# 安装Git

下载地址：<https://git-scm.com/download>；

默认安装时，Git会自动安装两种客户端：命令行客户端“Git Bash”和图形化客户端“Git GUI”；

图形化客户端：gitk（默认安装的是gitk）、SourceTree、GitHubDesktop、GitKraken、GitUp等；

Git中默认使用的编辑器是vim。

## Ubuntu中安装

类Unix系统中，Git要使用curl，zlib，openssl，expat，libiconv 等库的代码才能正常使用。

|  |  |
| --- | --- |
| **命令安装** | $ sudo apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev  $ apt-get install git |
| **源码安装** | 在官网下载源码（<https://git-scm.com/download>）；  $ sudo apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev  $ tar -zxf git-2.9.5.tar.gz  $ cd git-2.9.5  $ make prefix=/usr/local all  $ sudo make prefix=/usr/local install |

## Windows中安装

安装包下载地址：<https://gitforwindows.org/>，默认安装即可。

# git指令

|  |  |
| --- | --- |
|  | |
| **命令** | git --version |
| **解释** | 查看git版本 |

## ssh

|  |  |
| --- | --- |
|  | |
| **命令** | ssh-keygen –t rsa –C “burgesskzg@yeah.net” |
| **解释** | 利用该命令生成ssh密钥对。 |
| **备注** | 默认情况下，密钥对生成位置为：C:\Users\burgess\.ssh；  打开github的设置，可将” id\_rsa.pub “中的公钥添加进去使用；  更多关于ssh的操作可参考博客：https://www.zsythink.net/archives/2375 |
|  | |
| **命令** | ssh –T git@github.com |
| **解释** | 验证本终端的ssh是否正确配置 |

## git config

|  |  |
| --- | --- |
|  | |
| **命令** | git config --global user.name "burgesskzg" |
| **解释** | 配置用户名 |
|  | |
| **命令** | git config --global user.email burgesskzg@yeah.net |
| **解释** | 配置邮箱 |
|  | |
| **命令** | git config --global color.ui true |
| **命令** | git config --global color.ui auto |
| **解释** | 配置色彩显示 |
|  | |
| **命令** | git config --global core.autocrlf false |
| **解释** | 禁用自动转换换行符(LF转CRLF) |
|  | |
| **命令** | git config --global core.editor vim |
| **解释** | 配置编辑器 |
|  | |
| **命令** | git config --global merge.tool vimdiff |
| **解释** | 配置差异分析器 |
|  | |
| **命令** | git config --list |
| **解释** | 查看配置信息 |
|  | |
| **命令** | git config 变量名 |
| **命令** | eg：git config user.name |
| **解释** | 查看特定环境变量的值 |
|  | |
| **备注** | 系统中存在三种不同的配置文件，“/etc/gitconfig “针对所有用户，配置时使用”--systm“；”~/.gitconfig “针对当前用户，配置时使用”--global“；” 仓库目录的.git/config “针对当前仓库，配置时使用”--local“。低层级别配置会覆盖上层级别配置，针对同名环境变量，优先级（优先使用）.git/config > ~/.gitconfig > /etc/gitconfig。 |

## git init

|  |  |
| --- | --- |
|  | |
| **命令** | git init [path] |
| **解释** | 将当前目录或者指定目录初始化为一个仓库 |

## git clone

|  |  |
| --- | --- |
|  | |
| **命令** | git clone <repo> <directory> |
| **解释** | 将本地仓库repo拷贝到目录directory |
|  | |
| **命令** | git clone URL |
| **命令** | git clone URL NewRepoName |
| **解释** | 将服务器上的仓库克隆到本目录下，可以指定新项目名 |
|  | |
| **命令** | git clone git@URL |
| **解释** | 使用SSH协议克隆服务器仓库到本地 |
|  | |
| **命令** | git clone git://URL |
| **解释** | 使用GIT协议克隆服务器仓库到本地 |
|  | |
| **命令** | git clone <https://URL> |
| **解释** | 使用https协议克隆服务器仓库到本地 |

