1. <https://git-scm.com/download/linux>下载源代码（例如，git-2.17.0.tar.gz）
2. 建立安装目录 $ mkdir –p /BLC/epics/Gitsrc
3. 将安装包复制到 /BLC/epics/Gitsrc
4. 解压文件 $ tar –zxvf git-2.17.0.tar.gz
5. 配置安装目录并编辑和安装（超级用户）

# ./configure --prefix=/usr/local/git-2.17.0 && make install

1. 将编译好的Git添加到全局变量

# echo 'export PATH=$PATH:/usr/local/git-2.17.0/bin' > /etc/profile.d/git.sh

1. 使bashrc立即生效，全局变量立即生效

# source /etc/bashrc

1. 查看Git版本，输出版本号则证明安装成功

$ git --version

## 2.2 在Windows上安装Git

在Windows上安装Git有多种方法，下面仅介绍其中较为简单方便的一种。

（1）打开Git官方网站<https://gitforwindows.org/>，下载YsysgitGit（这是一个名为 Git for Windows的项目，也叫做 msysGit，和 Git 是分别独立的项目，可认为msysgit是Windows版的Git），下载结果如Figure 3所示：

C:\Users\123\Pictures\捕获.PNG

Figure 3．Git for Windows

（2）进行默认安装即可。安装完毕后，在开始菜单里面找到"Git --> Git Bash"，如所示，则说明安装成功。



Figure 4． Windows上Git安装成果的标志

## 2.3 初次运行Git前的设置

（1）用户信息

当安装完 Git 应该做的第一件事就是设置你的用户名称与邮件地址。 这样做很重要，因为每一个 Git 的提交都会使用这些信息，并且它会写入到你的每一次提交中，不可更改：

$ git config --global user.name “John Doe”

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

…

设置结束后，可查阅这些信息：

$ git config --list

user.name=John Doe

user.email=johndoe@example.com

…

再次强调，如果使用了 --global 选项，那么该命令只需要运行一次，因为之后无论你在该系统上做任何事情， Git 都会使用那些信息。 当你想针对特定项目使用不同的用户名称与邮件地址时，可以在那个项目目录下运行没有 --global 选项的命令来配置。

git config <key>： 来检查 Git 的某一项配置..

很多 GUI 工具都会在第一次运行时帮助你配置这些信息。

# 3. Git仓库基本操作

仓库又名版本库，英文名repository，可以简单的理解为一个目录，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，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻还可以将文件“还原”。

## 3.1 获取Git仓库

有两种取得 Git 项目仓库的方法：第一种是从一个服务器克隆一个现有的 Git 仓库；第二种是在现有项目或目录下导入所有文件到 Git 中，通过初始化获得一个Git仓库。

## 3.1.1克隆现有的仓库

从一个服务器或者GitHub上克隆现有的Git仓库，即可获得一份已经存在的Git仓库的拷贝，需要用命令git clone [url]实现。

需要说明的是，Git克隆的是该Git仓库上几乎所有的数据，而不是仅仅复制完成你的工作所需文件。当执行git clone 命令的时候，默认配置下远程 Git 仓库中的每一个文件的每一个版本都将被拉取下来。 事实上，如果某一个服务器的磁盘坏掉了，通常可以使用任何一个克隆下来的用户端来重建服务器上的仓库（虽然可能会丢失某些服务器端的挂钩设置，但是所有版本的数据仍在）。

比如，要克隆 GitHub上的 的可链接库 libgit2，可以用下面的命令：

$ git clone https://github.com/libgit2/libgit2 mylibgit

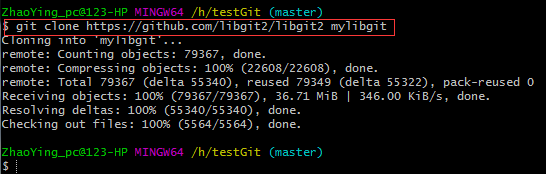


Figure 5．通过git clone命令克隆一个现存GitHub仓库

这会在当前目录下创建一个名为 “mylibgit”（自定义的本地仓库的名字） 的目录，并在这个目录下初始化一个 .git 文件夹，如下图所示。从远程仓库拉取下所有数据放入 .git 文件夹，然后从中读取最新版本的文件的拷贝。 如果进入到这个新建的 mylibgit 文件夹，会发现所有的项目文件已经在里面了，准备就绪等待后续的开发和使用。

Git 支持多种数据传输协议。上面的例子使用的是 https:// 协议，也可以使用 git:// 协议或者使用 SSH 传输协议，比如 user@server:path/to/repo.git 。

### 3.1.2在工作目录中初始化新仓库

（1）进入已有目录，输入命令$git init把这个项目管理起来，与此同时相应的目录下会多了一个.git的目录。例如，计划对“H:\testGit”进行项目管理， 可在进入该目录后，单击右键打开“Git bash here”打开命令窗口，输入“git init”指令，结果如Figure 6所示。

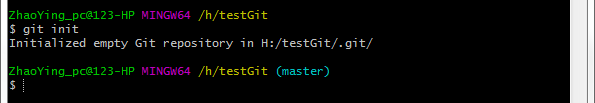


Figure 6．初始化新仓库

此时“testGit”目录下会多了一个.git子目录，这个子目录含有你初始化的 Git 仓库中所有的必须文件，这些文件是 Git 仓库的骨干。 但是，在这个时候，仅仅是做了一个初始化的操作，项目里的文件还没有被跟踪。

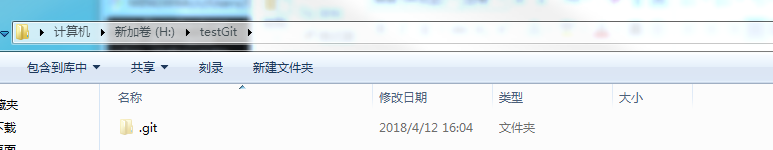


Figure 7．初始化成仓库后文件夹状态

## 3.2 记录每次更新到仓库

如果当前目录下有几个文件想要纳入版本控制，需要先用 $git add 命令告诉 Git 开始对这些文件进行跟踪，然后执行$git commit提交。

举例如下：

在testGit下有一个名为“test1.txt”的记事本文件（内容为一个数字“1”）。

第一步：使用命令git add test1.txt添加到暂存区里面去，如下：

C:\Users\123\AppData\Roaming\Tencent\Users\724095886\QQ\WinTemp\RichOle\V~353QEY0}[F6EYPDLAIEVF.png

