**Git基础总结**

# 创建目录

## 1. mkdir <file>：创建一个目录文件夹；

## 2. cd <file>：目录名 进入该目录；

## 3. pwd：显示当前所在目录路径；

# 将文件目录转换为git库并放置文件

## 1. 将目录转为git后缀的库

**git init：**将当前目录变为Git可管理仓库；返回Initialized empty Git repository in /Users/guoxiang/Documents/目录名称/.git/

**注：**若未看到。Git目录，是因为此目录是隐藏的，使用ls –an即可看见。

## 2. 将配置文件放置目录下并查看目录下文件状态

git status：标出所在库中所有文件状态。(需要cd进入该库)

**注：**修改未add提交，已add提交未commit提交，Nothing to commit(没有可提交)

## 总结

### ①．git init：将当前目录转变为Git仓库。

### ②．git status：显示库中文件状态。

### ③．ls –an：显示当前目录隐藏文件。

# 将目录下的文件/文件夹上传

## 1. 将文件转为“准备提交阶段”。

git add <file>：将文件转到准备上传阶段。git add .(将所有上传)

## 2. 将文件转为“已提交阶段”。

git commit –m “备注”：上传所有准备阶段文件，指定上传某个准备阶段的文件。(git commit -m)

**注：**file changed(被修改的文件数)、insertion(插入调整内容)、deletion(删除调整内容)

## 3. 上传完毕后查看此目录文件状态

查看方法→[[戳我]](#查看库中文件状态)

## 总结：

### ①．git add <File>：将本地文件上传至暂存区。

### ②．git commit –m “备注”：将暂存区的文件上传至本地库。

# 修改库中文件与查看修改前后对比

## 未commit提交前查看查看文件修改前后对比。

git status：标出所在库中所有文件状态。(需要cd进入该库)

**注：**修改未add提交，已add提交未commit提交，Nothing to commit(没有可提交)

git diff --<file>：查看指定文件内容修改的比对差异，只有处于“红色”及“绿色”状态下的文件才能查看前后差异。

git diff --cached：是查看“暂存区”与“版本库”的差别。

git diff HEAD -- <file>：是查看“工作区”和“版本库”的差别。

## 已完成提交的文件可根据上传备注查看历史记录。

git log：查看历史修改记录+(--pretty=oneline)缩略-横向展示历史记录，缩略显示不显示提交时间、提交用户。

**注：**缩略查看“git l”设定，键入git config --global alias.l "log --color --graph --decorate --abbrev-commit --pretty=format:'%Cblue%h %Cgreen%ad %Cred%an %Cgreen%d %Creset%s' --date=short"

## 总结

### ①．git diff --<File>：查看指定文件“工作区”与“暂存区”的差别。

### ②．git diff --cached：查看 “暂存区”与“版本库”的区别。

### ③．git diff --cached <File>：查看指定文件“暂存区”与“版本库”的区别。

### ④．git diff HEAD -- <File>：查看 “工作区”和“版本库”的差别。

### ⑤．git log：查看显示commit记录。

### ⑥．git log --pretty=oneline：缩略显示commit记录。

# 根据所查记录回退指定版本或返回指定版本

## 根据git log查看历史记录，记录目录按照先后顺序排列，第一列为最新提交版本，第二列次之，以此类推。

**注：**此序列“c4d8c4fded512888577ac9e37f087c26490ac961”为commit提交的版本号。

## 回退至上一个版本或上上个版本。

git reset HEAD^：将当前版本回退至上一次的版本。

**注：**HEAD表示为当前版本，HEAD^表示为上一个版本，以此类推HEAD^^既是回退到上上个版本，若版本较多，可调整为HEAD~2既回退两个版本。

## 回退后可查看当前版本是否是需要回退的版本。

cat <file>：显示文件内容。或使用git log根据add commit时的备注，查看是否到达该版本。

## 前往未来的版本。

git reset –hard <cmmit\_id>：根据未来版本id序列，可指定回到未来某个版本中。

**注：**版本号没必要写全，前几位即可，Git会自动去找。当然不能只写1个。

**注：**若终端已退出，没有之前的版本id记录，可用git reflog来重现操作记录，并从中找到。

## 指定单个文件版本回退。

git log <File>：查看此文件版本提交记录，记录要回退的commit序列。

git reset < commit\_id> <File>：将版本库指定文件回退至指定版本。

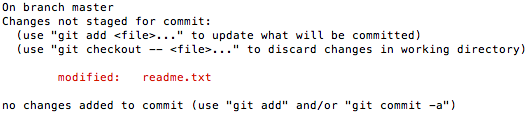
git checkout <File>：将回退的版本库指定文件同步至工作区。

# 如何撤销修改

## 1. 修改后查看状态；

git status：标出所在库中所有文件状态。(需要cd进入该库)