## git add

|  |  |
| --- | --- |
|  | |
| **命令** | git add |
| **解释** | 将当前目录或者指定目录初始化为一个仓库 |
|  | |
| **命令** | git add **.** |
| **解释** | 将工作区的新文件和修改的文件提交到缓存区（新文件和修改） |
|  | |
| **命令** | git add –u |
| **解释** | 监控已经被跟踪的文件（修改和删除） |
|  | |
| **命令** | git add -A |
| **解释** | 监控所有工作区的文件（新文件，修改和删除） |
|  | |
| **备注** | 新文件建立后，利用该命令跟踪；  旧文件修改后，利用该命令准备文件快照。 |

## git commit

|  |  |
| --- | --- |
|  | |
| **命令** | git commit –m “注解” |
| **解释** | 将暂存区的目录树写到版本库（对象库）中，做一次提交 |
|  | |
| **命令** | git commit –am “注解” |
| **解释** | 略过”git add“，直接提交 |

## git status

|  |  |
| --- | --- |
|  | |
| **命令** | git status |
| **解释** | 查看上次提交之后是否有修改（新文件，修改或删除） |
|  | |
| **命令** | git status –s |
| **解释** | 简短输出 |

## git diff

|  |  |
| --- | --- |
|  | |
| **命令** | git diff |
| **解释** | 查看缓存区和工作区的区别 |
|  | |
| **命令** | git diff --FileName |
| **解释** | 查看文件FileName在缓存区和工作区的区别 |
|  | |
| **命令** | git diff --cached |
| **解释** | 查看缓存区和最新提交之间的差异 |
|  | |
| **命令** | git diff HEAD |
| **命令** | git diff 最新提交哈希值 |
| **解释** | 查看工作区和最新提交之间的差异 |
|  | |
| **命令** | git diff CommitAHash CommitBHash |
| **命令** | git diff Commit**A**Hash..Commit**B**Hash |
| **解释** | 比较CommitAHash和CommitBHash两个提交的区别 |
|  | |
| **命令** | git diff --stat |
| **解释** | 显示摘要而非整个diff |
|  | |
| **备注** | HEAD <==>HEAD~0  HEAD ~<==>HEAD~1  HEAD ~~<==>HEAD~2  HEAD ~~~<==>HEAD~3  HEAD~n代表倒数第n+1个提交，在命令中可以使用。 |

## git show

|  |  |
| --- | --- |
|  | |
| **命令** | git show v1.0 |
| **解释** | 查看版本号为v1.0的提交修改的内容（v1.0通过标签实现） |
|  | |
| **备注** | 可查看blob对象，tree对象，commit对象和tag标签，使用方式是“git show对象Hash“ |

## git log

|  |  |
| --- | --- |
|  | |
| **命令** | git log |
| **解释** | 查看提交历史 |
|  | |
| **命令** | git log --oneline |
| **命令** | git log --oneline -5 |
| **解释** | 查看提交记录的简洁版本，可以指定显示行数 |
|  | |
| **命令** | git log --name-status |
| **解释** | 看哪些文件改变了 |
|  | |
| **命令** | git log --graph |
| **解释** | 以拓扑图的方式展示提交历史(包括分支和合并) |
|  | |
| **命令** | git log --author=burgesskzg |
| **解释** | 查找指定用户的提交日志 |
|  | |
| **命令** | git log --no-merges |
| **解释** |  |
|  | |
| **命令** | git log --before={1.weeks.ago} |
| **命令** | git log --after={2020-02-08} |
| **命令** | git log --until |
| **解释** | 可以指定查看某段时间的提交记录 |
|  | |
| **命令** | git log --reverse |
| **解释** | 逆向显示所有日志 |

## git reflog

|  |  |
| --- | --- |
|  | |
| **命令** | git reflog |
| **解释** | 当通过git reset命令回到以前提交，想要再回到当前提交的后面的提交，通过git log看不到哈希值，要使用该命令 |

