Git User Guide v1.0

Git 的基本操作

**目 录**

[目 录](#_Toc26683)

[变更记录](#_Toc4379)

[1 引言](#_Toc24546)

[1.1 安装git (windows)](#_Toc31556)

[1.2 缩写及术语](#_Toc9768)

[1.3 参考文献](#_Toc12751)

[1.4 使用对象](#_Toc8086)

[2 初次运行git前的配置](#_Toc17151)

[2.1 用户信息配置](#_Toc30134)

[2.2 查询配置信息](#_Toc7466)

[3 GIT GUI](#_Toc25782)

[3.1 把本地项目上传至 Github](#_Toc7749)

[3.1.1 创建本地仓库](#_Toc9008)

[3.1.2 把本地仓库与远程库进行关联](#_Toc6425)

[3.1.3上传](#_Toc7443)

[3.2 从远程库获取更新](#_Toc17653)

[3.3 克隆github上的项目到本地](#_Toc17160)

[4 Git Bash (命令行)](#_Toc29719)

[4.1 创建与操作本地git库](#_Toc31172)

[4.2 克隆现有仓库](#_Toc3613)

[4.3本地与远程仓库的交互](#_Toc20774)

[5分支](#_Toc30524)

[5.1切换到本地工作区](#_Toc10520)

[5.2从远程仓库克隆代码到本地仓库](#_Toc4635)

[5.3 基于本地master分支，创建本地develop分支](#_Toc23168)

[5.4 本地开发后代码推送到远程dev分支](#_Toc14711)

[5.5发布到测试分支开展测试工作（管理人员操作）](#_Toc27855)

[5.6测试人员提交bug后，开发从远程拉取代码到本地](#_Toc14338)

[5.7测试通过后，把代码合并到dev分支](#_Toc25835)

[5.8 开发测试全部完成后，合并到master分支](#_Toc14360)

[6 Tag](#_Toc24713)

[6.1 创建tag](#_Toc12863)

[6.2 查找和列出tag](#_Toc2352)

[6.3发布后发现bug，基于tag创建一个本地分支](#_Toc22673)

[6.4修复后查询更改的内容](#_Toc1150)

[6.5修复完毕后，要合并到dev和master分支](#_Toc11900)

[6.6合并完后，打一个新的tag，推送到远程](#_Toc29539)

[6.7删除本地的bug 分支](#_Toc14205)

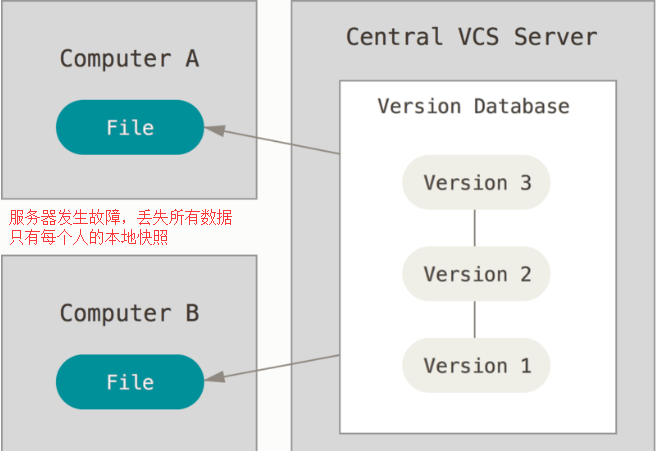
变更记录

| **版本号** | **变更内容** | **修订人** | **审阅人** | **修订日期** |
| --- | --- | --- | --- | --- |
| 1.0 | 初稿 | 郝林 |  | 2018.4.11 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