**注：**修改未add提交，已add提交未commit提交，Nothing to commit(没有可提交)



Git commit –a

将工作区转入暂存区并上传至版本库中

Git checkout-<file>

可以撤销修改

## 2. 撤销工作区文件或撤销暂存区文件的修改；

git checkout -- <file>：将版本库中的指定文件，同步到工作区中。

git reset HEAD <file>：可将指定暂存区的修改撤销掉，重新放回工作区。

注：文件回到最近一次“git commit“或”git add“时的状态。

**①．** 当“目标文件“修改后未git add <file>，撤销将回退到版本库当前版本指定文件；

**②．** 当“目标文件“修改后已git add <file>，撤销将回退到暂存区当前版本指定文件；

**②．** 多文件撤销可直接使用git checkout .来完成一次性撤销。

# 文件的删除

## 直接手动删除或rm命令删除；

rm <File>：删除当前所在目录下指定文件。

## 从工作区删除某文件，并查询哪个文件被删除了。

git status：显示工作区目录状态，通常红色表示已修改未add的文件，但此时文件被删除时，则也会红色表示，且会告知你删除了test.txt文件（既：deleted：test.txt）。

注：确定删除，就用git rm <File>删除，并git commit上传版本库。

注：失误删除，就用git checkout -- <File>，从版本库中同步到工作区中。

## 删除命令与恢复命令误区

rm<File>或手动删除，与执行git rm <File>是不同概念，git rm <File>是删除并同步暂存区。执行git rm <File>后是无法执行git chekcout -- <File>的。

**原因：**此时暂存区的是一条deleted命令，而非File文件，因此当前只有版本库中才有File文件。

**解法：**使用git checkout -- <File>无用，是因为此命令是同步版本库用的，但是工作区已无此文件可用于同步，此时应该是撤销deleted命令，将删除命令返回工作区，然后再使用git checkout -- <File>同步版本库。

git reset HEAD <File>：可将指定暂存区的修改撤销掉，重新放回工作区。

git checkout -- <File>：将版本库中的指定文件，同步到工作区中。

**注：**撤销存于暂存区的命令始终使用git reset HEAD < File >撤销。

**注：**虽删除指令在工作区，且工作区已无该文件，但可将删除命令视为对文件的修改，只要修改在工作区，即可执行git checkout -- <File>，同步版本库，取消删除指令。

# git远程仓库

# 注册Git仓库托管服务

## 进入GitHub官网注册：<https://github.com/>。

## 查看是否有.ssh目录及id\_rsa和id\_rsa.pub两个文件。

cd ~/.ssh：进入ssh目录。

ls：显示当前目录（既ssh目录下），查看有无id\_rsa和id\_rsa.pub两个文件。

**注：**在.SSH中id\_rsa和id\_rsa.pub是SSH Key的秘钥，id\_rsa是私钥，不能泄露，id\_rsa.pub是公钥，可以告诉他人。

## 没有.ssh目录和id\_rsa和id\_rsa.pub两个文件——进行创建SSH Key

ssh-keygen -t rsa -C "E-mails"：创建秘匙，并输入账号、密码、确认密码。

## 创建GitHub仓库

根据下图所示创建远程仓库，在SSH Key创建时，任意填写Title，在Key文本框里粘贴id\_rsa.pub文件的内容，id\_rsa.pub文件的内容可用cat <File>查看。

