# git命令

## git基本配置

|  |  |
| --- | --- |
| git config --global user.name “your name” | 设置名字 |
| git config --global user.email “your email” | 设置邮箱 |
| 如果设置错误，用以上命令修改即可 |  |
| git config --global core.editor emacs | 设置默认的编辑器，没设置的话默认是vim |
| git config --list | 列出所有配置 |
| git help config | 列出关于config的帮助文档，其他的类似 |

## ****git基本命令****

|  |  |
| --- | --- |
| git init | 初始化一个git仓库 |
| git add filename/(git add .) | 添加指定文件到版本库（或全部） |
| git commit -m “message” | 提交到仓库 |
| git commit –m “message” filename | 提交具体文件到仓库 |
| git commit -a -m ‘message’ | 跳过暂存区，直接提交 |
| git commit --amend | 如果提交出错，可以重新提交 |
| git status | 查看当前仓库状态 |
| git status –s | 仓库当前状态，以简短紧凑的方式输出  ??代表未追踪  M代表修改，但没有放入暂存区  A代表添加到暂存区 |
| git diff | 查看工作区与暂存区的修改（different）  此命令本身只显示尚未暂存的改动 |
| git diff --staged | 暂存区和上次提交的区别 |
| git log | 提交的历史信息 |
| git log --pretty=oneline | 让log信息以一行显示，除了oneline还有short、full、fuller |
| git log --pretty=format:”” | 格式化输出 |
| git log -p -2 | 显示每次提交的内容差异，-2显示最近两次的提交 |
| git log --stat | 每次提交的简略的统计信息 |

## Git版本回退

|  |  |
| --- | --- |
| HEAD | 表示当前版本 |
| HEAD^ 或者 HEAD~1 | 表示上个版本，以此类推 |
| git reset --hard HEAD~1 | 回退到上个版本 |
| git reset --hard commitId | 根据commit id 跳转到对应版本 |
| git reflog | 显示历史命令 |

## Git工作区与暂存区

|  |  |
| --- | --- |
| 工作区 | 实际工作的地方，文件创建与修改 |
| 版本库 | 隐藏目录.git,是git的版本库：版本库中存了很多东西，其中最重要的就是成为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 |
| 暂存区（stage或者index） | git add 实际上就是把文件的修改添加到暂存区  git commit 实际上是把暂存区的所有内容提交到当前分支  可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改 |

## Git管理修改

|  |  |
| --- | --- |
| git diff HEAD -- <filename> | 查看工作区和版本库里面最新版本的区别 |
| git checkout -- <filename> | 放弃工作区的修改，把<file>文件在工作区的修改全部撤销，让这个文件回到最近一次git commit 或 git add 时的状态 |
| git checkout -- \*.txt | 撤销所有的尚未提交的txt文件的修改 |
| git checkout -- . | 撤销所有的尚未提交的文件的修改 |
| git reset HEAD <filename> | 把暂存区的修改撤销掉，重新放回工作区 |
| git reset | 既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新版（看后面有没有跟文件路径） |
| git checkout | 既可以跳转分支，也可以丢弃工作区的修改（看后面有没有文件路径） |
| git rm filename | 删除一个文件，提交到git |
| git rm -f filename | 如果在删除之前修改过，并且提交到暂存区了，想删除的话就要用-f参数，强制删除 |
| git rm --cached filename | 如果提交了不该提交的文件，想把它从暂存区删除，但是不删除本地文件，就用这个命令 |
| git rm | 可以使用glob模式（正则表达式），比如：git rm log/\\*.log删除log目录下所有.log扩展名的文件。  或者git rm \\*~,删除所有以~结尾的文件 |
| git mv README.md README | 重命名文件，相当于：mv README.md README  git rm README.md  git add README |

## Git远程仓库

|  |  |
| --- | --- |
| git remote | 远程库信息 |
| git remote -v | 远程库的详细信息 |
| git remote show [remote-name] | 远程库的详细信息 |
| git fetch [remote-name] | 访问远程库，拉取你还没有的数据 |
| git remote rename pb paul | 给远程重命名，将pb改为paul |
| git remote rm paul | 移除一个远程库 |
| git fetch [remote-name] | 更新远程仓库引用,从远程仓库获取数据 |
| git checkout -b dev origin/dev | fetch之后，如果有新的远程分支，可以用这个命令创建一个本地分支作为这个远程分支的拷贝。以后就自动追踪这个远程分支了(dev可以改为其他名字) |
| git checkout --track origin/dev | 跟上面命令一样的效果，简化了命令 |
| git branch -u origin/dev | 设置已有的本地分支跟远程分支关联 |
| git fetch --all | 抓取所有的远程仓库 |
| git branch -vv | 查看本地和远程关联了的分支，会看到提交状况 |
| 创建SSH KEY | ssh-keygen -t rsa -C "youremail@example.com" |
| 在github添加ssh key |  |
| 在github上创建仓库 |  |
| 关联远程仓库 | git remote add origin git@github.com:Jiayupeng/learngit.git |
| 添加新的远程仓库，指定简写 | git remote add <shortname> <url> |
| git push (remote) (branch) | 把本地库的内容推送到远程，实际上是把当前分支master推送到远程 |
| git push -u [remote-name] [branch-name] | 推送到远程，加上-u参数，本地的master分支和远程的master分支会自动关联起来，以后用git push 就行了 |
| git pull | 相当于git fetch 和 git merge,从服务器获取数据，并合并 |
| 推送本地分支到远程 | git push origin local\_branch\_name:remote\_branch\_name |
| 删除与本地关联的远程分支 | git push origin :remote\_branch\_name |
| 删除远程分支 | git push origin --delete dev |
| 从远程库克隆 | git clone git@github.com:Jiayupeng/gitlearn.git |

