## git init

**Git 使用 git init 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以 git init 是使用 Git 的第一个命令。**

在执行完成 **git init** 命令后，Git 仓库会生成一个 .git 目录，该目录包含了资源的所有元数据，其他的项目目录保持不变（不像 SVN 会在每个子目录生成 .svn 目录，Git 只在仓库的根目录生成 .git 目录）。

使用我们指定目录作为Git仓库。

**git init newrepo**

**newrepo为指定的目录**

初始化后，会在 newrepo 目录下会出现一个名为 .git 的目录，所有 Git 需要的数据和资源都存放在这个目录中。

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，然后提交：

$ git add \*.c

$ git add README

$ git commit -m '初始化项目版本'

以上命令将目录下以 .c 结尾及 README 文件提交到仓库中。

Git 教程

[Git 教程](http://www.runoob.com/git/git-tutorial.html" \o "Git 教程" \t "http://www.runoob.com/git/_top)[Git 安装配置](http://www.runoob.com/git/git-install-setup.html" \o "Git 安装配置" \t "http://www.runoob.com/git/_top)[Git 工作流程](http://www.runoob.com/git/git-workflow.html" \o "Git 工作流程" \t "http://www.runoob.com/git/_top)[Git 工作区、暂存区和版本库](http://www.runoob.com/git/git-workspace-index-repo.html" \o "Git 工作区、暂存区和版本库" \t "http://www.runoob.com/git/_top)**[Git 创建仓库](http://www.runoob.com/git/git-create-repository.html" \o "Git 创建仓库" \t "http://www.runoob.com/git/_top)**[Git 基本操作](http://www.runoob.com/git/git-basic-operations.html" \o "Git 基本操作" \t "http://www.runoob.com/git/_top)[Git 分支管理](http://www.runoob.com/git/git-branch.html" \o "Git 分支管理" \t "http://www.runoob.com/git/_top)[Git 查看提交历史](http://www.runoob.com/git/git-commit-history.html" \o "Git 查看提交历史" \t "http://www.runoob.com/git/_top)[Git 标签](http://www.runoob.com/git/git-tag.html" \o "Git 标签" \t "http://www.runoob.com/git/_top)[Git Github](http://www.runoob.com/git/git-remote-repo.html" \o "Git 远程仓库(Github)" \t "http://www.runoob.com/git/_top)[Git CODING](http://www.runoob.com/git/git-remote-repo.html" \l "git-coding" \o "Git Coding" \t "http://www.runoob.com/git/_top)[Git 服务器搭建](http://www.runoob.com/git/git-server.html" \o "Git 服务器搭建" \t "http://www.runoob.com/git/_top)

[Git 工作区、暂存区和版本库](http://www.runoob.com/git/git-workspace-index-repo.html" \o "Git 工作区、暂存区和版本库)

[Git 基本操作](http://www.runoob.com/git/git-basic-operations.html" \o "Git 基本操作)

# Git 创建仓库

本章节我们将为大家介绍如何创建一个 Git 仓库。

你可以使用一个已经存在的目录作为Git仓库。

## **git init**

Git 使用 **git init** 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以 **git init** 是使用 Git 的第一个命令。

在执行完成 **git init** 命令后，Git 仓库会生成一个 .git 目录，该目录包含了资源的所有元数据，其他的项目目录保持不变（不像 SVN 会在每个子目录生成 .svn 目录，Git 只在仓库的根目录生成 .git 目录）。

### **使用方法**

使用当前目录作为Git仓库，我们只需使它初始化。

git init

该命令执行完后会在当前目录生成一个 .git 目录。

使用我们指定目录作为Git仓库。

git init newrepo

初始化后，会在 newrepo 目录下会出现一个名为 .git 的目录，所有 Git 需要的数据和资源都存放在这个目录中。

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，然后提交：

$ git add \*.c

$ git add README

$ git commit -m '初始化项目版本'