Figure 8．git add命令使用举例

如果没有任何提示，说明已经添加成功。

第二步：使用命令git commit –m“注释”将文件提交到仓库，如下：

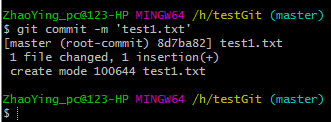


Figure 9．git commit 命令使用举例

现在已经提交了一个“test1.txt”文件，可以通过git status来查看是否还有文件未提交，如下：

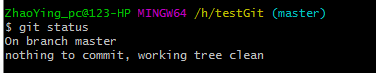


Figure 10．git status命令使用举例

说明没有任何文件提交。

继续修改“test1.txt”内容，比如在新的一行添加一个数字“2”，继续使用git status来查看结果，如下：

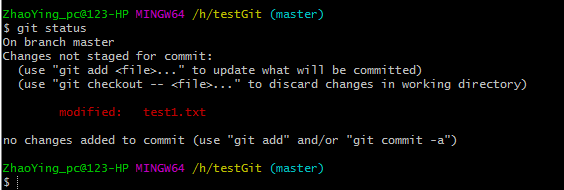


Figure 11．修改已追踪文件的提示

上面的结果显示“test1.txt”已经被修改，但是修改未被提交。

“test1.txt”具体的修改内容，可用过命令git diff查看，如下：



Figure 12．git diff使用方法

对修改后的文件提交至仓库，提交修改和提交文件是一样步骤（第一步git add，第二部git commit）如下：

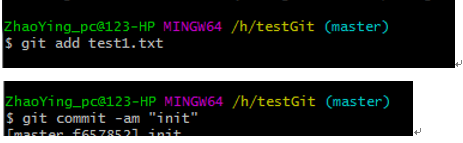


Figure 13．再次修改被追踪文件内容并提交

继续改动test1.txt内容，并提交至仓库。所有的操作完成后，利用git log看下历史记录，结果如下：

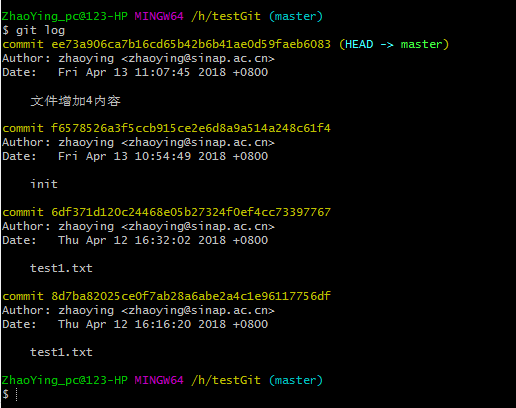


Figure 14．git log查看提交历史

可更简洁的显示提交历史git log --pretty=oneline ：

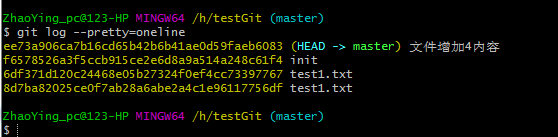


Figure 15．git log 其他使用方法

## 3.3远程仓库的管理和使用

要参与任何一个 Git 项目的协作，必须要了解该如何管理远程仓库。远程仓库是指托管在其他位置或网络上的项目仓库，可能会有好多个，其中有些只能读，另外有些可以写。同他人协作开发某个项目时，需要管理这些远程仓库，以便推送或拉取数据，分享各自的工作进展。 管理远程仓库的工作，包括添加远程库，移除废弃的远程库，管理各式远程库分支，定义是否跟踪这些分支，等等。以下介绍论远程库的管理和使用。

1. 添加远程仓库

要添加一个新的远程仓库，可以指定一个简单的名字，以便将来引用，运行 $git remote add [shortname] [url]。

例如，添加一个仓库 $ git remote add pb https://github.com/paulboone/ticgit ，以后就可以用字符串pb指代对应的仓库地址了。比如，要抓取所有 Paul 有的，但本地仓库没有的信息，可以运行 $git fetch pb。

1. 从远程仓库抓取数据并更新至本地

如上所示 ，可以利用如下命令从远程仓库抓取数据到本地：$ git fetch [remote-name]，此处[remote-name]与上述添加远程仓库时[shortname]相同，具体用法为：$git fetch [remote-name]将远程仓库所有更新抓取到本地（如， $git fetch pb，将pb指代的仓库的所有分支的所有更新抓取到本地）；$git fetch [remote-name] [branch-name]将远程仓库特定branch的更新抓取到本地（如，$git fetch origin master，将origin指代的仓库中branch分支的更新抓取到本地）。

如果是克隆获得了一个仓库，此命令会自动将远程仓库归于 origin 名下。抓取从你上次克隆以来别人上传到此远程仓库中的所有更新（或是上次 fetch 以来别人提交的更新），使用命令 ：$git fetch origin 。

fetch 命令只是将远端的数据拉到本地仓库，并不自动合并到当前工作分支，只有确实准备好了，才能手工合并（合并命令为$git merge）。$git pull同样可以抓取远程仓库更新到本地，但它本身带有合并功能，即git pull = git fetch +git merge。

在不考虑分支时，一般用法：

$ git fetch [remote-name]

$ git pull –rebase

1. 提交本地更新到远程仓库

$git push [remote-name] master（其中master是一般默认的分支名）。

综上所述，Git的基本操作包括：获取Git仓库、记录每次更新到仓库、添加远程仓库、抓取远程仓库更新并合并至本地、提交本地更新到远程仓库等。Figure 16为涵盖上述Git操作的例子。Figure 17为上述Git基本操作之间的关系。

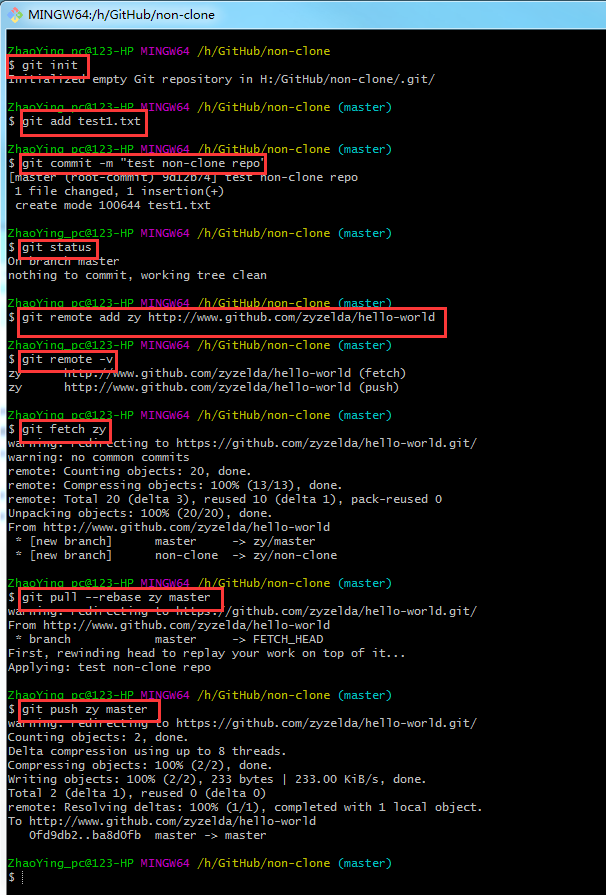


Figure 16．Git基本操作实例

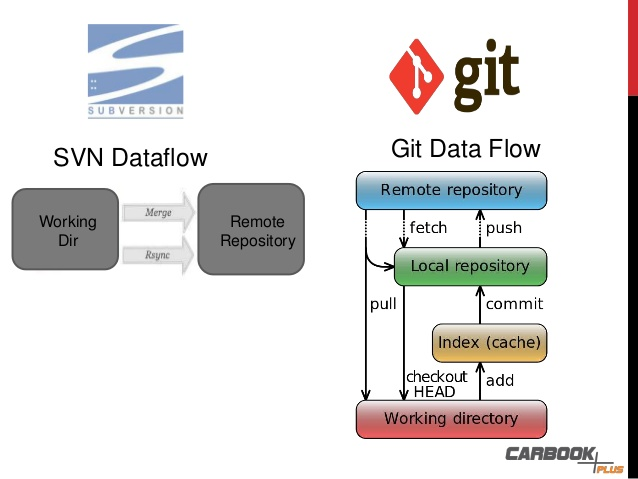


Figure 17．图解Git基本操作之间的关系

# 4. GitHub简介

GitHub 是最大的 Git 仓库托管商，是成千上万的开发者和项目能够合作进行的中心。 大部分 Git 仓库都托管在 GitHub，很多开源项目使用 GitHub 实现 Git 托管、问题追踪、代码审查以及其它事情。

