一、Git

[一、Git 1](#_Toc498075912)

[1.集中式vs分布式 2](#_Toc498075913)

[1.1集中式版本控制系统（SVN和CVS） 2](#_Toc498075914)

[1.2分布式版本控制系统（Git） 2](#_Toc498075915)

[2.Git的安装 2](#_Toc498075916)

[3.创建版本库 2](#_Toc498075917)

[4.时光机穿梭 3](#_Toc498075918)

[5.版本回退 4](#_Toc498075919)

[6.工作区和暂缓区 4](#_Toc498075920)

[6.1工作区（Working Directory） 4](#_Toc498075921)

[6.2暂缓区 5](#_Toc498075922)

[7.撤销修改 6](#_Toc498075923)

[8.删除文件 6](#_Toc498075924)

[9.远程仓库—GitHub 6](#_Toc498075925)

[9.1获取本机的ssh\_key与GitHub链接 6](#_Toc498075926)

[9.2添加远程库 6](#_Toc498075927)

[9.3从远程库克隆 7](#_Toc498075928)

[10.分支管理 7](#_Toc498075929)

[10.1创建于合并分支 7](#_Toc498075930)

[10.2Bug分支 8](#_Toc498075931)

[10.3Feature分支 8](#_Toc498075932)

[10.4多人协作 8](#_Toc498075933)

[11.标签管理 9](#_Toc498075934)

[11.1创建标签 9](#_Toc498075935)

[11.2操作标签 9](#_Toc498075936)

[12.GitHub和码云（gitee） 9](#_Toc498075937)

[12.1GitHub 9](#_Toc498075938)

[12.1码云 9](#_Toc498075939)

# 1.集中式vs分布式

## 1.1集中式版本控制系统（SVN和CVS）

先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器

**缺点：必须联网**

## 1.2分布式版本控制系统（Git）

分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库

**优点：安全性要高，不需要联网**

# 2.Git的安装

从[https://git-for-windows.github.io](https://git-for-windows.github.io/)下载（网速慢的同学请移步[国内镜像](https://pan.baidu.com/s/1kU5OCOB" \l "list/path=%2Fpub%2Fgit)），然后按默认选项安装即可。安装完成后，在开始菜单里找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明Git安装成功！



安装完成后，还需要最后一步设置，在命令行输入：

$ git config --global user.name "Your Name"$ git config --global user.email "email@example.com"

**注意：**

git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

# 3.创建版本库

版本库又名仓库，英文名**repository**，你可以简单理解成一个目录，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可以“还原”。

**首先，选择一个合适的地方，创建一个空目录：**

$ mkdir learngit$ cd learngit$ pwd

/Users/michael/learngit

//pwd命令用于显示当前目录

**第二步，通过git init命令把这个目录变成Git可以管理的仓库：**

$ git init Initialized empty Git repository **in** /Users/michael/learngit/.git/

**第三步，把文件添加到版本库**

现在我们编写一个readme.txt文件，一定要放到learngit目录下（子目录也行），因为这是一个Git仓库，放到其他地方Git再厉害也找不到这个文件。

1. 用命令git add告诉Git，把文件添加到仓库：

$ git add readme.txt

1. 用命令git commit告诉Git，把文件提交到仓库：

$ git **commit** -m "wrote a readme file"

git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容。

**总结：**

初始化一个Git仓库，使用git init命令。

添加文件到Git仓库，分两步：

1. 使用命令git add <file>，注意，可反复多次使用，添加多个文件；
2. 使用命令git commit，完成；

# 4.时光机穿梭

我们已经成功地添加并提交了一个readme.txt文件，现在，是时候继续工作了，于是，我们继续修改readme.txt文件，改成如下内容：

Git **is** a distributed version control system. Git **is** free software.

现在，运行git status命令看看结果：

$ git status*# On branch master# Changes not staged for commit:# (use "git add <file>..." to update what will be committed)# (use "git checkout -- <file>..." to discard changes in working directory)## modified: readme.txt#* no changes added to commit (use "git add" **and**/**or** "git commit -a")

git status命令可以让我们时刻掌握仓库当前的状态，上面的命令告诉我们，readme.txt被修改过了，但还没有准备提交的修改。

虽然Git告诉我们readme.txt被修改了，但如果能看看具体修改了什么内容，所以，需要用git diff这个命令看看：

$ git diff readme.txt

git diff顾名思义就是查看difference，显示的格式正是Unix通用的diff格式，可以从上面的命令输出看到，我们在第一行添加了一个“distributed”单词。