以上命令将目录下以 .c 结尾及 README 文件提交到仓库中。

**.git 这个子目录。 这就是你的 Git 仓库了，所有有关你的此项目的快照数据都存放在这里。**

## **git clone**

我们使用 **git clone** 从现有 Git 仓库中拷贝项目（类似 **svn checkout**）。

克隆仓库的命令格式为：

git clone <repo>

**参数说明：**

* **repo:**Git 仓库。
* **directory:**本地目录。

比如，要克隆 Ruby 语言的 Git 代码仓库 Grit，可以用下面的命令：

$ git clone git://github.com/schacon/grit.git

执行该命令后，会在当前目录下创建一个名为grit的目录，其中包含一个 .git 的目录，用于保存下载下来的所有版本记录。

***如果要自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：***

$ git clone git://github.com/schacon/grit.git mygrit

我演示该命令的时候加了 ***-s 参数，以获得简短的结果输出***。如果没加该参数会详细输出内容：

**ls 和 ls -a**

**展示当前目录的所有文件，只要是在仓库里的，都会显示**

git statu 和 git status -s

git status 命令用于查看项目的当前状态

查看仓库文件的状态，？？是没有添加到缓存

git add 命令可将该文件添加到缓存，commit前的操作（commit--》暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。）

详细可以去1文档看看

***修改文件(esc,insert键，在操作模式下，:wq 为写入退出，:q! 不保存退出 )***

vim filename

**Vim可以修改编辑某个文件，Insert可以切换插入和替换功能，如果修改完成，可以按esc退出键，然后输入:wq即可退出并保存**

$ git status -s

AM README

A hello.php

***"AM" 状态的意思是，这个文件在我们将它添加到缓存之后又有改动。改动后我们再执行 git add 命令将其添加到缓存中：***

$ git add .

$ git status -s

A README

A hello.php

### **git diff**

执行 git diff 来查看执行 git status 的结果的详细信息。

git diff 命令显示已写入缓存与已修改但尚未写入缓存的改动的区别。git diff 有两个主要的应用场景。

* 尚未缓存的改动：**git diff**
* 查看已缓存的改动： **git diff --cached**
* 查看已缓存的与未缓存的所有改动：**git diff HEAD**
* 显示摘要而非整个 diff：**git diff --stat**

**git commit -am '修改 hello.php 文件'**

[master 71ee2cb] 修改 hello.php 文件

1 file changed, 1 insertion(+)

### **git reset HEAD**

git reset HEAD 命令用于取消已缓存的内容。

我们先改动文件 README 文件，内容如下：

# Runoob Git 测试# 菜鸟教程

hello.php 文件修改为：

<?php

echo '菜鸟教程：www.runoob.com';

echo '菜鸟教程：www.runoob.com';

echo '菜鸟教程：www.runoob.com';?>

现在两个文件修改后，都提交到了缓存区，我们现在要取消其中一个的缓存，操作如下：

$ git status -s

M README

M hello.php

$ git add .

$ git status -s

M README

M hello.pp

$ git reset HEAD hello.php Unstaged changes after reset:

M    hello.php

$ git status -s

M README

M hello.php

现在你执行 git commit，只会将 README 文件的改动提交，而 hello.php 是没有的。

$ git commit -m '修改'[master f50cfda] 修改

1 file changed, 1 insertion(+)

$ git status -s

M hello.php

可以看到 hello.php 文件的修改并未提交。

这时我们可以使用以下命令将 hello.php 的修改提交：

$ git commit -am '修改 hello.php 文件'[master 760f74d] 修改 hello.php 文件

1 file changed, 1 insertion(+)

$ git statusOn branch master

nothing to commit, working directory clean

简而言之，执行 git reset HEAD 以取消之前 git add 添加，但不希望包含在下一提交快照中的缓存。

### **git rm**

如果只是简单地从工作目录中手工删除文件，运行 **git status** 时就会在 **Changes not staged for commit** 的提示。