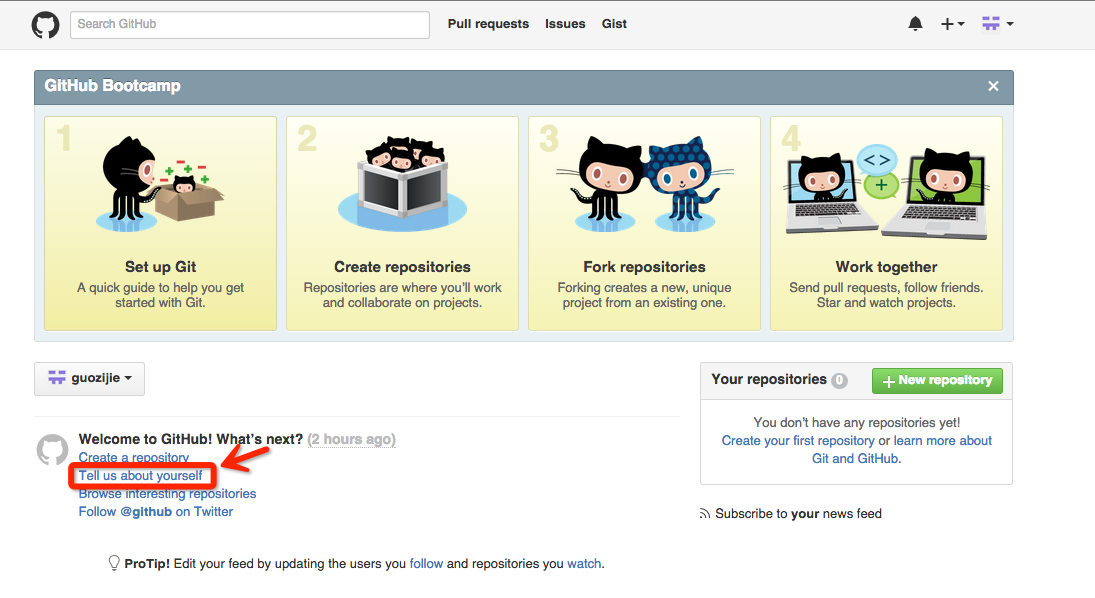


图1：Tell us about yourself

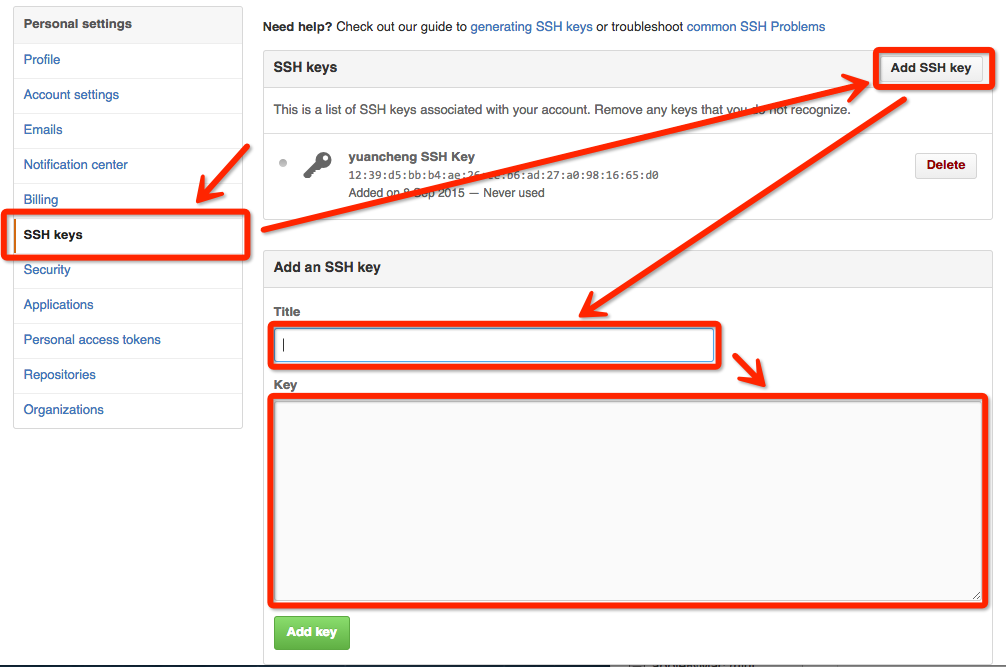


图2：SSH Keys→Add SSH Key→Title→Key

# 在GitHub创建新的远程仓库

## 如下图所示，登陆GitHub，然后在左下角点击“Create a repository”按钮，进入远程仓库创建界面；

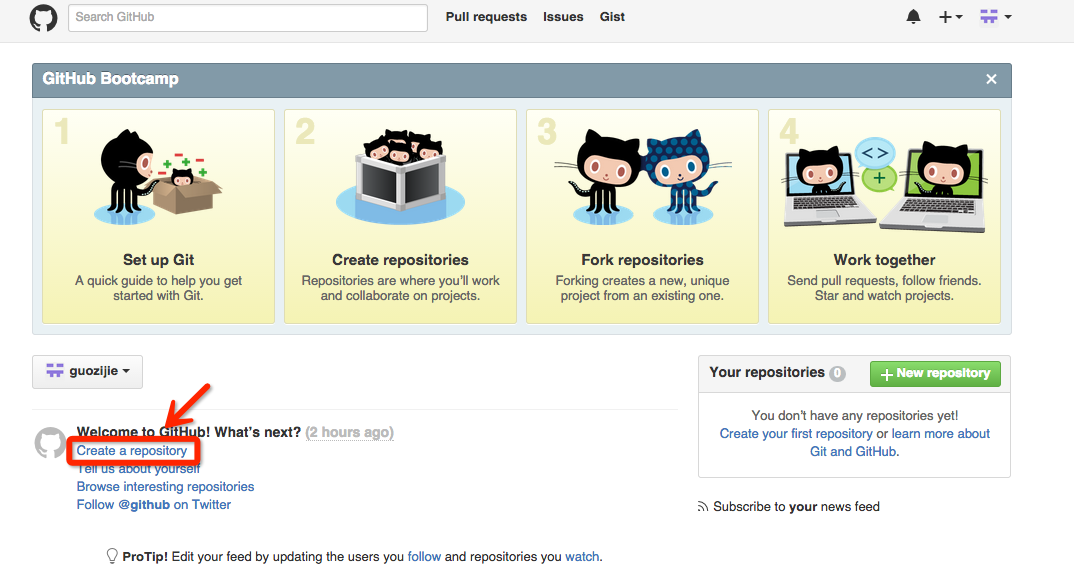


图3：Create a repository

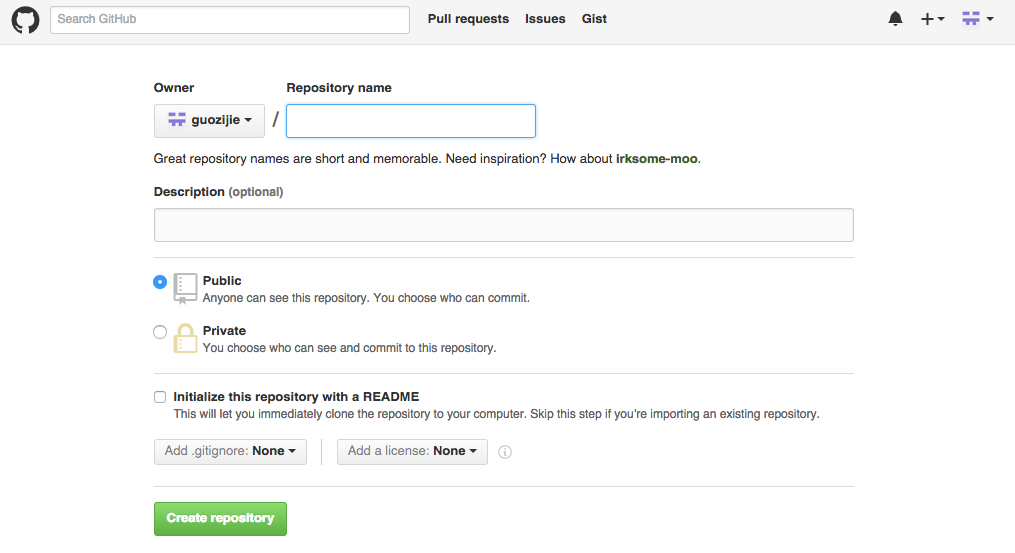


图4：远程仓库创建界面

## 创建一个仓库

在Repository name内输入库名，其余保持默认设置，并完成创建，创建完毕。

## 创建一个远程仓库

git remote add <库名> git@github.com:< GitHub账户名>/learngit.git

**注：**创建完毕后，可用git remote将远程仓库名称简略的列出来。

**注：**也可用git remote –v列出远程仓库的详情。

## 将本地版本库所有内容推送至远程仓库中

git push -u origin master：本地推送远程使用git push命令，将当前分支master推送到远程。

**注：**由于远程库是空的，第一次推送master分支时，加上-u参数，Git不但会把本地的master分支的内容推送至远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

**注：**若关联了错误的远程仓库，可以使用git remote命令调出远程仓库名称，然后使用git remote rm <>将远程仓库删除，再删除后重建即可。