知道了对readme.txt作了什么修改后，再把它提交到仓库就放心多了，提交修改和提交新文件是一样的两步，第一步是git add：

$ git add readme.txt

同样没有任何输出。在执行第二步git commit之前，我们再运行git status看看当前仓库的状态：

$ git status

git status告诉我们，将要被提交的修改包括readme.txt，下一步，就可以放心地提交了：

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

提交后，我们再用git status命令看看仓库的当前状态：

$ git status*# On branch master*nothing to commit (working directory clean)

Git告诉我们当前没有需要提交的修改，而且，工作目录是干净（working directory clean）的。

**小结：**

1. 要随时掌握工作区的状态，使用git status命令。
2. 如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

# 5.版本回退

1. 查看历史记录：git log
2. 回退到上一个版本：git reset --hard HEAD^

用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100

返回上一个版本

$ git reset --hard HEAD^

**注意：**

当返回上一个版本后，当前版本将消失，想再次查看当前版本，则需要知道当前版本的commit id

$ git reset --hard 3628164

3. 记录你的每一次命令：git reflog

**总结：**

1. HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令

git reset --hard commit\_id

2.穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。

3.要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

# 6.工作区和暂缓区

## 6.1工作区（Working Directory）

就是你在电脑里能看到的目录，比如我的learngit文件夹就是一个工作区

## 6.2暂缓区

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步：是用git add把文件添加进去，实际上就是把文件修改添加到暂存区stage；

第二步：是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前master分支。

一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是“干净”的：

$ git status*# On branch master*nothing to commit (working directory clean)

现在版本库变成了这样，暂存区就没有任何内容了：



**注意**：git commit 只能提交git add之后的

# 7.撤销修改

git checkout -- readme.txt意思是把readme.txt文件在**工作区**的修改全部撤销。

git reset HEAD file\_name是把**暂存区（add后）**的修改撤销掉（unstage），重新放回工作区.

git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。

**总结：**

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令

git checkout -- file

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区(add后)时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)一节，不过前提是没有推送到远程库。

# 8.删除文件

当你要删除文件的时候，可以采用命令：rm test.txt

这个时候（也就是说这个时候只执行了rm test.txt）有两种情况：

第一种情况：的确要把test.txt删掉，那么可以执行

$ git rm test.txt

$ git commit -m "remove test.txt"

然后文件就被删掉了

第二种情况:：错文件了，不应该删test.txt，注意这时只执行了rm test.txt，还没 有提交，所以可以执行git checkout test.txt将文件恢复。

**注意：**

并不是说执行完git commit -m "remove test.txt"后还能用checkout恢复，commit之后版本库里的文件也没了，自然没办法用checkout恢复，而是要用其他的办法。

# 9.远程仓库—GitHub

## 9.1获取本机的ssh\_key与GitHub链接

ssh-keygen -t rsa -C "youremail@example.com"

在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

## 9.2添加远程库

1. 在github上创建一个仓库，取名**learnGit**
2. 执行 git remote add origin git@github.com:849673404/**learnGi**t.git

$ git remote add origin git@github.com:849673404/**learnGi**t.git

**注意**：

添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

3. 执行 git push -u origin master 就可以把本地库的所有内容推送到远程库上

**注意**：

第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的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。以后提交只需执行 git push origin master

**总结：**

1. 要关联一个远程库，使用命令

git remote add **origin** git@github.com:849673404/**repo-name**.git

1. 使用命令git push -u **origin** master 第一次推送master分支的所有内容；
2. 可以使用命令git push **origin** master推送最新修改；

## 9.3从远程库克隆

1. 在github上创建一个仓库，取名**gitskills**
2. 执行 git clone git@github.com:849673404/**gitskills**.git

# 10.分支管理

## 10.1创建于合并分支

查看分支：git branch 查询结果前面带 \*，表示处于当前分支。

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

丢弃一个没有被合并过的分支（强行删除）：git branch -D <name>

**注意：**

1. git log --graph命令可以看到分支合并图
2. 合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

