**官网手册** [**https://git-scm.com/book/zh/v2**](https://git-scm.com/book/zh/v2)

**Git安装和配置**

<https://git-for-windows.github.io>

[国内镜像](https://pan.baidu.com/s/1kU5OCOB#list/path=%2Fpub%2Fgit)

下载安装后执行Git Bash，并进行全局配置：

git config --global user.name "Your Name"

git config --global user.email "email@example.com"

git config --global credential.helper cache

git config --global color.ui true

--global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。

在项目根目录下进行单独配置

git config user.name "Your Name"

git config user.email [email@xx.com](mailto:email@xx.com)

git config --list 查看当前配置

在当前项目下，查看的是全局配置+当前项目的配置, 使用的时候会优先使用当前项目的配置

**建立Git版本库**

**~ cd /e 切换到E盘**

**~ mkdir git\_repo 创建版本库目录**

**~ cd git\_repo 进入版本库目录**

**~ git init 初始化版本库管理**

**注意：**所有的版本控制系统，其实只能跟踪文本文件的改动，而图片、视频这些二进制文件，没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。

不幸的是，Microsoft的Word格式是二进制格式，因此，版本控制系统是没法跟踪Word文件的改动的。（Notepad++的默认编码设置为UTF-8 without BOM）

**添加文件到版本库**

先新建文本文件readme.txt，编辑完内容并保存到e:/git\_repo/目录下：

**~ git add readme.txt** 把文件添加到仓库

**~ git commit -m "wrote a readme file"** 把文件提交到仓库

-m参数表示本次提交的说明，最好是有意义的

注意：因为commit可以一次提交很多文件，所以你可以多次add不同的文件

**修改文件到版本库【查看状态和查看修改内容】**

修改readme.txt文件并保存后:

**~ git status** 查看当前仓库工作区状态

**~ git diff** 查看工作区【默认】和暂存区的区别

**~ git diff 版本号** 查看工作区【默认】和分支的区别

**~ git diff 版本号 版本号** 查看分支和分支的区别

**~ git diff --cached [版本号]** 查看暂存区和分支【默认为HEAD】的区别

提交修改只需两步（同提交新文件一样）：

**~ git add readme.txt** 把文件添加到仓库

**~ git commit -m "add distributed"** 把文件提交到仓库

**注意：**要随时掌握工作区的状态，使用git status命令，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，git diff HEAD -- <file>中的HEAD表示版本库最新版，也可指定某个版本号与工作区<file>进行比较。

**版本回退**

**~ git log** 查看提交的历史记录（从最近到最远的提交日志）

**~ git log**查看版本历史记录

**--pretty=oneline**用列表单行逐行显示

**--graph** 可以看到分支合并图

**--abbrev-commit** 压缩版本号显示

**~ git reflog** 查看命令的历史记录（常用于重返未来版本）

这里，类似3628164...882e1e0的是**commit id（版本号）**，是一个SHA1计算出来的一个非常大的数字，用十六进制表示

**~ git reset --hard HEAD^** 回退到上一个版本

**~ git reset --hard 版本号** 回退到指定的某个版本，其实就是将指定的版本号文件内容写回到工作区，并清空暂存区

**注意**：

① 在Git中，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^^，往上100个版本写成HEAD~100；

② 若想回退到指定某个版本号时，版本号只需写前7位（最少前4位）即可，Git会自动去找。

**说明**：Git的版本回退速度非常快，因为Git在内部有个指向当前版本的HEAD指针，当你回退版本的时候，Git仅仅是改变HEAD指向，并顺便把工作区的文件更新。

**工作区、暂存区和分支**

工作区：git\_repo目录；

版本库：git\_repo/.git目录；

在版本库里面有一个称为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**HEAD指针**。



往Git版本库添加和修改文件的**操作原理**如下：

第一步：用git add文件，实际上是把文件添加和修改的内容全部保存到暂存区

第二步：用git commit提交，实际上是把暂存区的所有内容提交到当前分支

即：把要提交的所有文件添加和修改先全部add到暂存区，再一次性将暂存区所有内容commit到当前分支上，可见，**Git管理的是修改（暂存区内容），而不是文件**。