## git ls-files

|  |  |
| --- | --- |
|  | |
| **命令** | git ls-files -s |
| **解释** | 查看暂存区文件 |

## git reset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
| **命令** | git reset HEAD | | | |
| **命令** | git reset--mixed HEAD | | | |
| **解释** | 暂存区（的目录树）被当前分支的最新提交替换，不影响工作区 | | | |
| **备注** | git reset 默认选项就是--mixed | | | |
|  | | | | |
| **命令** | git reset HEAD --FileName | | | |
| **解释** | 暂存区的文件FileName被当前分支的最新提交中的对应文件替换，不影响工作区 | | | |
|  | | | | |
| **命令** | git reset --soft CommitHash | | | |
| **解释** | 将HEAD指向CommitHash对应的提交，是分离头吗？与git checkout CommitHash制造分离头的区别？ | | | |
|  | | | | |
| **命令** | git reset --hard CommitHash | | | |
| **解释** | 回到某个版本，git的重要功能，“--hard“选项用CommitHash对应的提交替换工作区和暂存区 | | | |
| **备注** | 可能会用到” git reflog”命令。回到以前版本，又想回到之后的版本，需要这个命令查看之后版本的哈希值 | | | |
|  | | | | |
| **备注** |  | 工作区 | 索引 | HEAD |
| --soft | 否 | 否 | 是 |
| --mixed | 否 | 是 | 是 |
| --hard | 是 | 是 | 是 |

## git checkout

|  |  |
| --- | --- |
|  | |
| **命令** | git checkout |
| **解释** | 用暂存区全部文件替换工作区文件，会清除工作区未添加到暂存区的改动，谨慎使用。 |
|  | |
| **命令** | git checkout -- <file> |
| **解释** | 用暂存区指定的文件替换工作区的文件，危险操作，谨慎使用。 |
|  | |
| **命令** | git checkout HEAD . |
| **解释** | 用HEAD指向的分支的最新提交中的全部文件替换暂存区和工作区中的文件，危险操作，谨慎使用。 |
|  | |
| **命令** | git checkout HEAD <file> |
| **解释** | 用HEAD指向的分支的最新提交中的指定文件替换暂存区和工作区中的对应文件，危险操作，谨慎使用。 |
|  | |
| **命令** | git checkout CommitHash |
| **解释** | 出现**分离头** |
|  | |
| **命令** | git checkout BranchName |
| **解释** | 切换分支，Git用该分支最后提交的快照替换你的工作目录 |
|  | |
| **命令** | git checkout -b BranchName |
| **解释** | 创建新分支并立即切换到该分支下 |

## git restore

|  |  |
| --- | --- |
|  | |
| **命令** | git restore <file> |
| **解释** | 放弃工作区中的修改 |

## git branch

|  |  |
| --- | --- |
|  | |
| **命令** | git branch |
| **解释** | 列出分支 |
|  | |
| **命令** | git branch -v |
| **解释** | 查看分支的详细信息 |
|  | |
| **命令** | git branch -vv |
| **解释** | 查看本地分支和远程分支的对应关系 |
|  | |
| **命令** | git branch BranchName |
| **解释** | 创建分支 |
|  | |
| **命令** | git branch –d BranchName |
| **解释** | 删除分支,如果分支没有合并，无法删除 |
|  | |
| **命令** | git branch –D BranchName |
| **解释** | 强制删除 |
|  | |
| **命令** | git branch new --set-upstream-to=origin/new |
| **命令** | git branch new -u origin/new |
| **解释** | 设置上游分支 |

## git merge

|  |  |
| --- | --- |
|  | |
| **命令** | git merge |
| **解释** | 合并分支，可多次合并。合并之后可直接删除被并入的分支。合并时如果发生冲突，需手动修改发生冲突的文件，然后调用“git add”和”git commit“告诉git冲突已经解决。 |
|  | |
| **命令** | git merge --no-ff |
| **解释** | 不使用fast-forward模式的合并（即使满足） |
|  | |
| **命令** | git merge --ff-only |
| **解释** | 只有满足fast-forward才执行，否则终止 |
|  | |
| **命令** | git merge branchName --no-ff –m “注解” |
| **解释** | 不使用fast-forward模式，同时添加注解 |
|  | |
| **命令** | git merge --no-commit |
| **解释** | 合并的时候如果没有冲突也需要手动提交，类似有冲突之后的执行步骤 |
|  | |
| **命令** | git merge --no-edit |
| **解释** | 合并的时候使用默认提交，不用弹出vim写注解 |
|  | |
| **命令** | git merge [alias]/[branch] |
| **解释** | 将远程服务器的更新合并到当前分支 |