如：git merge --no-ff -m "merge with no-ff" dev

加上-m参数，把commit描述写进去

## 10.2Bug分支

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除。

当手头工作没有完成时，先把工作现场git stash一下，可以把当前工作现场“储藏”起来，然后去修复bug，修复后，再git stash pop或者git stash apply，回到工作现场。

git stash apply：恢复后，stash内容并不删除，你需要用git stash drop来删除；

git stash pop：恢复的同时把stash内容也删了；

git stash list：查看储藏的工作现场；

git stash apply **stash@{0}**：多次stash后，恢复指定的stash；

## 10.3Feature分支

软件开发中，总有无穷无尽的新的功能要不断添加进来。

添加一个新功能时，你肯定不希望因为一些实验性质的代码，把主分支搞乱了，所以，**每添加一个新功能，最好新建一个feature分支**，在上面开发，完成后，合并，最后，删除该feature分支。

## 10.4多人协作

git remote：可以查看远程库的信息，名称；

git remote -v ：显示更详细的信息；可以抓取和推送的远程仓库的地址；

git push origin **master**：将本地的**master**分支提交推送到远程库的master分支；

**多人协作的工作模式通常是这样：**

1. 首先，可以试图用git push origin <branch-name>推送自己的修改；
2. 如果推送失败，则是远程分支比你的本地更新，需要先用git pull试图合并；
3. 如果合并有冲突，则**解决冲突**，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功;

如果git pull提示“no tracking information”，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream branch-name origin/<branch-name>

**总结:**

1. 查看远程库信息，使用git remote -v；
2. 本地新建的分支如果不推送到远程，对其他人就是不可见的；
3. 从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；
4. 在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；
5. 建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；
6. 从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

# 11.标签管理

## 11.1创建标签

git tag <name>可以新建一个标签，默认为HEAD，也可以指定一个commit id；

如：$ git tag v1.0

$ git tag v0.9 6224937

git tag -a <tagname> -m "blablabla..."：可以指定标签信息；

git tag -s <tagname> -m "blablabla..."：可以用PGP签名标签；

git tag：可以查看所有标；

git show <tagname>：查看标签信息；

## 11.2操作标签

git push **origin** <tagname>：可以推送一个本地标签；

git push **origin** --tags：可以推送**全部**未推送过的本地标签；

git tag -d <tagname>：可以删除一个**本地标签**；

git push **origin** :refs/tags/<tagname>：可以删除一个**远程标签**。

# 12.GitHub和码云（gitee）

## 12.1GitHub

在GitHub上，可以任意Fork开源仓库；

自己拥有Fork后的仓库的读写权限（一定要从自己的账号下clone仓库，这样你才能推送修改）；

可以推送pull request给官方仓库来贡献代码；

## 12.1码云

在本地库上使用命令git remote add把它和码云的远程库关联：

$ git remote add origin git@gitee.com:849673404/**learngit**.git

之后，就可以正常地用git push和git pull推送了！

git remote -v：查看远程库信息

git remote rm **origin**：删除远程库origin

**注意：**

**github和gitee远程库不能同名**

**13.自定义Git**

**13.1忽略特殊文件**

有些时候，你必须把某些文件放到Git工作目录中，但又不能提交它们，比如保存了数据库密码的配置文件啦，等等，每次git status都会显示Untracked files...

**在Git工作区的根目录下创建一个特殊的.gitignore文件**，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。

不需要从头写.gitignore文件，GitHub已经为我们准备了各种配置文件，只需要组 合一下就可以使用了。所有配置文件可以直接在线浏览：<https://github.com/github/gitignore>

**忽略文件的原则是：**

1. 忽略操作系统自动生成的文件，比如缩略图等；
2. 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的.class文件；
3. 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

**强制添加忽略的文件**： git add -f <file-name>

如：$ git add -f App.class

**13.2配置别名**

**如：**

$ git config --global **alias**.st status st就表示status；

$ git config --global **alias**.co checkout co表示checkout； $ git config --global **alias**.ci commit ci表示commit； $ git config --global **alias**.br branch br表示branch；

$ git config --global **alias**.last 'log -1' last表示最后一次提交信息；

$ git config --global **alias**.unstage 'reset HEAD' unstage表示把暂存区的修改撤销掉；

**注意：**

配置Git的时候，加上--global是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用。

每个仓库的Git配置文件都放在.git/config文件中

**仓库的配置**

[core]

repositoryformatversion = 0

filemode = false

bare = false

logallrefupdates = true

symlinks = false

ignorecase = true

hideDotFiles = dotGitOnly

[remote "origin"]

url = git@github.com:849673404/JavaEE\_demo.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

pushurl = git@github.com:849673404/JavaEE\_demo.git

pushurl = git@gitee.com:luotf/JavaEE\_demo.git

[branch "master"]

remote = origin

merge = refs/heads/master

[user]

name = 849673404

email = 849673404@qq.com