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除，然后提交。可以用以下命令完成此项工作

git rm <file>

如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项 **-f**

git rm -f <file>

如果把文件从暂存区域移除，但仍然希望保留在当前工作目录中，换句话说，仅是从跟踪清单中删除，使用 **--cached** 选项即可

git rm --cached <file>

如我们删除 hello.php文件：

$ git rm hello.php

rm 'hello.php'

$ ls

README

不从工作区中删除文件：

$ git rm --cached README

rm 'README'

$ ls

README

可以递归删除，即如果后面跟的是一个目录做为参数，则会递归删除整个目录中的所有子目录和文件：

git rm –r \*

进入某个目录中，执行此语句，会删除该目录下的所有文件和子目录。

### **git mv**

git mv 命令用于移动或重命名一个文件、目录、软连接。

我们先把刚移除的 README 添加回来：

$ git add README

然后对其重名:

$ git mv README README.md

$ ls

README.md

创建分支命令：

git branch (branchname)

切换分支命令:

git checkout (branchname)

当你切换分支的时候，Git 会用该分支的最后提交的快照替换你的工作目录的内容， 所以多个分支不需要多个目录。

合并分支命令:

git merge

你可以多次合并到统一分支， 也可以选择在合并之后直接删除被并入的分支。

### **删除分支**

删除分支命令：

git branch -d (branchname)

$ git branch\* master

newtest

$ ls

README        test.txt    test2.txt

$ git merge newtest

Updating 2e082b7..556f0a0Fast-forward

test2.txt | 1 -

1 file changed, 1 deletion(-)

delete mode 100644 test2.txt

$ ls

README        test.txt

### **合并冲突**

合并并不仅仅是简单的文件添加、移除的操作，Git 也会合并修改。

当两个分支同时操作一个文件的时候，才会出现这个情况

冲突的文件要手动修改，再按照正常add,commit提交即可

Head -1 filename.txt

**Cat** filename.txt

可以读取第一行，如果没有-1，读取整个文件数据

生成SSH Keys

ssh-keygen -t rsa -C "3416890246@qq.com"

$ ssh-keygen -t rsa -C "3416890246@qq.com"

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/陈胜君/.ssh/id\_rsa):

打开这里的id\_rsa.pub，拷贝进去就OK啦~

为了验证是否成功，输入以下命令：

$ ssh -T [git@github.com](mailto:git@github.com)

Hi tianqixin! You've successfully authenticated, but GitHub does not provide shell access.

ssh-keygen -t rsa -C "2639824500@qq.com"

<https://www.cnblogs.com/wuer888/p/7655856.html>

抓取远程仓库



github那里创建一个空的项目，只勾选public复选框，复制项目网址，本地抓取git远程仓库

要添加一个新的远程仓库,

陈胜君@DESKTOP-D47U175 MINGW64 /d/GITHUB练习/1/check (master)

$ git remote add origin https://github.com/PerfictGrassland/one.git

查看远程仓库

陈胜君@DESKTOP-D47U175 MINGW64 /d/GITHUB练习/1/check (master)

$ git remote

origin

陈胜君@DESKTOP-D47U175 MINGW64 /d/GITHUB练习/1/check (master)

$ git remote -v

origin https://github.com/PerfictGrassland/one.git (fetch)

origin https://github.com/PerfictGrassland/one.git (push)

eg：

$ mkdir runoob-git-test # 创建测试目录

$ cd runoob-git-test/ # 进入测试目录

$ echo "# 菜鸟教程 Git 测试" >> README.md # 创建 README.md 文件并写入内容

$ ls # 查看目录下的文件

README

$ git init # 初始化

$ git add README.md # 添加文件

$ git commit -m "添加 README.md 文件" # 提交并备注信息[master (root-commit) 0205aab] 添加 README.md 文件