## git rm

|  |  |
| --- | --- |
|  | |
| **命令** | git rm <file> |
| **解释** | 缓冲区没有该文件时这么删除 |
|  | |
| **命令** | git rm --cached <file> |
| **解释** | 只删除缓存区，不影响工作区 |
|  | |
| **命令** | git rm –r \* |
| **解释** | 递归删除目录下的所有文件和子目录； |
|  | |
| **命令** | git rm –f <file> |
| **解释** | 缓存区有该文件时这么删除 |

## git mv

|  |  |
| --- | --- |
|  | |
| **命令** | git mv SrcName DescName/PathName |
| **解释** | 移动或重命名一个文件、目录、软连接 |

## git remote

|  |  |
| --- | --- |
|  | |
| **命令** | git remote |
| **解释** | 查看当前仓库与哪些远程仓库关联 |
|  | |
| **命令** | git remote add [远程仓库别名] [url] |
| **解释** | 为本地仓库添加远程仓库 |
|  | |
| **命令** | git remote -v |
| **解释** | 显示每个远程仓库别名的实际链接地址 |
|  | |
| **命令** | git remote rm [别名] |
| **解释** | 删除远程仓库和本地仓库的关联 |

## git fetch

|  |  |
| --- | --- |
|  | |
| **命令** | git fetch |
| **解释** | 从远程仓库下载新分支与数据，之后需要执行合并git merge |
|  | |
| **命令** | git fetch [alias] |
| **解释** | 获取远程仓库有本地仓库没有的数据 |

## git pull

|  |  |
| --- | --- |
|  | |
| **命令** | git pull |
| **解释** | 更新本地仓库至最新改动(相当于git fetch和git merge一起操作) |

## git push

|  |  |
| --- | --- |
|  | |
| **命令** | git push origin master:master |
| **解释** | origin代表远程仓库，左master代表本地分支，右master代表远程分支 |
|  | |
| **命令** | git push origin master:newbranch |
| **解释** | 可以使用这个命令在远程仓库中新建一个newbranch分支 |
|  | |
| **命令** | git push origin new:new |
| **解释** | 推送本地分支到远程 |
| **备注** | ”git branch new --set-upstream-to=origin/new“或“git branch new -u origin/new “设置上游分支 |
|  | |
| **命令** | git push --set-upstream origin new:new |
| **解释** | 推送本地分支的同时设置本地分支的上游分支(远程分支) |
|  | |
| **命令** | git push origin --delete BranchName |
| **解释** | 删除远程仓库的分支 |

## git tag

|  |  |
| --- | --- |
|  | |
| **命令** | git tag |
| **解释** | 查看所有标签 |
|  | |
| **命令** | git tag –a vx.x |
| **解释** | 给当前分支的最新提交创建一个带有注解的标签 |
|  | |
| **命令** | git tag –a vx.x CommitHash |
| **解释** | 给指定的提交添加标签 |
|  | |
| **命令** | git tag -a <tagname> -m "注解 " |
| **解释** | 同时指定标签和注解 |
|  | |
| **命令** | it tag -s <tagname> -m "注解 " |
| **解释** | PGP签名标签 |
|  | |
| **命令** | git tag –d xxx |
| **解释** | 删除标签 |
|  | |
| **备注** | “git log --decorate”可以查看提交的标签 |

## git stash

|  |  |
| --- | --- |
|  | |
| **命令** | git stash |
| **解释** | 临时存储所有修改后的跟踪文件到堆栈中 |
|  | |
| **命令** | git stash save “注解” |
| **解释** | 临时存储所有修改后的跟踪文件到堆栈中,并添加注解 |
|  | |
| **命令** | git stash pop |
| **解释** | 弹出到工作区并删除 |
|  | |
| **命令** | git stash list |
| **解释** | 列出堆栈内容 |
|  | |
| **命令** | git stash drop stash@{n} |
| **解释** | 丢弃 |
|  | |
| **命令** | git stash clear |
| **解释** | 清空所有堆栈 |

## git rev-parse

|  |  |
| --- | --- |
|  | |
| **命令** | git rev-parse ShortHash/HEAD~n/BranchName/TagName |
| **解释** | 获取全部长度哈希值 |