以下将介绍如何高效地使用 GitHub，包括如何注册和管理账户、创建和使用 Git 仓库、向已有项目贡献的通用流程以及如何接受别人向你自己项目的贡献、GitHub 的编程接口和很多能够让这些操作更简单的小提示。

## 4.1 GitHub用户申请及设置

打开GitHub官网[https://github.com，看到界面如](https://github.com，看到界面如Figure1)Figure 18所示。

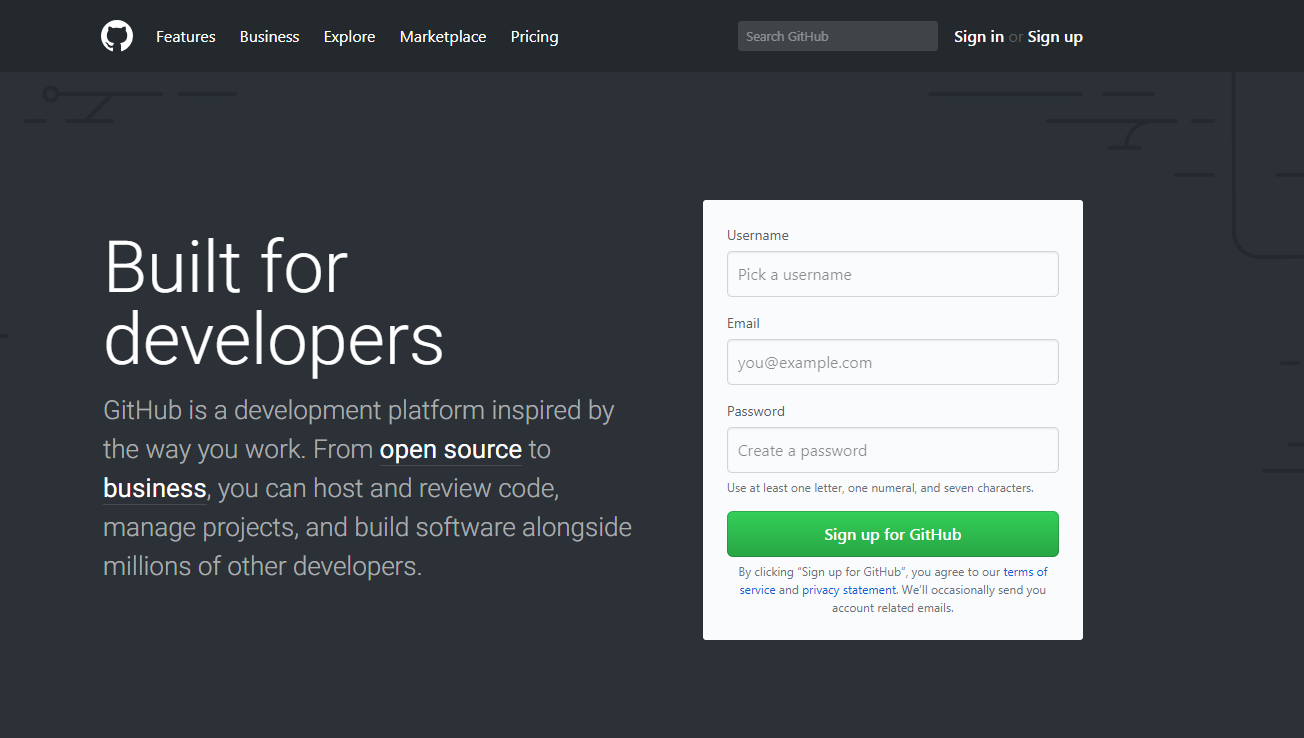


Figure 18．GitHub官网页面

选择一个未被占用的用户名，提供一个电子邮箱和密码，点击“Sign up for Git”，进入“Welcome to GitHub”页面。

“Welcome to GitHub”页面，personal plan为默认选项即可，点击“Continue”。

“Tailor your experience”可根据自身情况填写或直接忽略，点击“Continue”，至此就创建一个免费的GitHub账户。

随后GitHub 会给提供的邮件地址发送一封验证邮件，需尽快到该邮箱进行验证。

如需设置SSH访问、头像、邮件地址、安全设置等可点击窗口右上角的Settings链接进行账户设置，见Figure 19。

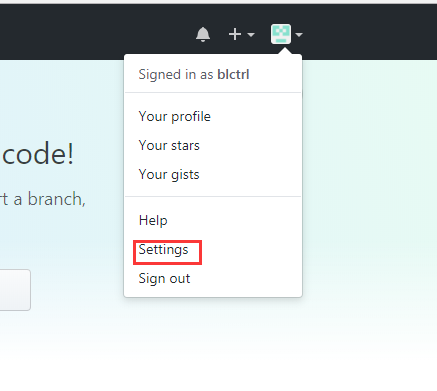


Figure 19．GitHub设置

## 4.2 创建GitHub仓库

通过创建仓库可以实现项目的共享。

（1）如Figure 20所示，通过点击“start a project ”，或者窗口右侧的“New repository”，或者顶部工具条用户名旁边的 + 按钮，进入Create a new repository页面。