**撤销修改**

**①** 仅修改了工作区的文件（注意该文件必须在暂存区或版本库中存在）时：

**这里要理解：错误发生仅发生在工作区！**

**~ git checkout -- readme.txt** 表示用暂存区或版本库内容还原工作区的修改

**注意**：这里仅能将工作区文件内容回退到最新的暂存区或最新的版本库的内容，优先选用【暂存区】的内容，【版本库】的内容次之；

另外，命令中的 -- 很重要，没有 -- ，就变成了"切换到另一个分支"的命令。

**②** 当修改了工作区的文件并已经添加到了暂存区时（注意版本库中必须存在该文件）：

**这里要理解：错误已经发生到暂存区了！！**

**~ git reset [--mixed] [版本号] readme.txt** 表示将readme.txt文件指定的[版本号]内容拿出来放到暂存区（[版本号]可省略，默认为HEAD当前最新版本）。

再用 **~ git checkout -- readme.txt** 将暂存区的内容写回到工作区。

**注意**：如果暂存区仅有1个文件并且有错误，可以通过回退版本一步到位，如果暂存区有10个文件，仅有1个有错误，回退就不划算了，因为剩下9个正确的更新也被回退了。

**③** 如果错误的修改已经从暂存区提交到了版本库，可以回退版本，不过前提是还没有推送到远程库。

**这里要理解：错误已经发生到版本库了！！！**

如果提交后的新版本中有10个文件，只有1个文件readme.txt有错误，回退版本其实是不划算的，因为剩下9个正确的也被回退到工作区了；这时可以先用 **~ git reset HEAD^ readme.txt**将最新版的上一版的那个readme.txt文件放入暂存区，再用 **~ git checkout -- readme.txt**将暂存区内容写回到工作区，用于修改错误。

**删除文件**

**~ rm test.txt** 表示仅从工作区删除文件(相当于手动删除)

**~ git rm test.txt** 表示从工作区删除文件并提交到暂存区，还需要使用 **~ git commit**提交到版本库才能从版本库中删除

当要恢复删除的文件时:（注意版本库中必须存在该文件）

若文件已从工作区删除或删除已提交到暂存区，则先使用 **~ git reset test.txt**，再使用 **~ git checkout -- test.txt** 来恢复；若删除已经提交到版本库，同样可用上述方法恢复当前版本的上一个版本的test.txt文件，也可以直接回退上一个版本来恢复；原理参考【撤销修改】章节。

**注意**：若版本库中都不存在该文件，则删除的文件是不能恢复的

**远程仓库**

<https://github.com> : 一个提供Git仓库托管服务的网站

注册一个Github账号，就可免费获得Git远程仓库。

本地：Git仓库<——————**SSH加密传输**——————>远程：GitHub仓库

**一、创建SSH Key密钥对**

打开Shell（windows下打开Git Bash）：

**~ ssh-keygen -t rsa -C "crackhu@163.com"**

然后一路回车，使用默认值即可。然后在**用户主目录**下找到：

**.ssh目录 --> id\_rsa私钥文件 + id\_rsa.pub公钥文件**

**注意**：windows下用户主目录为：C:/Users/Administrator/

**二、实现本地与远程仓库的匹配识别**

登录GitHub，右上角选择“Settings”，左侧列表选择“SSH Keys”，右侧区域单击“New SSH Key”按钮，然后将id\_rsa.pub的内容复制粘贴到Key文本区域，标题随意取，然后单击“Add SSH Key”按钮即可。

**注意**：可将多台电脑的key都添加到同一个GitHub仓库上,实现相互推送

**三、将本地库上传到远程库**

在GitHub账户laker1上创建一个对应的远程仓库git\_repo，

然后在本地git\_repo仓库目录下：**添加**对应的**远程仓库**的**地址（默认名称origin）**，建立映射关系，可添加多个远程映射到本地:

**~ git remote add origin git@github.com:laker1/git\_repo**

**~ git remote add private git@github.com:laker1/git\_private**

**【执行后此时配置文件应该会有分支映射记录 remote=origin 】**

接下来在本地git\_repo仓库目录下：将**本地**仓库master分支的**内容推送**到**远程**仓库origin的master分支上：

**~ git push [-u] origin master //** 第1次推送时，加**-u**可建立分支关联

**注意：**origin是远程仓库名称，是Git默认的叫法

~ git ls-remote 显示所有远程库的所有分支引用完整列表

~ git remote -v 可以查看与本地仓库有映射关联的所有远程仓库的详细信息

~ git remote add origin git@github.com:lake/rep可以添加远程仓库origin

~ git fetch origin 将远程仓库origin的master分支的最新内容更新到本地master分支中

~ git fetch --all 将所有的远程仓库的master分支的最新内容更新到本地

git pull == git fetch + git merge

**说明：**git fetch 命令从服务器上抓取数据，并不会修改工作目录中的内容，git pull 会查找当前分支所跟踪的服务器与分支，然后从服务器上抓取数据然后尝试合并入那个远程分支到本地

~ git checkout --track origin/dev 将远程仓库origin的dev分支的最新内容更新到本地dev分支中

~ git checkout -b localdev origin/dev 将远程仓库origin的dev分支最新内容更新到本地localdev分支中【本地分支与远程分支可以名称不相同】