## git cat-file

|  |  |
| --- | --- |
|  | |
| **命令** | git cat-file -t ShortHash |
| **解释** | 查看某个哈希值对应的git对象的类型 |
| **备注** | “-p”选项可查看对象的内容 |

## gitk

|  |  |
| --- | --- |
|  | |
| **命令** | gitk |
| **解释** | 打开图形界面gitk |

## gitignore

|  |  |
| --- | --- |
|  | |
| **操作** | * 在项目（仓库）根路径下新建一个.gitignore文件； * 编辑.gitignore文件，添加一些相对路径或通配符来避免文件提交； |
| **备注** | * 每一行指定一个忽略规则 * 只能忽略那些原来没有被track的文件，如果某些文件已经被纳入了版本管理中，则修改.gitignore是无效的 * # 开头的文件标识注释 * ! 开头的模式标识否定，该文件将会再次被包含，如果排除了该文件的父级目录，则使用 ! 也不会再次被包含； * / 结束的模式只匹配文件夹以及在该文件夹路径下的内容，但是不匹配该文件； * / 开始的模式匹配项目跟目录； * 如果一个模式不包含斜杠，则它匹配相对于当前 .gitignore 文件路径的内容，如果该模式不在 .gitignore 文件中，则相对于项目根目录； * \*\* 匹配多级目录，可在开始，中间，结束； * ? 通用匹配单个字符； * [] 通用匹配单个字符列表； * \* 通用匹配多个字符； |
| **示例** | * bin/: 忽略当前路径下的bin文件夹，该文件夹下的所有内容都会被忽略，不忽略 bin 文件 * /bin: 忽略根目录下的bin文件 * /\*.c: 忽略 cat.c，不忽略 build/cat.c * debug/\*.obj: 忽略 debug/io.obj，不忽略 debug/common/io.obj 和 tools/debug/io.obj * \*\*/foo: 忽略/foo, a/foo, a/b/foo等 * a/\*\*/b: 忽略a/b, a/x/b, a/x/y/b等 * !/bin/run.sh: 不忽略 bin 目录下的 run.sh 文件 * \*.log: 忽略所有 .log 文件 * config.php: 忽略当前路径的 config.php 文件 |
| **示例** | 在某个目录下添加.gitignore文件，内容为\*和!.gitignore可以实现忽略所有文件但保留目录 |
| **疑问** | 有了.gitignore文件，还能用git add –A来一次提交所有修改吗？也就是说.gitignore会不会被提交？ |

# github

## 功能

* git代码仓库托管
* 在线文件编辑器
* 协作图谱（报表）
* 代码片段分享（Gist）
* 基本Web管理
* 订阅
* 讨论组
* 文本渲染

## 开发者选项

* 托管项目，star喜欢的项目，fork并pull request项目；
* 为项目建立静态主页（gh-pages分支）；
* 通过特殊命名的仓库(http://username.github.io)来建立个人静态网站（GitHub Pages）；
* Vim vundle用户重度依赖Github来管理插件；
* 可作简历（<http://resume.github.io/>）；
* 在线编译和测试社区Travis CI依赖Github；
* 用Gists管理代码片段，同时可以内嵌到网页；
* 托管编程环境，在Github上搜索dotfiles学习怎么管理vim, git,tmux等配置文件，一般可以通过submodule管理插件；
* 为项目建立wiki,github/gollum 是其backend。可用markdown写wiki，用git来管理wiki；
* 检出Github上项目的docs目录，并用sphinx来生成在线文档（Read the Docs），Read the Docs也是个静态服务器；
* 用github issues系统来做评论系统(petal，comcom ) ；
* 用Github的markdown接口来渲染markdown；
* 用github的oauth来做第三方登录；
* 借助GitStats项目进行项目统计

## 如何掌握github

* 看[GitHub官方博客](https://github.com/blog)（https://github.com/blog），看Github的tip，[Explore](https://github.com/explore)（https://github.com/explore），写博客；
* 搜索有意思的项目，watch、fork和pull request
* 给牛人项目写wiki，提交测试用例和问题
* 帮牛人翻译中文；
* 学习区刻意练习
* GitHub公司传道士[schacon](https://github.com/schacon)所作projit（<https://github.com/progit/progit>）
* GitHub官方帮助文档（<https://help.github.com/en>）