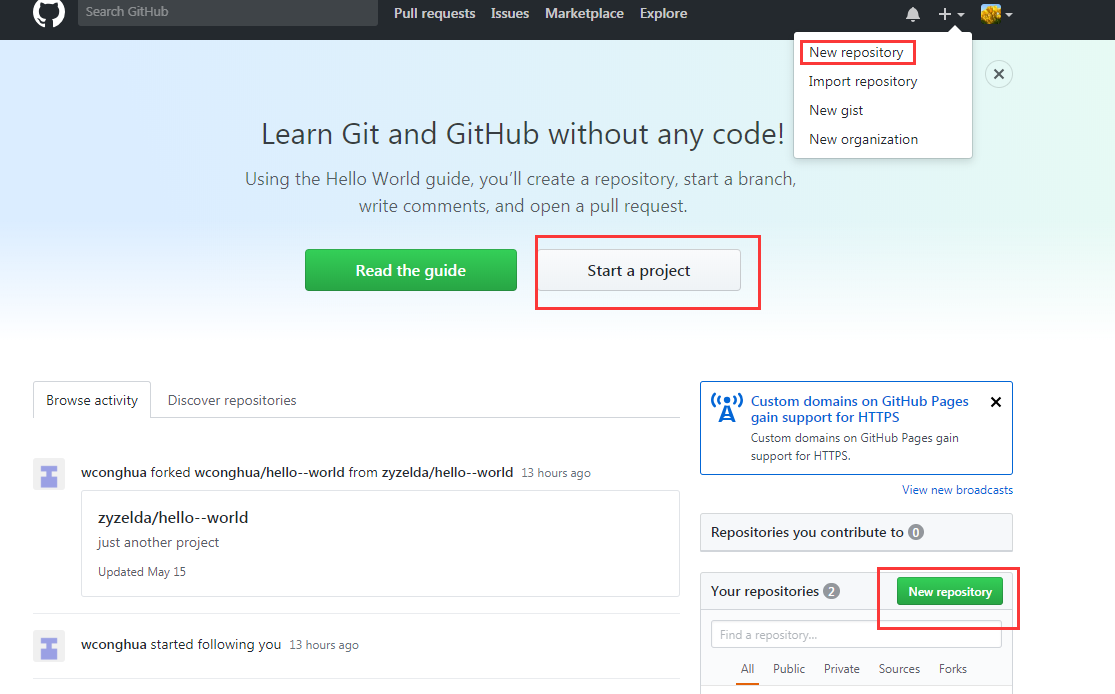


Figure 20．创建Git仓库的方式

（2）“new repository”表单如Figure 21所示，填写仓库名称，根据需要添加仓库的描述，（建议勾选“**Initialize this repository with a README**”），点击“Create repository”，至此在Github上拥有了一个以**<owner>/<repository\_name>**命名的仓库。

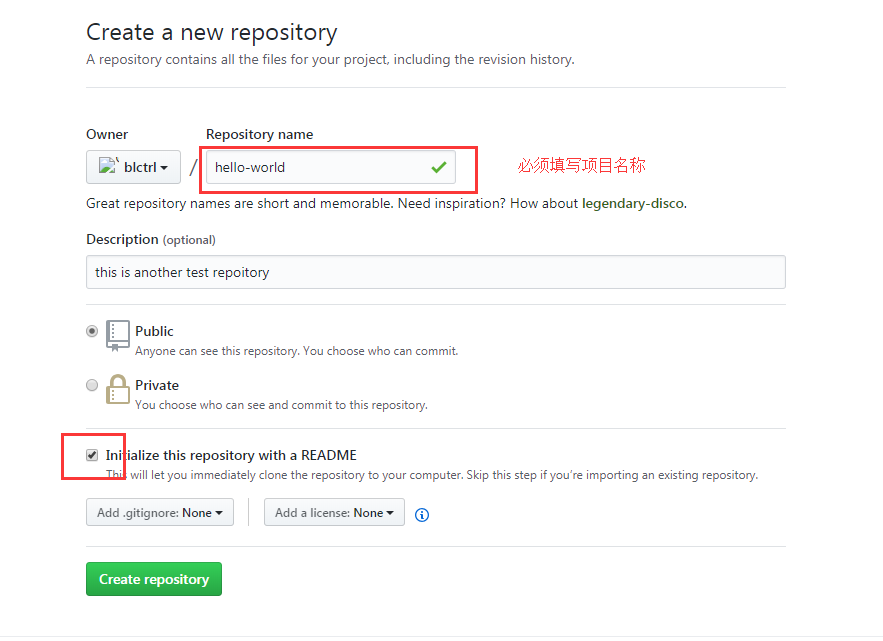


Figure 21. new repository表单

现在项目就托管在Github上了，可以吧URL给任何人。Github上的项目可以通过HTTP或SSH访问，格式分别是：

https://github.com/<owner>/<repository\_name>

SSH ：git@github.com:<owner>/<repository\_name>

## 4.3使用仓库

点击新建的仓库，看到如Figure 22所示界面，可看出对仓库code的操作包括：创建新的文件、上传文件、寻找文件、克隆或下载仓库、创建分支、建立新的pull请求、编辑版库的说明等，以下进行简单说明和介绍。

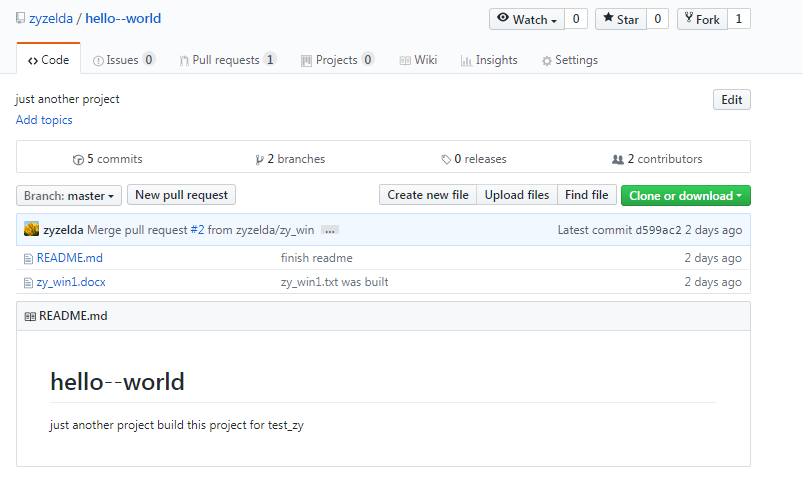


Figure 22．GitHub上一个名为“hello-world”仓库的主页面

（1）创建新文件

点击仓库主页面上的“Create new file”，进入如Figure 23所示界面，根据需要编辑文档，对commit的内容进行描述，然后点击“Commit new file”即完成了新文档的创建及上传。