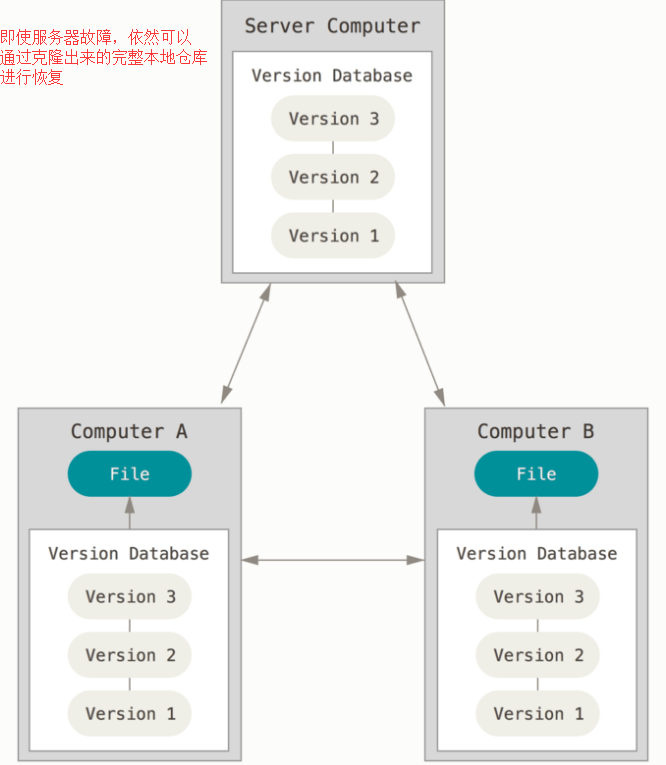
# 引言

**Git**是一个快速、可扩展的分布式版本控制系统，与Subversion、CVS 等集中式版本控制系统相比，它会把每次提交的文件的全部内容都会记录下来，而不是只存储每次提交(commit)之间的差异。

集中式版本控制系统



分布式版本控制系统



本文重点介绍如何通过 GIT GUI 和 GIT Bash（命令行）去实现项目的下载上传、更新；分支拉取、关联、合并和删除；以及Tag的创建

## 安装git (windows)

Download URL: [**https://www.git-scm.com/download/**](https://www.git-scm.com/download/)

可用的其他客户端：TortoiseGit、SourceTree 等

## 缩写及术语

| **术语** | **备注** |
| --- | --- |
| 已提交(committed) | 表示数据已经安全保存在本地仓库中，供更新到远程 |
| 已修改(modified) | 表示数据已修改，但未保存到本地仓库 |
| 已暂存(staged) | 表示已标记了修改的数据，下次提交的快照中将会包含它 |
| GIT GUI | Git 的图形化界面 |
| GIT BASH | Git 的命令行模式 |
|  |  |

## 参考文献

| **序号** | **文献名称或平台** | **版本号** |
| --- | --- | --- |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |

## 使用对象

§ 汇链丰研发人员、项目管理人员及文档人员

# 初次运行git前的配置

## 用户信息配置

首先要配置的是个人的github用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，它会随更新内容一起被永久纳入历史记录：

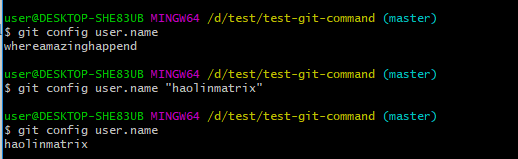
git config --global user.name “whereamazinghappend”

git config --global user.email [“271486662@qq.com”](mailto:“271486662@qq.com”)

但是有时候，我们需要针对特定文件或项目使用不同的用户名和电子邮件来提交，可以在**这个项目目录下**使用没有全局选项的命令重新配置即可

git config user.name “linhaomatrix”

git config user.email [“linhao@matrix.space”](mailto:“linhao@matrix.space”)



## 查询配置信息

git config user.name

git config user.email

# 3 GIT GUI

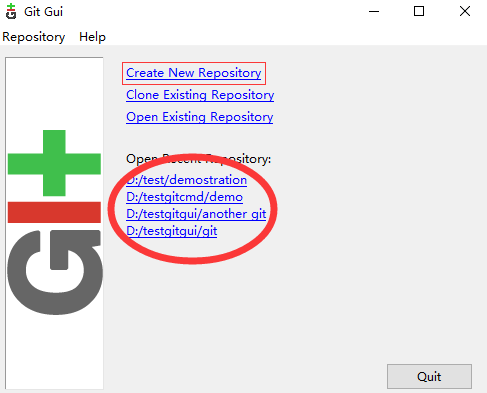
GIT GUI 是 GIT 的图形化界面，如果只是想简单地通过Git 进行版本控制，这个界面可以满足需要。通过这个界面可以方便地：