创建可查看[[第3点]](#创建一个远程仓库)。

## 本地分支自动关联远程分支。

git branch --set-upstream <本地分支名> origin/<远端分支名>：此命令将本地指定分支与远端指定分支自动关联，以后只要进入本地此分支，直接git push或git pull即可，无需再添加其他参数。

验证：

git config -l | grep 'branch\.<当前分支名>'：查看当前分支跟踪远程库的什么分支。

## 本地修改提交远程仓库

git push origin master：将本地master分支的最新修改推送至GitHub。

git pull origin master：将远端更新至本地。

# 从远程库克隆

## 1. 创建远程库

创建远程库的方法戳这里→[[点我]](#在GitHub创建新的远程仓库)，但是在创建时勾选“Initialize this repository with a README”这样GitHub会自动创建“README.md”文件。

## 2. 从远程仓库克隆到本地库

git clone git@github.com:<GitHub\_id>/<目标库>.git：远程库克隆至本地当前目录。

git clone git@github.com:<GitHub\_id>/<目标库>.git --depth 1：克隆指定远程库最近一次commit版本，depth为深度，1则表示克隆最近1次。

# 分支管理

# 创建与合并分支

## 创建分支-及概念示意图

git branch <分支name>：带上分支名，即为创建@名称的分支。

git checkout <分支name>：带上分支名，即为切换至@分支。

git checkout –b <分支name>：命令加上-b参数表示创建并切换分支，相当于上述两种命令的结合。

git push --set-upstream origin <新建分支名>：创建后上传至远端分支，并执行本地与远程分支关联。