在**跟踪分支**里输入 **~ git push**，Git 会自行推断应该向哪个服务器的哪个分支推送数据。同样，在这些分支里运行 **~ git pull** 会获取所有远程索引，并把它们的数据都合并到本地分支中来。

~ git push origin dev 表示推送dev分支到远程仓库origin上

【这里被简化了，应该是 ~ git push origin dev:dev，另外，远程端可以取不同名字，比如：~ git push origin dev:dev-origin】

~ git push origin :dev 表示删除远程仓库origin中的dev分支【相当于推送本地的空分支替换远程的dev分支】（Git版本1）

~ git push origin --delete dev 表示删除远程仓库origin中的dev分支【只是从服务器上移除这个指针】。Git 服务器通常会保留数据一段时间直到垃圾回收运行，所以如果不小心删除掉了，通常是很容易恢复的。(Git 版本2)

Git使用SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入yes回车即可。

**四、从远程库克隆到本地库（仅克隆master主分支）**

在GitHub账户laker1上创建一个对应的远程仓库gitskills，

【建议勾选Initialize this repository with a README】

接下来在本地待放置克隆库的目录下：将**远程库克隆到本地**指定目录中

**~ git clone git@github.com:laker1/gitskills.git**

**注意**：Git支持多种协议，默认使用SSH，也可使用https协议，对应写法为：

**~ git clone https://github.com/laker1/gitskills.git**

使用https除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令

**分支管理**

**一、创建与删除分支**

主分支，即默认master分支线，master指针指向主分支当前最新的提交；

新分支，即新建 dev分支线，dev指针默认指向与master指针当前指向相同的提交；

**HEAD默认指向master指针，创建dev分支后，切换指向dev指针；**



**~ git branch** 表示**查看**目前所有分支以及当前所处分支（\*）

**~ git branch -vv** 表示**查看**目前所有分支以及设置的所有**跟踪**分支

**~ git branch dev**表示**创建**新dev分支

**~ git branch -d dev** 表示**删除**dev分支**（已合并）**

**~ git branch -D dev** 表示强行**删除**dev分支**（未合并，多用于Feature分支）**

**~ git branch -u origin/dev** 表示设置**当前分支**为**跟踪分支**并跟踪远程dev分支

**~ git branch --set-upstream-to dev origin/dev 同上**

**~ git checkout dev** 表示**切换**到dev分支

**~ git checkout -b dev** **[\*branch]** 表示在本地**创建并切换**到dev分支，内容与指定分支【默认为创建之前的当前分支】指向内容相同

等价于：

**~ git branch dev** 【在本地创建dev分支，与当前所处分支指向内容相同】

**~ git checkout dev** 【在本地切换到dev分支】

**二、合并分支**

创建分支dev，修改并提交版本后，切换回主分支：

**1、~ git merge dev** 表示将本地dev分支**合并**到当前所处分支上（将当前分支指针指向与dev指针当前指向相同的提交）【默认"快进模式"】

当前分支的commit-id修改为与dev最新commit-id相同

~ git merge origin/dev --allow-unrelated-histories

表示远程dev分支合并到本地当前所处分支上，参数--allow-unrelated-histories表示要把两个不同的项目合并

**2、~ git merge --no-ff -m "merge with no-ff" dev**表示将本地dev分支**合并**到当前所处分支上【"普通模式"】，但是会产生一个新提交commit-id

**~ git merge --no-ff origin/dev**表示远程dev分支合并到本地当前所处分支上

**注意：**合并操作是取两个分支的并集，因此可能会产生冲突，需手动解决。

**另外：**不同分支共用同一个暂存区，因此合并之前要记得完成提交。

**分支策略：**

首先，master分支应该是非常稳定的，仅用来发布新版本，平时不能在上面干活；

团队成员都在各自的dev分支上干活，发布时把dev分支合并到master分支上即可。



合并分支--快进模式 合并分支--冲突 合并分支--普通模式

**三、Bug分支**

1、在dev分支上，做出修改后：执行 **~ git stash** 表示**保存现场**

2、切换回主分支 **~ git checkout master**

创建临时Bug分支 **~ git checkout -b issue-101**修复bug并提交版本

3、再切换回主分支 **~ git checkout master**

用普通模式合并bug分支，并删除bug分支

4、切换回dev分支：执行 **~ git stash list**表示**查看所有现场**

**~ git stash pop** **stash@{0}** 表示**恢复并删除**现场

相当于 **~ git stash apply stash@{0}** **恢复**现场