1. 把本地项目上传至 Github
2. 从远程库获取更新
3. 克隆github上的项目到本地

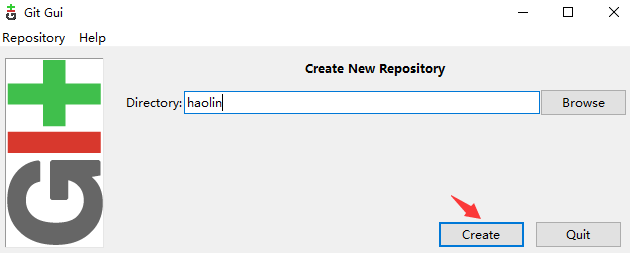
## 3.1 把本地项目上传至 Github

### 3.1.1 创建本地仓库

任选一个本地文件夹（位置随意），右键选择Git GUI Here ，会弹出以下框，选择create new Repository（红色圈内历史创建的记录）。

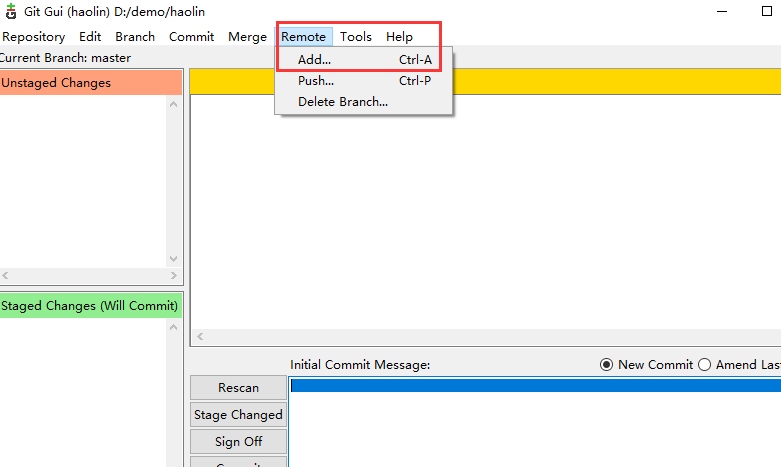


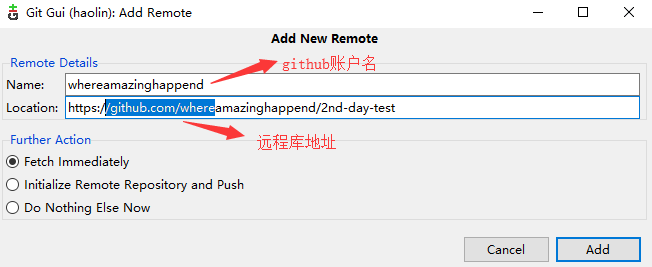
在弹出的输入框中，输入一个文件夹名称。这个文件夹就是我们的项目仓库。只要将需要上传的项目放在文件夹中即可。



### 3.1.2 把本地仓库与远程库进行关联

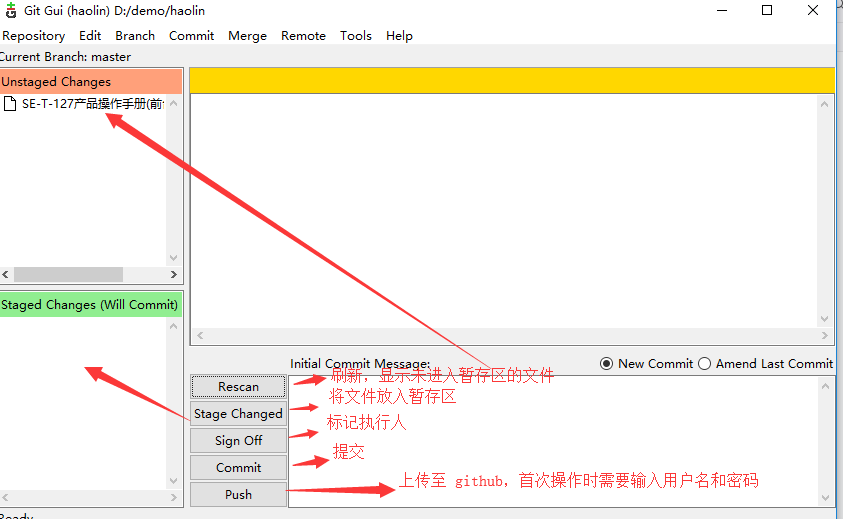
创建好本地仓库以后，需要将其关联到要上传项目的远程库。在上一步完成创建后弹出的可视化窗口中，选择 Remote -> Add...，然后输入 github 的账户名和目标地址，点击 Add，提示添加成功。