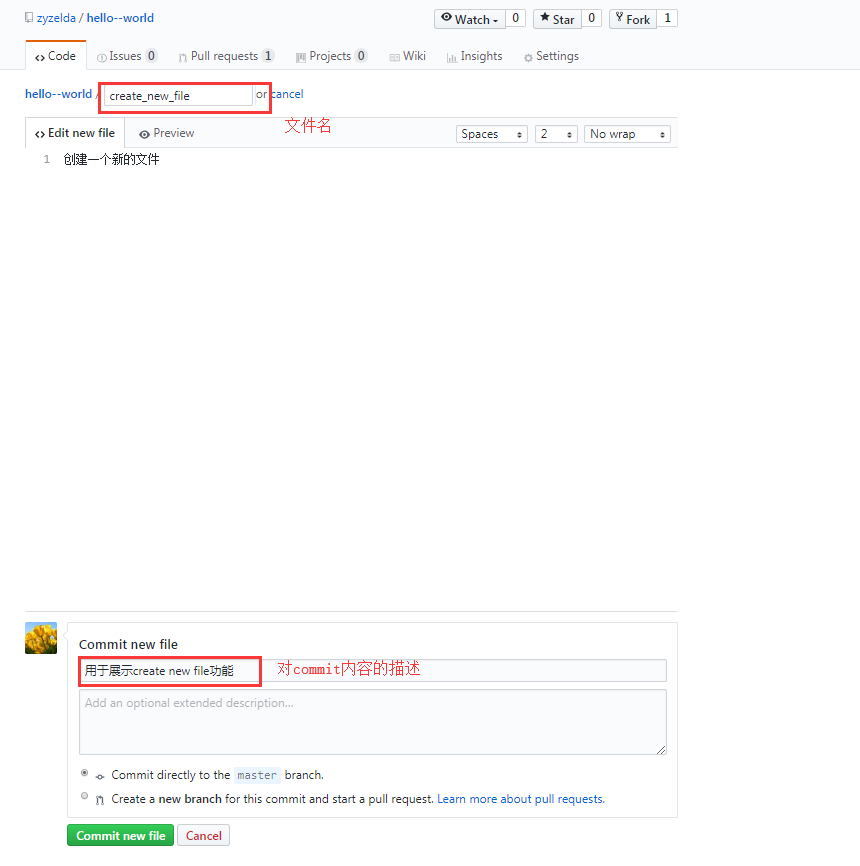


Figure 23. Create new file

（2）上传文件

点击仓库主页面上的“Upload files”，进入Figure 24所示界面。根据需要选择上传文件，对commit的内容进行描述，点击“Commit changes”完成文件的上传。

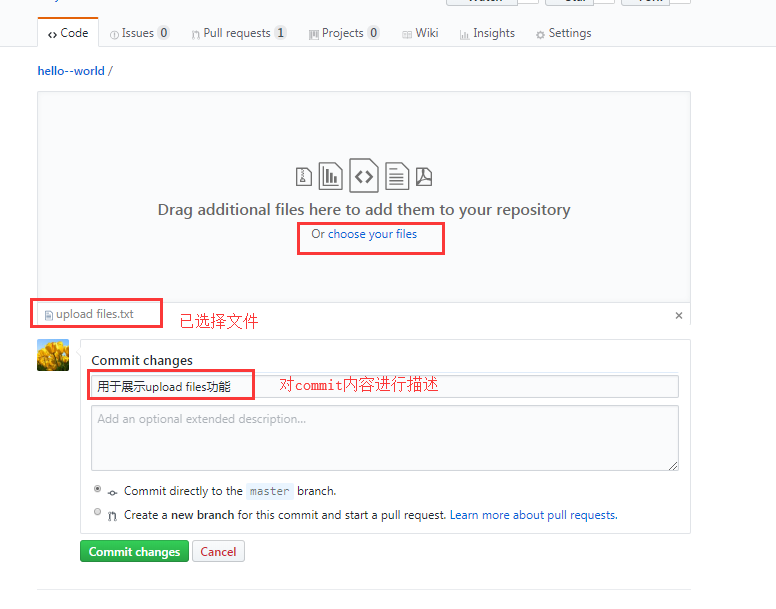


Figure 24．Upload files

（3）创建分支、Merge pull request

几乎所有的版本控制系统都以某种形式支持分支。 使用分支意味着你可以把你的工作从开发主线上分离开来，以免影响开发主线。与许多其它版本控制系统不同，Git 鼓励在工作流程中频繁地使用分支与合并，哪怕一天之内进行许多次。

仓库默认的分支为master，创建新的分支意味着对当前master内容的拷贝。在新的分支上工作过程中，如master的内容发生改变，可以随时同步这些更新。Figure 25展示了master分支和一个名为feature分支的工作关系。



Figure 25．分支工作原理示意图

点击Figure 26中的Branch：master命名一个新的分支（例如，创建一个名为“feature”的分支），点击Create branch feature from ‘master’，进入feature分支。

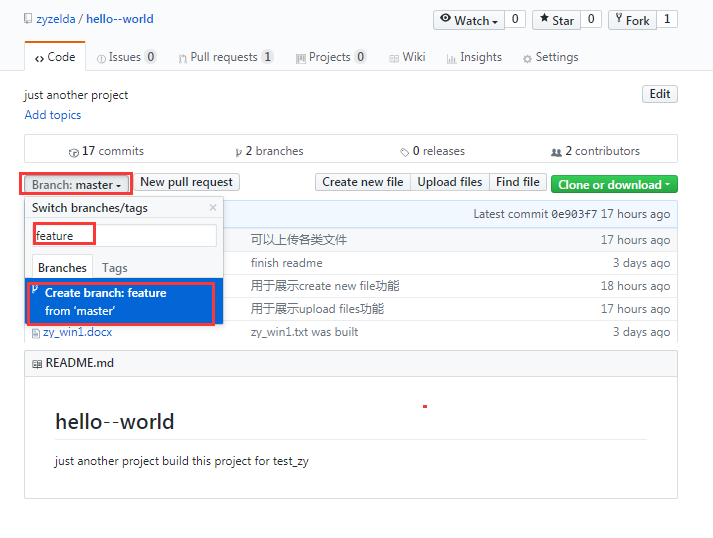


Figure 26．建立一个名为“feature”的分支

（4）Create pull request

进入feature分支，创建了一个名为“branch\_feature”的文件，进入如Figure 27所示界面。如确定位于feature分支上的工作已完成，点击图中的Compare & pull request。