**~ git stash drop stash@{0}** **删除**现场

**四、多人协作dev分支**

1、从本地推送分支，使用 **~ git push origin dev**；

2、如果推送失败，远程提示未更新，先用 **~ git pull**抓取远程的新提交；

3、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，用下面的命令建立关联：

**~ git branch --set-upstream-to dev origin/dev**

4、再次使用 **~ git pull**抓取远程的新提交，

5、如果 ~ git pull合并提示有冲突，则解决冲突，并在本地commit提交；

6、再次使用 **~ git push origin dev** 推送本地分支

**标签管理**

标签是版本库的一个快照，是指向某个commit的指针。

**一、创建与查看标签**

**~ git tag v1.0** 表示给当前分支的当前版本打一个标签

**~ git tag v1.0 [版本号]** 表示给指定版本号打标签

**~ git tag -a v1.0 -m "desp" [版本号]** 表示给指定版本号打标签(带描述)

**~ git** **tag -s v0.2 -m "desp" [版本号]** 表示用私钥给指定版本号打标签

**~ git tag** 查看所有分支的所有标签名

**~ git show v1.0** 表示查看标签名为v1.0的详细信息

**二、标签的远程推送与删除**

**~ git push origin v1.0** 表示推送本地v1.0标签到远程origin仓库

**~ git push origin --tags** 表示一次性推送全部尚未推送到远程的本地标签

**~ git tag -d v1.0** 表示删除本地标签v1.0

**~ git push origin :refs/tags/v1.0** 表示从远程仓库origin删除标签v1.0

**使用GitHub**

1、参与开源项目（以bootstrap为例）

访问项目主页<https://github.com/twbs/bootstrap>，点“Fork”就在自己的账号下克隆了一个bootstrap仓库，然后在本地目录下克隆自己账号仓库下的bootstrap仓库（**~ git clone git@github.com:laker1/bootstrap.git**）。

2、本地修改后提交并推送到自己账号下的远程仓库，发起pull request待对方接受。



**自定义Git**

**1、自动忽略特殊文件：**

在Git工作区的根目录下创建一个特殊的 .gitignore文件，把要忽略的文件名填进去即可，最后把 .gitignore也提交到Git版本库即可。

**忽略文件的原则**：

* 忽略操作系统自动生成的文件，比如缩略图等；
* 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；
* 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。

验证忽略效果：~ git add test.class（发现添加不了）

可强制添加到暂存区：~ git add -f test.class（一旦加入版本下次就不必再强制）

检查验证规则：~ git check-ignore -v app.class（提示哪条忽略规则生效）

**策略：先\*忽略全部, 之后再!设定不被忽略的内容**

**2、配置别名**

~ git config --global alias.st status 表示git st == git status

~ git config --global alias.co checkout

~ git config --global alias.ci commit

~ git config --global alias.br branch

~ git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

* 每个仓库的Git配置文件都放在.git/config文件中
* 当前用户的Git配置文件放在用户主目录下的一个隐藏文件.gitconfig中

**3、搭建Git服务器**

**加密工具GPG入门教程**

教程：<http://www.ruanyifeng.com/blog/2013/07/gpg.html>

下载：<https://www.gnupg.org/download/index.en.html>

-> <https://www.gpg4win.org/download.html>

**~ gpg --gen-key** 生成密钥

用户ID：mapleboy <crackhu@163.com>

053DB3DA 这是"用户ID"的Hash字符串，可以用来替代"用户ID"

**~ gpg --gen-revoke [用户ID或Hash字符串]** 生成一张"撤销证书"，以备以后密钥作废时，可以请求外部的公钥服务器撤销你的公钥

iQEfBCABAgAJBQJZWky3Ah0AAAoJENj12qwFPbPaycQIAI2Rl7j9HwrsT9Of67Ys

cV24u3+hm2VB3BoPmXiWbbmUHW2N/xC/N68343wFODK9Di9ZzoS+7STpxtyyBvm6

dNOdkBh10hF8LPx8/NEtXxDRUMicL3xbIyHOMVq2PGb6lZB6YjBCe12TJS3sA1C+

hPxPECimhpfAkGLYu5JB59PTmlWCyVK8ZT39EuMX9YIKmTKrhsZMfqUH4bBriFsC

Z+jFABn+e5ChgXHLbhRPpIllQHI0fdw4+wdydxZNeiAReWk31eeSVRx5R5ewcqtq

qHY8o8umeFecN+M0UDAOr/EGwS9M60pUOSAzHGQDIKT0DmAFg8JQlIP8FjWoSQ1Y

eH0=

=o0iR

**~ gpg --list-keys** 列出系统中已有的密钥