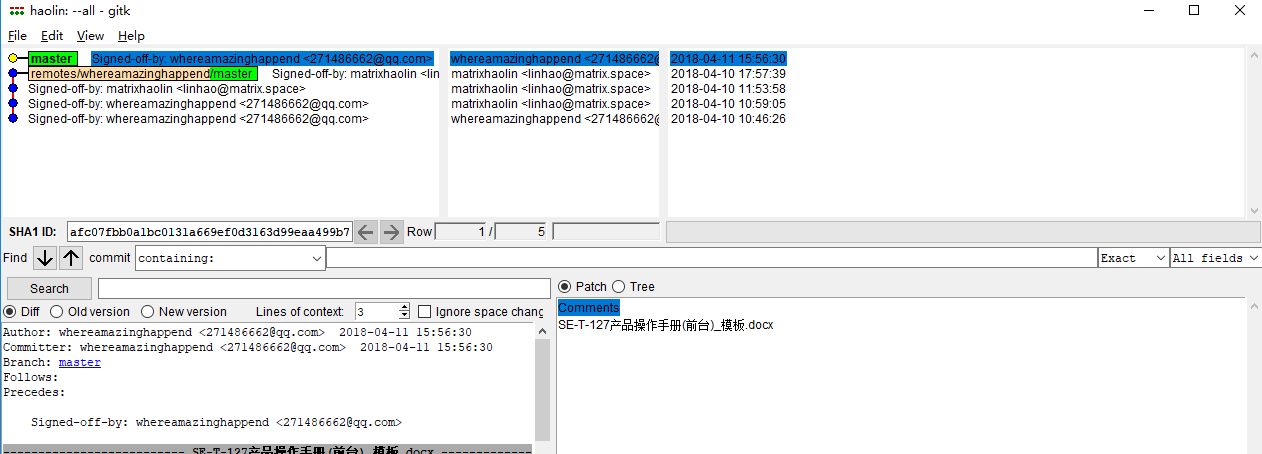


### 3.1.3上传

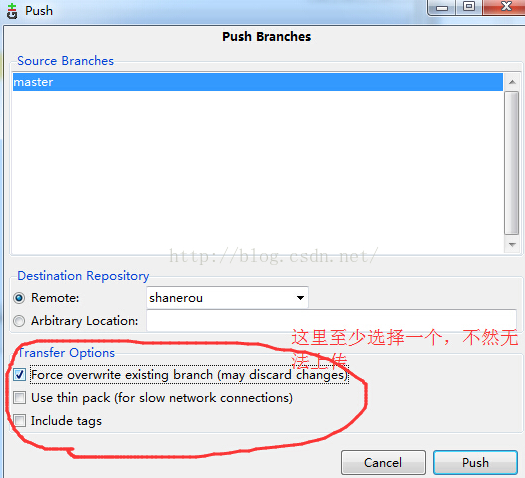
接下来，在文件夹下放入一个文件：SE-T-127产品操作手册(前台)\_模板.doc，并依次执行下图中的操作（以下简称“五步操作”）：



在第四步 commit 后，可以在Repository->Visualize All Branches History中看到提交的历史记录（这是提交到本地而不是提交到远程服务器），如下图所示：



注意：点击Push后，会出现一个对话框：



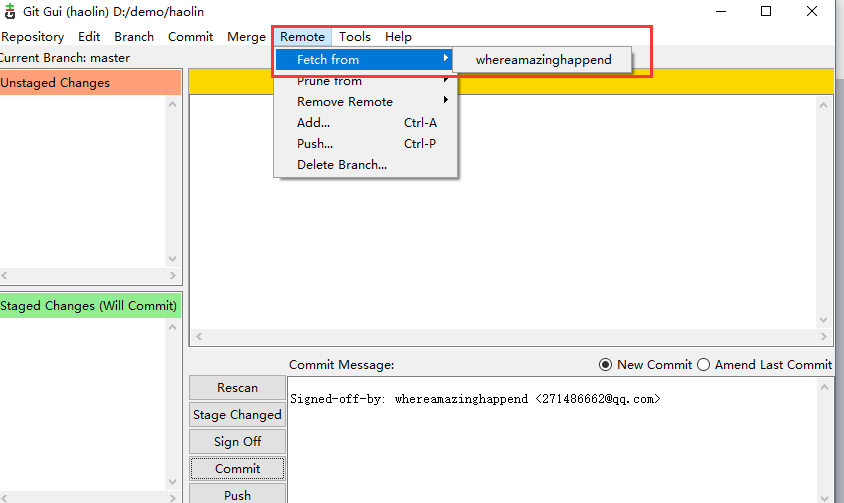
上传完成后，可以登录到github上查看刚刚上传的文件。

另：如果文件内容有更新，或者新增了其他文件，都需要重新执行“五步操作”