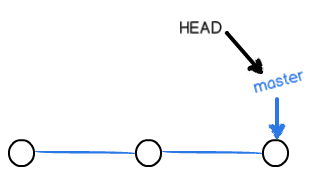


图5：master主分支

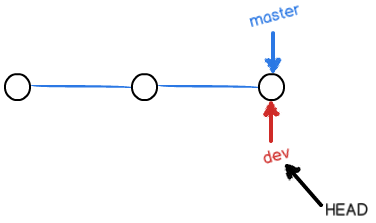


图6：主分支外创建一个分支

## 查看所有分支

git branch：列出本地所有分支，同时会在当前分支前显示“\*”号。

git branch -r：列出远端所有分支。

git branch -a：列出远端和本地所有分支。

**注：**切换分支后，所有操作修改，仅是针对所创建的分支进行的，而master主分支版本进度则终止在分支切换上。

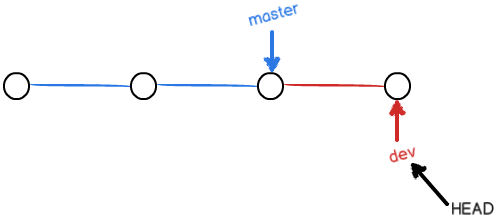


图7：创建分支后进行修改并提交 新建分支版本超过master版本

## 主副分支合并

### merge全部融入：在确保目标分支与当前分支最新的情况下，执行git merge <目标分支>命令，是将“目标分支”的差异提交全部同步至“当前所在分支”的操作，同步后需要执行commit push上传；

### cherry-pick：指定某次修改融入：确保当前所在分支为最新的情况下，执行git cherry-pick <目标分支的commitID>，将目标分支的某次提交同步至当前分支，多条执行时，按照时间线由远及近覆盖，注“④”。

### 注：

### ①. 尽量保持"开发分支"向"目标分支"（外放分支）融入，避免"目标分支"融入"开发分支"；

### ②. 若改动少且熟悉，尽量使用cherry-pick进行指定融入；

### ③．若能确定目标分支完全覆盖当前分支，一切冲突数据均使用目标分支时，遇到冲突，可执行git checkout —theirs <冲突文件>，可强行将对方文件覆盖己方文件，反之则使用git checkout —ours <冲突文件>，无视冲突使用己方文件；