## 疑问

wiki怎么使用？

借助GitStats项目进行项目统计？

|  |  |
| --- | --- |
| **网址** | <https://github.com/dmitryn/GitStats> |
| **操作** | #复制GitStats项目到本地  cd ~/dev  git clone git://github.com/trybeee/GitStats.git  python ~/dev/gitstats/git-stats /youproject public |

基于github写博客的几种方法？

|  |  |
| --- | --- |
| **方法一** | 借助[Jekyllbootstrap](http://jekyllbootstrap.com/)（http://jekyllbootstrap.com/），在Github上快速搭建一个基于jekyll的博客系统（参考http://www.yangzhiping.com/tech/wordpress-to-jekyll.html） |
| **方法二** | Octopress（参考http://www.yangzhiping.com/tech/octopress.html） |
| **方法三** | GitHub Pages（参考http://pages.github.com/） |

私有仓库能有多少个成员？

怎么从github公有仓库内下载单个目录？

答：进入要下载的文件内，鼠标右键选项Raw,选择“从链接另存文件为”即可。

# Git 服务器搭建

以Centos为例搭建 Git 服务器。

## 安装Git

|  |
| --- |
| 安装git |
| $ yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel perl-devel  $ yum install git |
| 创建git用户和用户组 |
| $ groupadd git  $ useradd git -g git |

## 创建证书登录

|  |
| --- |
| 搜集所有需要登陆用户的SSH公钥，导入到/home/git/.ssh/authorized\_keys文件里，每行一个公钥； |
| $ cd /home/git/  $ mkdir .ssh  $ chmod 755 .ssh  $ touch .ssh/authorized\_keys  $ chmod 644 .ssh/authorized\_keys |

## 初始化Git仓库

|  |
| --- |
| 选定一个目录作为git仓库，假定/home/gitrepo/test.git（git服务器上的仓库通常以.git结尾） |
| $ cd /home  $ mkdir gitrepo  $ chown git:git gitrepo/  $ cd gitrepo  $ git init --bare test.git  $ chown -R git:git runoob.git |

## 克隆仓库

|  |
| --- |
| git服务器已经部署完毕，可开始访问（需要有服务器IP） |
| git clone git@xxx.xxx.xxx.xxx:/home/gitrepo/test.git |

# Q&A

仓库怎么创建？

答：可以在空目录中建立，可以在已有文件目录建立，可以克隆远程仓库，可以fork别人的。

怎么知道某个文件对应的blob对象的哈希值和目录对应的tree对象的哈希值？

答：先知道commit对象，再知道tree对象，再知道blob对象。

创建分支之后，在分支中回到分支创建之前的提交，对于其他分支有什么影响，合并的时候会不会出问题？

答：测试发现，在新分支上回到创建分支之前的版本是不影响其他分支的

远程私有仓库怎么添加成员？

答：

.gitignore怎么使用？

答：

本地仓库怎么和某个远程空仓库建立连接？

答：

git branch –d 删除分支的时候不会谈警告提醒合并吗？

答：

SHA等那些码是什么意思?

答：

GNU Privacy Guard (GPG) 来签署或验证？

答：

pull requests和pull命令的区别？

答：

介绍工作区、缓存区和版本库？

答：

HEAD~和HEAD^区别？

答：

HEAD和master和commit之间的关系？

答：一般情况，HEAD指向master分支，master分支指向最近一次提交的指针。

.git/objects目录的作用？

答：在.git/objects目录下，git对象的哈希码前两位以目录形式存在，以后的作为文件名。

.git/index目录的作用？

答：./git/index目录下存储了索引区(暂存区)的git对象，但是无法直接查看，可通过命令“git ls-files -s“查看。

“分离头“有什么作用？

答：用来在以往提交的基础上做测试，会出现一条匿名分支，可转正也可丢弃这个匿名分支。

合并的时候fast-forward是什么？

答：目标分支无提交的合并，正常情况是创建合并提交(需要添加注解)。

合并操作注意事项？

答：先检出到目标分支，再执行合并

什么情况下属于冲突？

答：两个分支中同一文件的同一行不一样(需要在两个分支中至少有一次提交，否则不算冲突，算修改)。