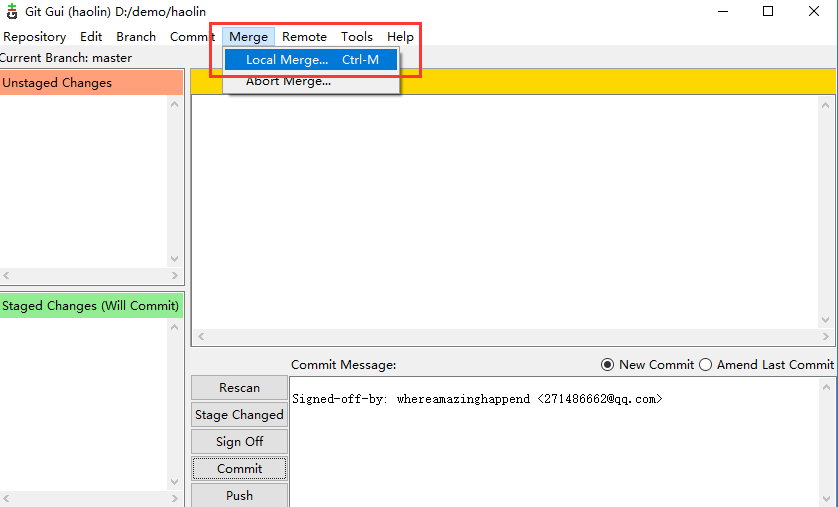
## 3.2 从远程库获取更新

对于多人参与的项目开发，如果每个人都只push到远程，而忽略其他人的进度，毫无疑问会引起问题。最佳惯例是，每次push到远程之前，先从远程把目前进度fetch下来，在和自己目前项目进度合并后，再将它push到远程。

打开Git Gui，在菜单栏中选择Remote->Fetch from->whereamazinghappend，便可以从远程服务器更新到本地，但是尚未与自己当前项目合并。



在菜单栏中选择 Merge->Local Merge，就可以与本地项目完成合并。

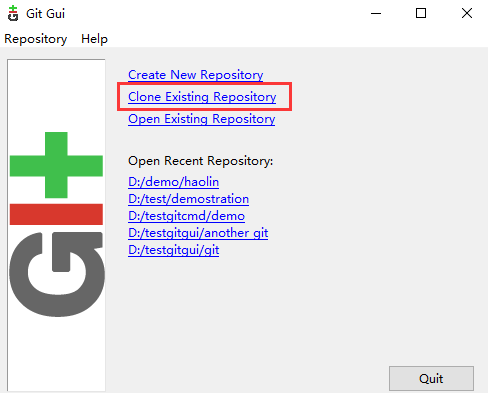


注意：合并过程中可能产生冲突，例如，有一个方法在本地与远程上分别被修改为不同的名称，合并时会遇到冲突，可以右键空白部分选择使用哪个版本（合并冲突解决是一个大课题，后续单独针对不同场景分析）

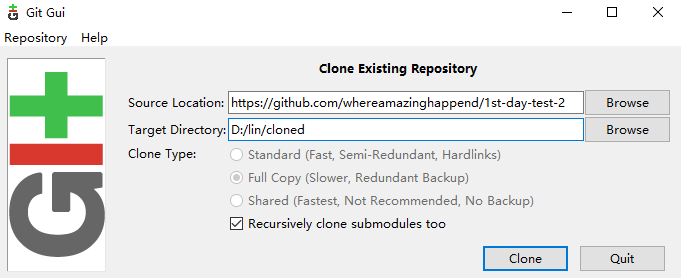
## 3.3 克隆github上的项目到本地

与fetch and merge不同的是，克隆功能是从远程库上把项目完整克隆到本地，本地并没有库，克隆的内容包括日志信息，分支等。

1. 在Git Gui的主界面选择克隆已有版本库 (Clone Existing Repository)。



1. 在弹出的窗口中，Source Location 填写 Github 中项目的地址，Target Directory 填写在本地存放项目的路径。点击Clone，成功从github将项目克隆到本地。



# 4 Git Bash (命令行)

## 4.1 创建与操作本地git库

**1、**进入相应的文件夹下，创建一个空目录

**$ mkdir TEST**

**$ cd TEST**

**$ pwd**

**2、**通过git init命令把这个目录变成Git可以管理的的仓库(根据需要，也可以直接执行第二步)

**$ git init**

**3**、创建一个示例文件：readme.txt

**$ touch reademe.txt**

**4、**把文件添加到暂存区

**$ git add readme.txt**

**如果是克隆的现有仓库，并对项目作出了更新（例如，新增文件、修改文件、删除文件），都需要先加入到暂存区，然后进行commit。**

**对应的命令包括：**

1. **git add -A -----添加所有文件进入暂存区，包括新增的，修改的和删除的**
2. **git add -u ----不涵盖新增的，只添加修改的和删除的**