### ④. 1个文件两次提交，产生2个commitID，cherry-pick其中1个commitID时，不会将另一个提交同步，因此有可能会造成丢失的情况。

**注:** git branch：使用此命令查看当前所在分支。

$ git merge dev

Updating d17efd8..fec145a

Fast-forward

readme.txt | 1 +

1 file changed, 1 insertion(+)

**注：**返回信息会告知本次合并的模式，此次合并为“Fast-forward”模式，既“快进模式”。

## 删除分支

git branch -d <分支name >：加-d和分支名将删除指定分支，-d=Deleted。

**注：**删除后使用git branch即可查看分支目录中是否已经删除指定分支。

# 解决冲突

## A 分支与 B 分支有相同位置修改时，合并将产生冲突。

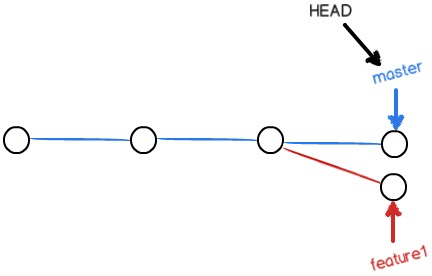


图9：分支合并产生冲突

$ git merge feature1 //将feature1分支合并至当前master分支。

Auto-merging readme.txt //自动合并“readme.txt”文档。

CONFLICT (content): Merge conflict **in** readme.txt //内容冲突 合并冲突文档

Automatic merge failed; fix conflicts **and** **then** commit the result.//自动合并失败，需解决冲突后再行提交。

## 修改冲突文档

Git is a distributed version control system.

Git is free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

<<<<<<< HEAD //当前所在分支

Creating a new branch is quick & simple. 冲突内容

=======

Creating a new branch is quick AND simple. 冲突内容

>>>>>>> feature1//冲突分支

**注：**只需要将标出的删除并调整成想要的修改即可，然后上传。

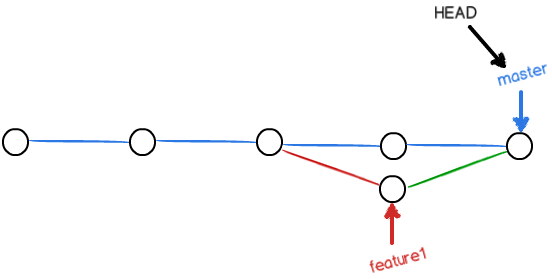


图10：成功合并后示意图

## 查看分支合并情况

git log --graph --pretty=oneline --abbrev-commit：查看分支合并记录。

**注：**合并后无用的分支线即可删除→[[戳我]](#删除分支)。

三、分支管理策略

**说明：**一般情况下的分之合并，Git会自动使用“Fast Forward”模式合并，既快速合并分支，但快速合并分支后，若删除经过快速合并的分支，会丢失分支信息，因此可以禁用“Fast Forward”方式合并，这样Git就会在“merge”合并时生成一个新的commit序列，这样即可在分支历史上看到分支信息。

git merge --no-ff -m "merge with no-ff" dev：合并分支，将dev并入master，并禁用“Fast Forward”模式合并。

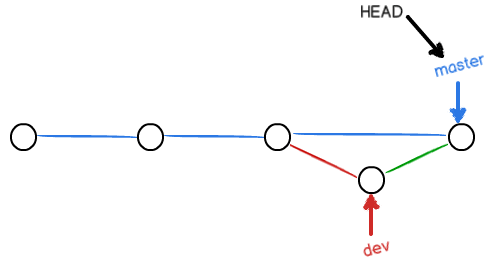


图11：递归合并结果示意图

总结：

①．--no-ff：本次合并分支禁用“fast forward”快速合并，使用递归方式进行。

②．--no-ff –m：创建新的commit，加上-m，将commit描述写入log。