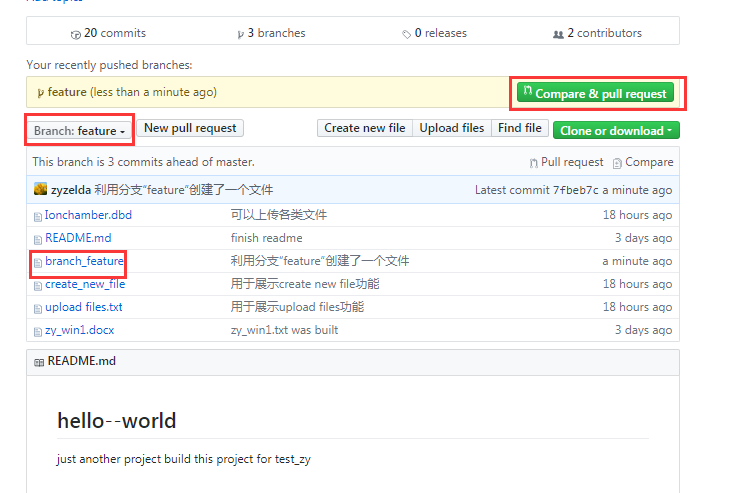


Figure 27．利用分支创建一个新的文件

给pull request一个注释，写一个简要的描述，而后点击Create pull request。

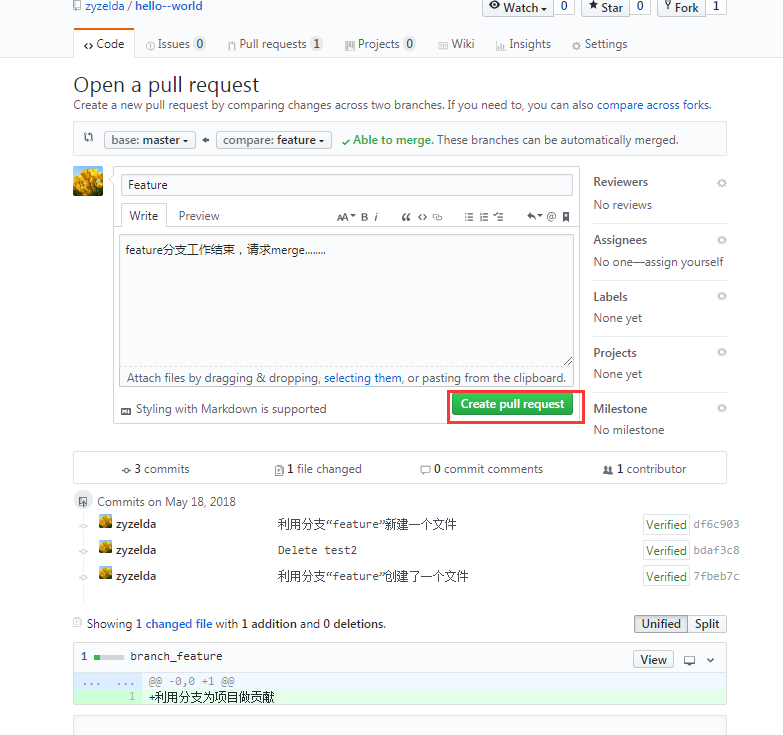


Figure 28．Feature分支create pull request

（5）Merge pull request

确认feature分支的工作后，点击Merge pull request，接下来Confirm merge，同时可对分支的工作加以评论。

操作完成后，feature分支上的工作已被合并至master上，因此可点击Delete branch直接删除分支。

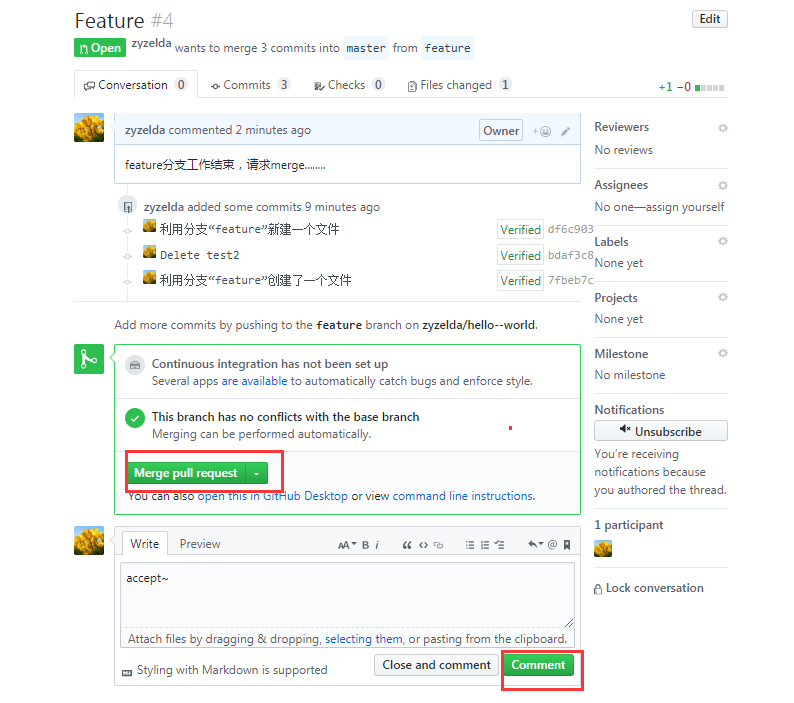


Figure 29 Merge pull request

（6）派生（Fork）

发出pull request其目的是对项目/仓库做贡献，除了应用在分支上，还有一种是应用在fork上，如Figure 30所示。

如果想要参与某个项目，但是并没有推送权限，这时可以对这个项目进行“派生”。 派生的意思是指，GitHub 将在你的空间中创建一个完全属于你的项目副本，且你对其具有推送权限。将修改推送到派生出的项目副本中，并通过创建合并请求（Pull Request）来让他们的改动进入源仓库。

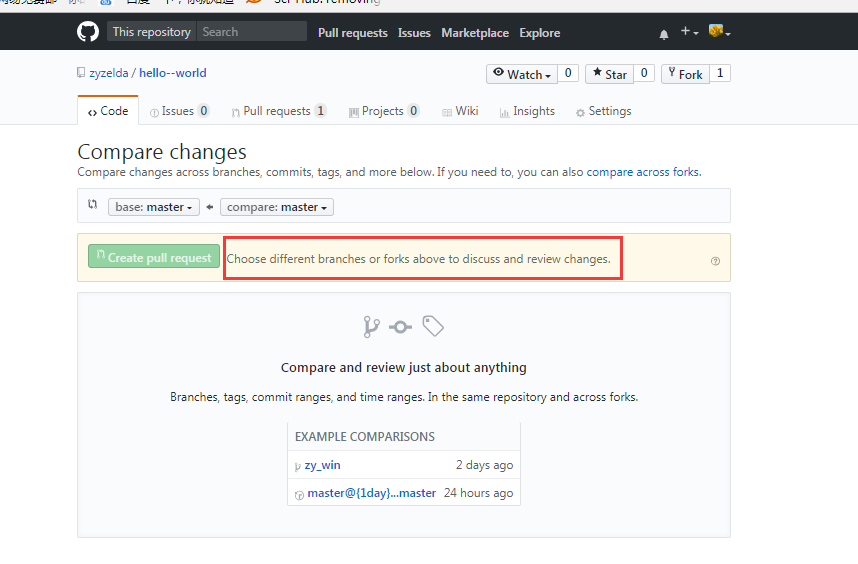


Figure 30. create pull request的两种情况

例如,通过HTTP地址，找到为blctrl（用户名）下的仓库test，如Figure 31所示。点击Fork，则该仓库副本即派生到当前登录的用户下，在当前用户（zyzelda）下多了两个内容相同的仓库blctrl/test及zyzelda/test，fork结束后默认跳到zyzelda/test。

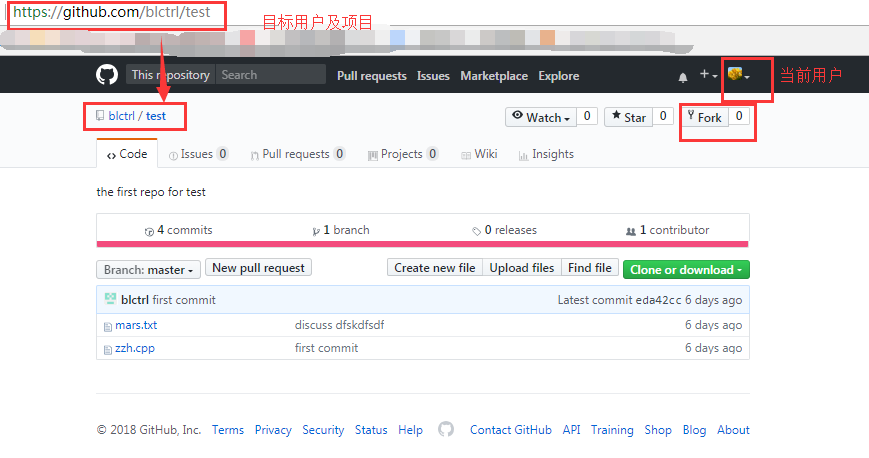


Figure 31．对blctrl/test项目进行Fork

在当前用户下，对派生出的仓库进行修改，并提交New pull request，继续点击Create pull request。至此，zyzelda针对修改内容已提交至仓库拥有者（blctrl）那里。