**5、**把文件提交到本地仓库

**$ git commit -m"message" (message处输入提交信息，例如，wrote a readme file)**

6、查看文件状态

**$ git status**

7、对文件修改后，查看修改的内容

**$ git diff readme.txt**

8、修改后的文件提交到本地仓库

**$ git add readme.txt**

**$ git status**

**$ git commit -m"add distributed"**

**9、查看提交历史（或提交日志）**

**$ git log**

举例：

commit 2d5161f7c4fa7470fac3837f611802fae72ca751 (HEAD -> master, origin/master, origin/HEAD)

Author: whereamazinghappend <38200299+whereamazinghappend@users.noreply.github.com>

Date: Wed Apr 11 11:42:46 2018 +0800

Update 学习笔记.txt

111

commit 4acb213f3030f5b6add9daf7d9b52be49b09a4b4

Author: whereamazinghappend <271486662@qq.com>

Date: Wed Apr 11 11:37:17 2018 +0800

ws

Signed-off-by: whereamazinghappend <271486662@qq.com>

commit 0ce13f6e0ad6503ae227e064deb9bab4fcf4918f

Author: whereamazinghappend <271486662@qq.com>

Date: Wed Apr 11 11:34:36 2018 +0800

Signed-off-by: whereamazinghappend <271486662@qq.com>

**10、**根据commit id回退到指定的版本

$ git reset --hard3628164 (commit id 可以只写前几位)

11、撤销修改

10 中的回退可以理解为对文件进行了修改，且提交到版本库后，想要撤销修改；而如果是提交到版本库前（即工作区中）撤销修改，可以使用：

$ git checkout --readme.txt

## 4.2 克隆现有仓库

命令格式：git clone <https://github.com/whereamazinghappend/test-git-command>

克隆命令拉取的是远程仓库中所有版本的所有文件。如前文所述，即使服务器crash掉，仍可以使用克隆下来的数据重建服务器上的仓库。

## 4.3本地与远程仓库的交互

1. **把本地仓库与远程库建立关联**

$ git remote add origin [git@github.com:michaelliao/learngit.git](mailto:git@github.com:michaelliao/learngit.git)

1. 将本地分支的更新推送到远程主机

完整命令：git push <远程主机名> <本地分支名>:<远程分支名>

（远程主机名一般为 origin）

场景一举例：如果想把本地的分支test 提交到远程仓库，并作为远程仓库的master分支，或者作为另外一个名叫test的分支，那么可以这么做。  
git push origin test:master         // 提交本地test分支 作为 远程的master分支  
git push origin test:test              // 提交本地test分支作为远程的test分支

场景二举例（常见）：如果省略掉远程分支名，则表示将本地分支推送与之存在追踪关系”的远程分支(通常两者同名)，如果该远程分支不存在，则会被新建。

追踪关系一般是基于某个远程分支创建本地分支时，或者克隆时，自动建立的。

这里我们举个例子： 假设我们的网络里有一个在 git.matrix.com 的 git 服务器，也可以是github上的你账户下的地址。 从这里克隆后，Git 的 clone 命令会为你自动将其命名为 origin，拉取它的所有数据，创建一个指向它的 master 分支的指针，并且在本地将其命名为 origin/master。 Git 也会给你一个与 origin 的 master 分支在指向同一个地方的本地 master 分支，这样你就有工作的基础。

# 5分支

首先，我们一定要有一个主分支，包括 master 分支和 develop 分支。master 分支用来版本发布，develop 分支就是我们的日常开发分支，保存了开发的最新代码。最简单的模式一般都包含这两个分支：develop 分支用来开发功能，开发完成并且测试没有问题则将 develop 分支的代码合并到 master 分支并发布。

由于develop 分支只有发布完了才能进行下一个版本开发，可能影响整体开发进度。因此根据具体的flow，我们还可能引入其他分支：

feature：具体的功能开发分支，只与 develop 分支交互。

release：release 分支可以认为是 master 分支的未测试版。比如说某一期的功能全部开发完成，那么就将 develop 分支合并到 release 分支，测试没有问题并且到了发布日期就合并到 master 分支，进行发布。