1 file changed, 1 insertion(+)

create mode 100644 README.md

# 提交到 Github

$ git remote add origin git@github.com:tianqixin/runoob-git-test.git

$ git push -u origin master

Git 有两个命令用来提取远程仓库的更新。

1、从远程仓库下载新分支与数据：

git fetch

该命令执行完后需要执行git merge 远程分支到你所在的分支。

2、从远端仓库提取数据并尝试合并到当前分支：

git merge

该命令就是在执行 git fetch 之后紧接着执行 git merge 远程分支到你所在的任意分支。

远程仓库的名字

git push -u origin master

git push

提交到远程仓库的主分支

git remote add test https://github.com/HappyMoment/myfirstProjects.git

git remote remove test

更新本地仓库

<https://www.cnblogs.com/zknublx/p/6113667.html>

**以下两种方法是本地没有做任何改动的时候可以使用的（没有合并冲突）**

***第一种方法：***

$ git fetch origin master 这句的意思是：从远程的origin仓库的master分支下载代码到本地的origin master

1. 查看远程仓库

$ git remote -v

1. 从远程获取最新版本到本地

$ git fetch origin master

$ git fetch origin master 这句的意思是：从远程的origin仓库的master分支下载代码到本地的origin master  
3. 比较本地的仓库和远程参考的区别

$ git log -p master.. origin/master

1. 把远程下载下来的代码合并到本地仓库，远程的和本地的合并

$ git merge origin/master

***第二种方法***  
1.查看远程分支，和上面的第一步相同  
2. 从远程获取最新版本到本地

$ git fetch origin master:temp

**git fetch origin master:temp 这句命令的意思是：从远程的origin仓库的master分支下载到本地并新建一个分支temp**

**3，**比较本地的仓库和远程参考的区别

$ git diff temp

**4. 合并temp分支到master分支**

**$ git merge temp**

合并的时候可能会出现冲突，有时间了再把如何处理冲突写一篇博客补充上。

5.如果不想要temp分支了，可以删除此分支

**$ git branch -d temp**

Deleted branch temp (was d6d48cc).

su@SUCHANGLI /e/eoe\_client/android-app (master)

如果该分支没有合并到主分支会报错，可以用以下命令强制删除**git branch -D** <分支名>

接下来就是，当我们clone别人的项目到自己的库中的时候，如果别人的项目更新了，我们如何把别人的项目更新到本地以及我们在拷贝来的项目做修改后，如何pull request给项目主人，主人觉得我们的修改有用，就会merge(合并request)~

<https://blog.csdn.net/u012426298/article/details/80315149>

<http://www.cnblogs.com/zhishuai/p/7831441.html>

1、git add -A(所有的，当然你也可以指定单个文件)

2、git commit -m"提交描述"

3、git remote add 新建git项目的地址

4、git push -u origin master

***第三种方法***

下载别人的git项目：

可以先fork 后和对方的地址进行关联，但是一般不需要这样；

直接：

git clone 别人的git项目地址（所有都克隆下来）

git pull 别人的git项目地址 master

$ **git remote add** tarobjtu <https://github.com/tarobjtu/WebFundamentals.git> // 这时候可以看到有两个源了

$ git remote

// laispace // tarobjtu

**拉取源仓库的代码到本地**

$ git fetch tarobjtu (master)

**合并源仓库的 master 分支代码到本地**

$ git merge tarobjtu/master

（

**git rebase tarobjtu (查看冲突,手动修改冲突,和 git merge类似 )**

**Git add .**

**Git rebase --continue //add 后不需要git commit**

**Git push -u origin master**

**远程github pull request**

**项目主要人merge request**

）

**提交代码到我们自己的仓库**

$ git add --all

$ git commit -m '合并源仓库代码'