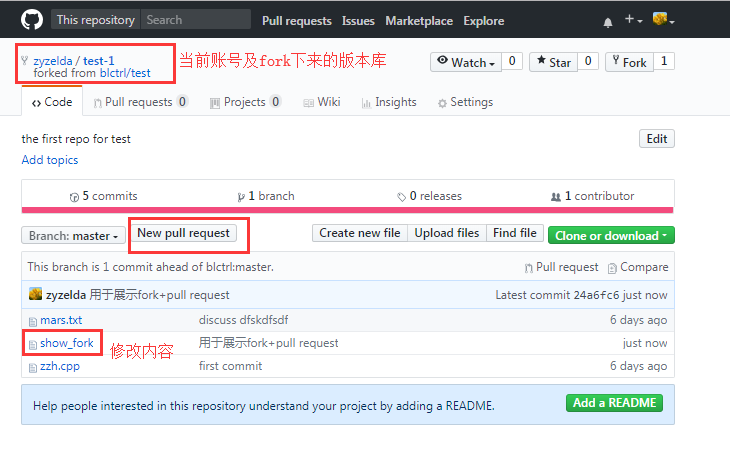


Figure 32．修改fork下来的仓库副本并提交pull request

用户blctrl在仓库blctrl/test的主页面Pull request处，可看到pull request数量，如Figure 33所示。

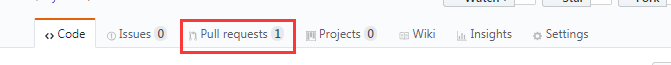


Figure 33．仓库拥有者收到Pull request

点击Pull request，进入新的页面并点击推送内容，如确认合并，则点击Merge pull request，并点击Confirm merge。至此，zyzelda对项目做出的改变被合并至blctrl/test仓库中。

# 5. Git与GitHub的联动使用

如上所述，只利用Git在本地建立Git仓库，即可实现对本地项目/文件的版本控制。如需和外界进行工作交互，添加远程版本库即可。Figure 34展示了本地工作目录，本地Git仓库及远程仓库的关系，要实现版本控制，红色区域的本地仓库是必须的，而位于灰色云端的远程仓库，则根据需要选择添加与否。

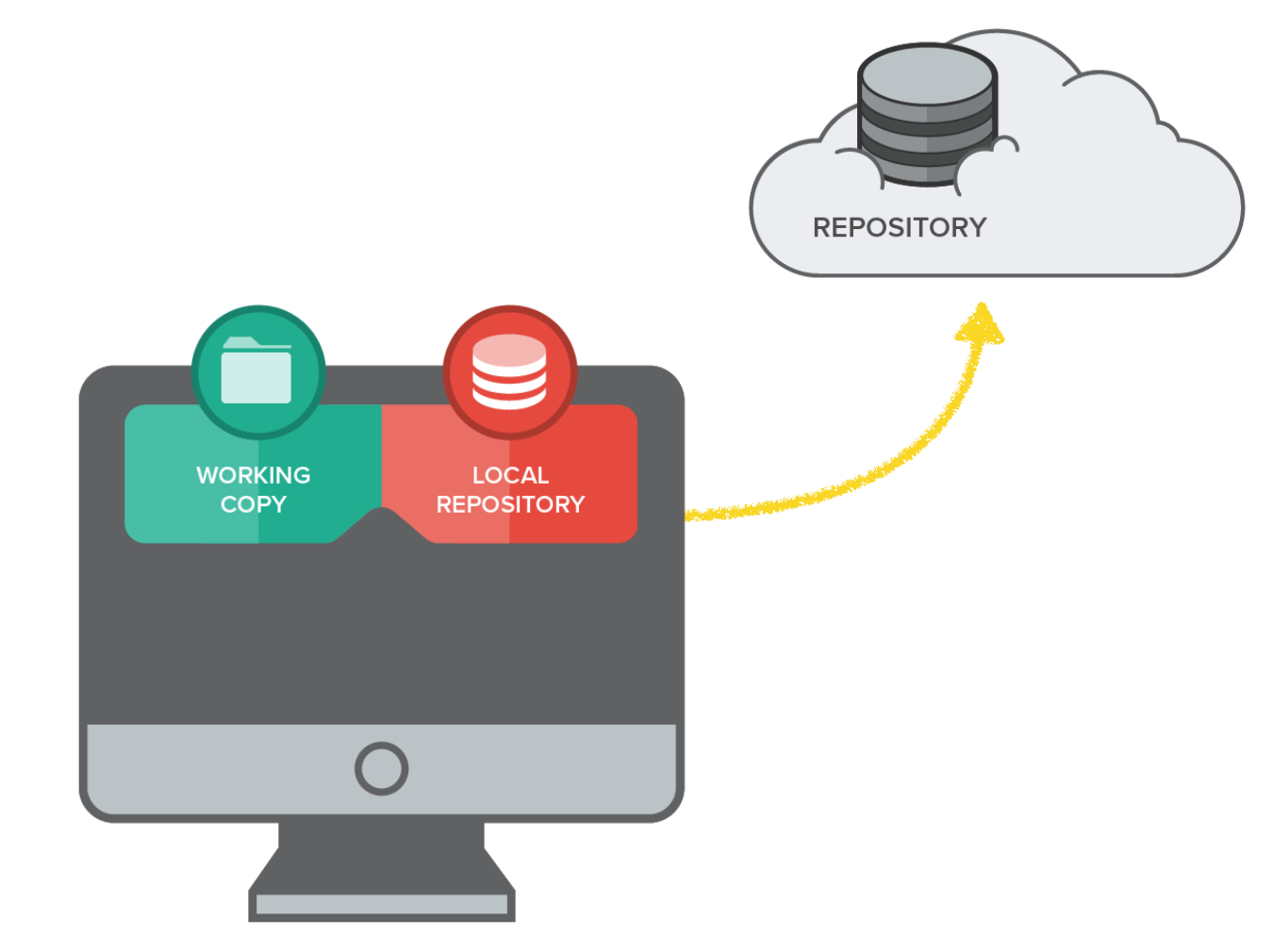


Figure 34．本地工作目录、Git本地仓库与远程仓库

同样的，只利用GitHub也可实现对项目/文件的版本控制。如需要对他人的项目做贡献或团队开发，可借助GitHub提供的包括fork在内的强大功能实现。

Git与GitHub联动使用，可以在无网络时将改动更新至本地Git仓库，在有网络时集中将本地更新提交至GitHub远程仓库。Figure 35展示了Git和GitHub联动使用时的关系，此时GitHub存在为远程仓库。