综上所述，操作其实并不复杂，但是管理略微复杂，必须熟悉不同分支之间的派生、合并关系。

以下以新功能开发工作流为例：

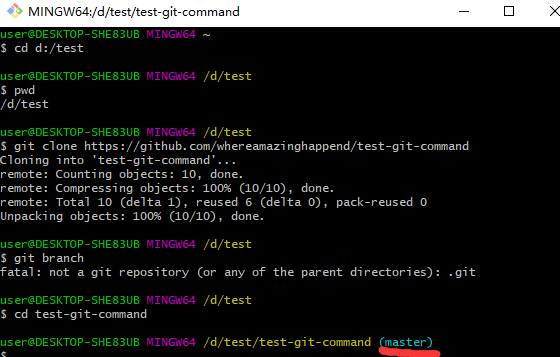
## 5.1切换到本地工作区

cd d:/test

## 5.2从远程仓库克隆代码到本地仓库

git clone <https://github.com/whereamazinghappend/test-git-command>

CD 到克隆好的本地仓库，可以看到，后面有一个master的标记，说明现在是master分支



此处有三个命令：

1. 查看本地所有分支：git branch
2. 查看所有分支：git branch -a
3. 查看远程所有分支：git branch -r

## 5.3 基于本地master分支，创建本地develop分支

切换到master分支

$git checkout master

基于master分支克隆develop分支，并在克隆完毕后直接跳转到develop分支

$git checkout -b develop

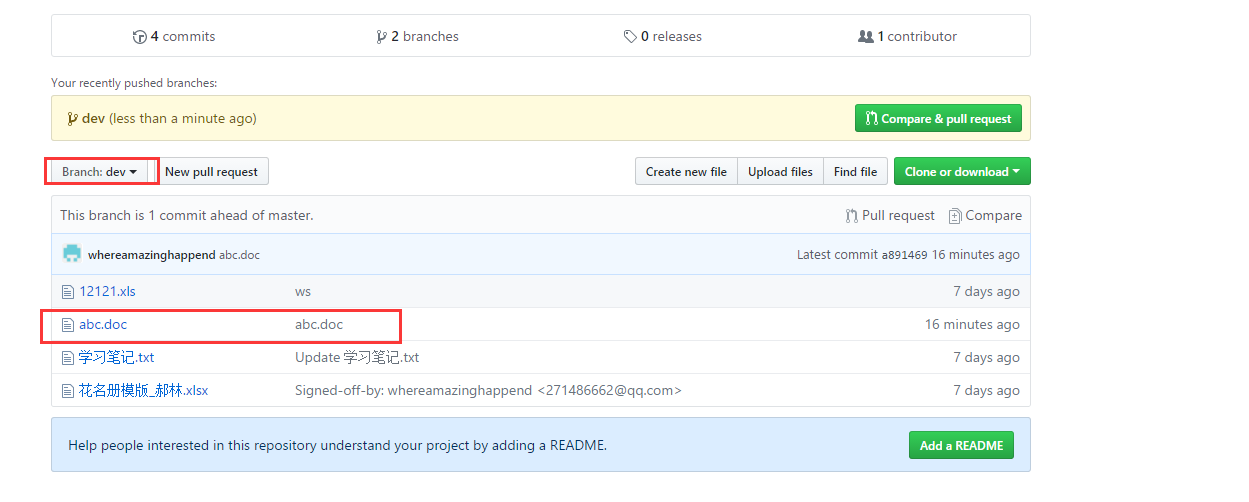
推送develop分支到远程仓库

$git push origin develop

## 5.4 本地开发后代码推送到远程dev分支

由于开发工作是基于dev分支，因此我们在这个分支下增删改一些文件，相当于完成特定功能点的开发后，把代码提交到本地仓库

1. 执行命令：touch abc.doc，增加一个文件进入暂存区
2. 执行：git add abc.doc，增加文件至工作区（此时可以看到文件夹下增加了这个文件）
3. 提交到本地仓库：git commit -m “taskID:001”(如果不执行步骤2和3，那么所做的改动同样会反映到master分支)
4. 打个标签：git tag -m “add smart contract” v0.1 (不是每一个Tag都需要提交到远程仓库，比如可以在完成一个功能点的编码工作后未编译就打一个包，仅存储于本地仓库，在编译成功&测试通过后，再打一个新的Tag包（里程碑Tag包），仅将里程碑Tag包推送到远程仓库)
5. 代码推送到远程仓库：git push origin dev