$ git push

这样自己的github就是最新的了，与原始的代码同步。有更新的代码贡献。 可以pull request

合并的时候有冲突，如何处理冲突

<https://www.jianshu.com/p/5e6c8ba3afa2>

一、git fetch 和 git pull 的区别

1、    git fetcj <远程主机名> <远程分支名>:<本地分支名>

         git fetch origin master ：temp             //将远程仓库origin的master分支的

        git diff temp                    //比较本地代码和远程代码的区别

        git merge temp               // 合并temp分支到本地的master分支

2、git pull <远程主机名> <远程分支名>:<本地分支名>

***//取回远程分支的更新，并直接与本地分支合并***

***区别：git  pull 可以看作是git fetch 和 git merge 两个步骤的集合。***

**（以下就要注意了，可以说是很常用的模式，可以查看冲突，手动解决冲突然后进行远程更新）**

二、多人协作，当他人修改文件后，后提交的必须先pull在合并，并且在合并的时候会出现冲突***（使用场景:A修改添加文件，B修改添加文件，B首选要把A的最新修改更新合并到本地，然后push到远程，之后在远程github后台pull request到A那边，A查看pull request，然后merge request，这样就完成了两边的合并与更新，代码一致(这种情况要注意手动解决合并冲突哦)）***

    1、当远程仓库的代码更新后，我们在push提交时会出现提交不了的情况，这时我们必须pull更新后的远程代码。但是在远程代码合并本地代码时发生CONFLICT冲突，这时需要我们手动解决冲突，最后再push提交到远程代码。

    2、解决方法  ：

             git checkout  branch          //选择分支

            git fetch origin master       //拉取远程更新代码（只拉取不合并），这里不能git pull （合并有冲突）

**git rebase  origin/master   //查看conflict ，手动修改冲突**

            git add  .

**git rebase --continue       //add 后不需要git commit**

            git push origin

***下面稍做改变***

    git fetch origin master:temp       //拉取远程更新代码（只拉取不合并），这里不能git pull （合并有冲突）

**git rebase temp**         //查看conflict ，手动修改冲突

            git add  .

**git rebase --continue**

**//add 后不需要git commit**

**小提示：**

**$ git status**

**On branch master**

**Your branch is ahead of 'origin/master' by 5 commits.**

**(use "git push" to publish your local commits)**

**nothing to commit, working tree clean**

            git push origin master //推送到远程分支

**如果是直接git pull origin或者git fetch origin master(:temp) + git merge orgin/mstet(temp)，会有合并冲突，本地修改后，然后手动修改，接着**

**Add,commit等在push上去就ok。另一种就是，在获取远程源之后，git rebase temp**

**查看conflict冲突，修改之后，git add . 不用提交，直接运行git rebase --continue**

**最后推送就可以了，git push /git push origin /git push -u origin master**

**删除远程分支**

**目前项目中有两个remote**

**$ git remote -v**

**现在想删除其中的一个test，方法为:**

**git remote remove <name>**

**$ git remote remove test**

**github克隆别人的项目后，在原作者修改后，本地获取最新项目，合并修改后推送到Github远程仓库后，再把自己和原作者合并的项目pull request到作者的github仓库里，作者阅读同意即可merge你的request，这样就实现了两个仓库代码同步更新了~**

**Git clone [https://\*\*\*\*\*](https://*****)**

**Git remote add neworigins [https://\*\*\*\*](https://****)**

**修改添加等操作后**

**Git fetch neworigins master**

**...**

**//Git pull(git fetch,git merge neworigins/master)**

**--->(git merge)--->git rebase neworigins/master**

**-->git diff -->手动解决冲突（git rebase的好处就是不显示冲突警告）--》git add . --->git rebase --continue(不用git commit -m “\*\*\*”)---》git status --》没问题后---》git push origin master //推送到远程分支**

**个人QQ；**

**夜。铭星**

**2291259064**

**abc822912**

**black@moon**

**3416890246**

**abc822912**

**aloneWolf**

**2639824500**

**abc822912**

**生成SSH Keys**

**ssh-keygen -t rsa -C "3416890246@qq.com"**

**$ ssh-keygen -t rsa -C "3416890246@qq.com"**

**Generating public/private rsa key pair.**

**Enter file in which to save the key (/c/Users/陈胜君/.ssh/id\_rsa):**

**打开这里的id\_rsa.pub，拷贝进去就OK啦~**

**为了验证是否成功，输入以下命令：**

**$ ssh -T git@github.com**

**Hi tianqixin! You've successfully authenticated, but GitHub does not provide shell access.**

**https://www.cnblogs.com/wuer888/p/7655856.html**

**抓取远程仓库**