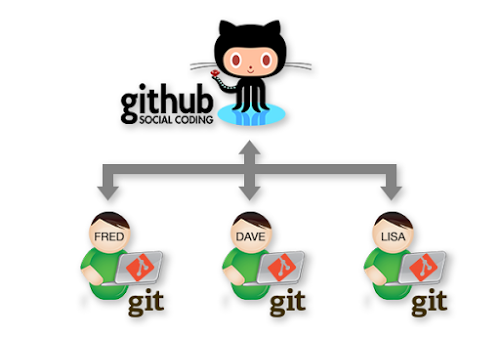


Figure 35．Git&GitHub联动使用

Git与GitHub联动使用需要的前提条件为：已安装Git，已拥有GitHub账号。以下将介绍具体的步骤。

## 5.1 克隆一个GitHub仓库并实现Git与GitHub联动使用

### 5.1.1克隆现有的仓库

如果想为某个开源项目贡献自己的一份力，用 $git clone 命令即可获得一份已经存在了的 GitHub仓库的拷贝。

比如，要克隆GitHub的可链接库 hello-world，可以用下面的命令

$ git clone https://github.com/zyzelda/hello-world。

这会在当前目录下创建一个名为 “hello-world” 的文件夹，这个文件夹下包含一个 .git 文件夹，并包含了所有的项目文件。该文件夹即为本地Git仓库，准备就绪等待后续的开发和使用。



Figure 36．克隆已存在的仓库到本地

### 5.1.2 记录每次更新到本地及远程仓库

目前本地存在一个真实项目的仓库。接下来，对本地仓库里的文件做些修改，并提交至本地仓库及远程仓库。

例如，在本地仓库hello-world中创建一个文本文档test2.txt，最终将该文档推送至GitHub上的远程仓库，其操作流程如Figure 37所示。

各行命令简单解释如下：

1. git add test2.txt

使用该命令将开始跟踪test2.txt，并文件暂存在本地磁盘。

1. git commit –m “创建test2.txt”

使用该命令将test2.txt提交至本地仓库，注释是对提交内容进行简单说明。提交后自动生成了很多提示信息，包括，test2.txt是在哪个分支（master分支）提交的，本次提交的完整SHA-1校验和（哈希值b33a1ba），以及在本次提交中，有多少文件修订过（1个）等。

每一次运行提交操作，都是对项目作一次快照，以后可以回到这个状态，或者进行比较。

1. git push origin master

克隆到本地的文件夹，用户名默认为是origin，使用git push origin master，将test2.txt提交至远程GitHub仓库。

输入git push origin master后会出现提示，按要求输入GitHub的用户民和密码，填入自己的账号和密码即可。

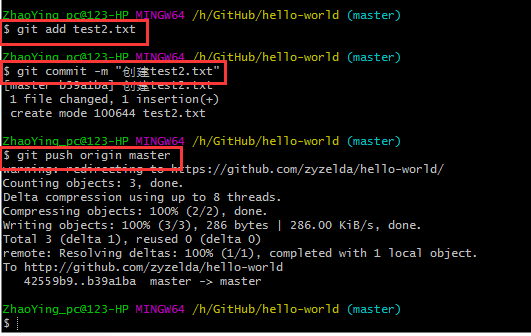


Figure 37．将本地改动推送至远程仓库的命令

## 5.2 本地Git仓库内容上传至GitHub仓库

如果需要将本地文件夹里的内容通过Git命令推送至GitHub仓库，操作与上述内容略有不同。

如本地存在一个名为non-clone的普通文件夹，现在希望将文件夹里的test1.txt上传至GitHub仓库（URL: <https://github.com/zyzelda/hello--world>），所需操作如Figure 38所示。

各行命令简单解释如下：

1. git init 仓库初始化，完成该命令后non-clone成为一个仓库
2. git add test1.txt 跟踪文件test1.txt
3. git commit –m “test non-clone repo” 提交文件test1.txt至non-clone仓库
4. git status 查看当前状态
5. git remote add zy http:www.github.com/zyzelda/hello--world

添加远程仓库，利用<shortname>（zy）代指整个URL(http:www.github.com/zyzelda/hello--world)，如果<shortname>处空缺，则默认的<shortname>为origin。

1. git remote –v

显示需要读（fetch）写（push）的远程仓库对应的URL(http:www.github.com/zyzelda/hello--world)，及使用Git 保存的简写名称（zy）。

1. git fetch zy 从远程仓库中获得最新数据。添加远程仓库后，远程仓库如果有变动，在提交本地修改时候，由于本地仓库与远程仓库内容不一致，直接推送改动到远程仓库是会报错的，如Figure 39中提示的那样。针对这个问题，要git fetch获取test仓库的内容，并且git pull同步获取内容到本地，然后再推送。
2. git pull –rebase <shortname> <branch> 将远程仓库数据同步至本地
3. git push <shortname> <branch> 将本地改动推送至远程仓库

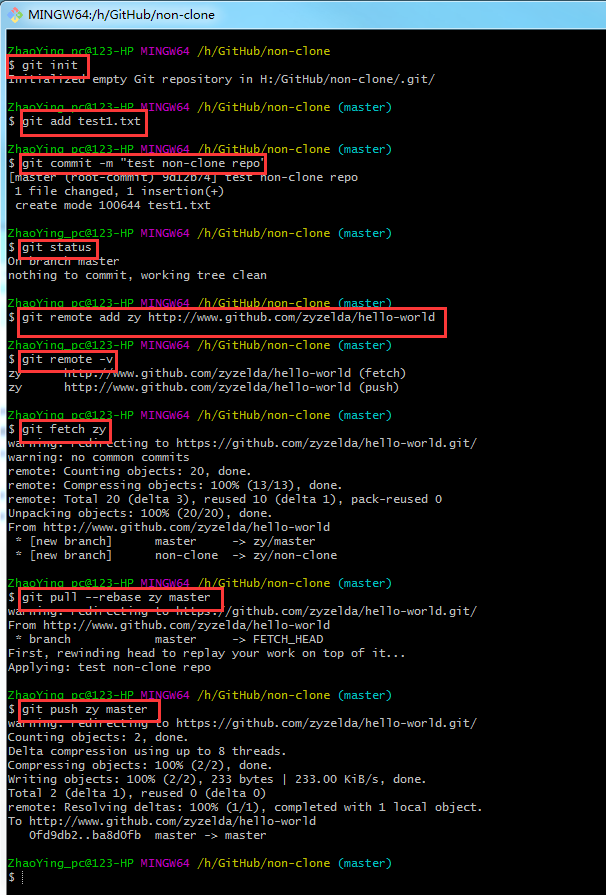


Figure 38．一般文件夹内容的推送流程

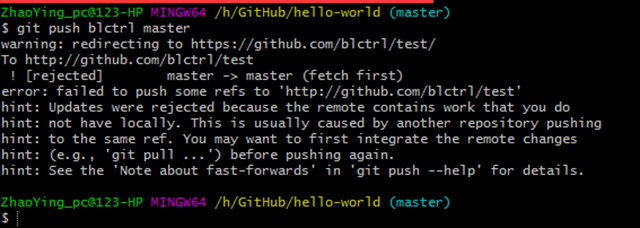


Figure 39．推送改动至远程仓库时错误提示远程仓库已被改动