**重要：1. 为了防止代码冲突，先执行拉取git pull**

**编译未通过的代码不要提交到远程，可以先提交到本地仓库，全部开发完毕并编译通过后，推送到远程。**

## 5.5发布到测试分支开展测试工作（管理人员操作）

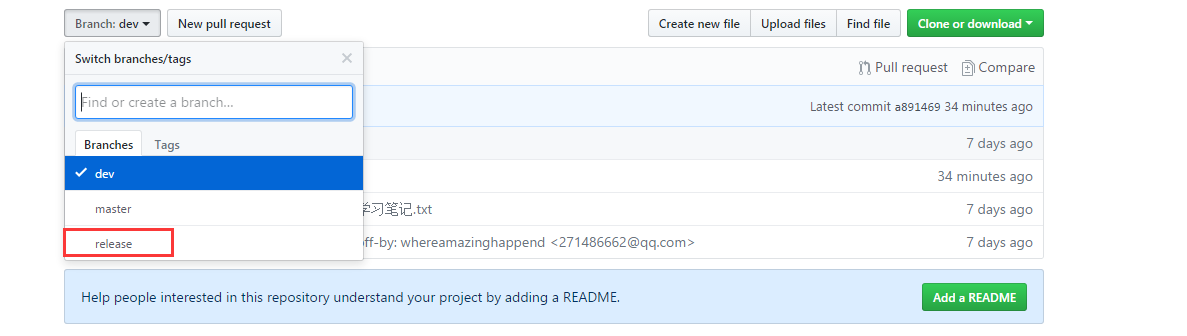
阶段性的开发工作完成后，我们需要把代码发布到测试分支供测试人员使用。

git checkout dev --切换到dev分支

git checkout -b release --基于dev分支创建release分支，并切换

git push origin release --推送到远程

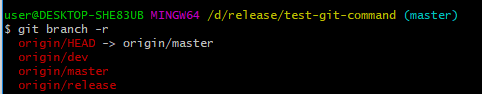
结果如图所示：



## 5.6测试人员提交bug后，开发从远程拉取代码到本地

1. 进入到 release文件夹下克隆仓库：$ git clone <https://github.com/whereamazinghappend/test-git-command> （只会clone master分支）
2. 查看远程仓库分支情况：克隆仓库时只能克隆master分支，因此需要拉取指定分支，

git branch -r 展示远程所有分支



1. 拉取测试分支（即: 基于远程release创建并切换到本地release）：git checkout -b release origin/release

成功提示：user@DESKTOP-SHE83UB MINGW64 /d/release/test-git-command (master)

$ git checkout -b release origin/release

Switched to a new branch 'release'

Branch 'release' set up to track remote branch 'release' from 'origin'.

1. 修改步骤如前文所讲，注意commit 时添加描述bug id keyword

## 5.7测试通过后，把代码合并到dev分支

首先切换到dev分支：git checkout dev

执行合并：git merge release

## 5.8 开发测试全部完成后，合并到master分支

git checkout master

git merge develop

# 6 Tag

前文中提到过，标签一般用于版本发布，给特定时间点的版本做标记。

标签种类有两种：

1. 轻量标签：只是指向所提交对象的引用，格式为 git tag vXXX
2. 附注标签：可以添加附注，格式为：git tag -m “desc” vXXX

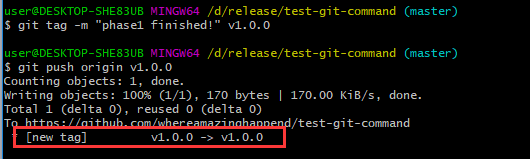
## 6.1 创建tag

一般在阶段性开发工作完成后，我们会打一个tag

先切换到master分支：git checkout master

创建tag：git tag -m “phase1 finished!!” v1.0.0

推送tag到远程：git push origin v1.0.0



## 6.2 查找和列出tag

列出所有tag: git tag

查找符合要求的tag：git tag -l‘v0.1.\*’

## 6.3发布后发现bug，基于tag创建一个本地分支

git checkout -b [创建的分支名称] [里程碑Tag名称]

例如：git checkout -b bugfix-v1.0.0 v1.0.0

## 6.4修复后查询更改的内容

git diff

## 6.5修复完毕后，要合并到dev和master分支

合并到develop

git checkout develop

git merge bugfix-v1.0.0

提交到远程仓库develop分支

git push origin develop

合并到master：如果随下一个版本再发布，可不用合并至master分支

git checkout master

git merge develop

提交到远程仓库master分支

git push origin master

## 6.6合并完后，打一个新的tag，推送到远程

git tag -m "Bug#001 fixed" v1.0.1

git push origin v1.0.1

## 6.7删除本地的bug 分支

合并完成后，bug分支就没有用了，可以删除

git branch -d bugfix-v1.0.0

git push origin: bugfix-v1.0.0