## Git分支管理

### 创建与合并分支

|  |  |
| --- | --- |
| git checkout -b dev | 创建并切换到dev分支 |
| git log --oneline --decorate | 查看各个分支当前所指的对象 |
| git branch dev | 创建dev分支 |
| git checkout dev | 切换到dev分支 |
| git branch | 查看当前分支 |
| git branch -v | 每个分支的最后一次提交 |
| git merge | 用于合并指定分支到当前分支，比如 git merge dev |
| git rebase | 将一系列提交按照原有次序依次应用到另一分支上 |
| git rebase [basebranch] [topicbranch] | 将特性分支变基到目标分支上，变基完成后，就可以快速合并了  git checkout [basebranch]; git merge [topicbranch],最后删除分支 |
| rebase(变基)的原则 | **不要对在你的仓库外有副本的分支执行变基** |
| git branch -d dev | 删除dev分支 |

### 合并分支时冲突的解决

如果不同分支对同一个文件进行了修改，合并时就会产生冲突，此时需要手动解决冲突

|  |  |
| --- | --- |
| git status | 查看当前冲突的状态 |
| git log --graph | 可以看到分支合并图 |
| git log --graph --pretty=oneline --abbrev-commit | 显示合并图，一行显示，显示提交的内容 |

### 分支查看

|  |  |
| --- | --- |
| git branch |  |

### 分支管理策略

|  |  |
| --- | --- |
| 合并模式fast forward | 删除分支后，会丢掉分支信息 |
| 合并模式 --no-ff | 禁用fast forward模式，merge时生成一个commit |
| git merge --no-ff -m “commit message” | 具体命令 |
| 开发中分支策略的几个原则 | 1. master分支是最稳定的版本，仅用来发布新版本，平时不在上面干活 2. 平时在dev上面干活，当完成一个阶段的工作时，再把dev跟master合并 3. 每个人都有自己的分支，时不时的向dev合并就可以了。 |

### Bug分支

|  |  |
| --- | --- |
| git stash | 先把当前的工作现场储存起来，去别的分支修复bug |
| git stash list | 现有的stash |
| git stash apply | 应用stash,但是这个stash还在，没有删除 |
| git stash pop | 应用且删除这个stash |
| git stash drop | 删除这个stash |
| git stash apply stash@{0} | 可以多次stash，恢复的时候，先用git stash list查看，然后指定恢复哪个 |

### Feature分支

|  |  |
| --- | --- |
| git branch -D branchname | 要丢弃一个没有合并的分支，用这个命令，强制删除一个分支 |

### 多人协作

|  |  |
| --- | --- |
| git push origin master | 将本地master分支推送到远程 |
| git push origin dev | 将本地dev分支推送到远程 |
| git pull | 如果其他人也做了修改并推送到远程，你需要先把远程的最新版pull下来，在本地合并之后，才能继续push |
| git commit -m | 如果pull之后，有冲突，要先手动解决冲突，再提交，再push |
| 多人协作的模式 | 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，则因为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 |
| 问题解决 | 如果提示“no tracking information”，说明本地和远程没有关联，关联了就好了，命令在上面 |

## Git标签

|  |  |
| --- | --- |
| git tag tagname | 给最新的commit打标签 |
| git tag | 显示标签 |
| git tag tagname commitId | 对历史的提交打标签，后面跟上那次提交的commit id就行了，用  git log --pretty=oneline --abbrev-commit查看历史提交 |
| git tag -a <tagname> -m “message” | 创建带有说明信息的tag |
| git show tagname | 查看tag的说明信息 |
| git tag -d tagname | 删除标签 |
| git push origin tagname | 推送某个标签到远程 |
| git push origin --tags | 推送所有的未推送的标签到远程 |
| 删除远程的标签 | 1. 先删除本地标签 git tag -d tagname 2. 在删除远程标签：git push origin :refs/tags/tagname |

## Git忽略文件

|  |  |
| --- | --- |
| .gitignore文件 | touch .gitignore |

|  |
| --- |
| 所有空行或者以 ＃ 开头的行都会被 Git 忽略 |
| 可以使用标准的 glob 模式匹配。（指 shell 所使用的简化了的正则表达式） |
| 匹配模式可以以（/）开头防止递归。 |
| 匹配模式可以以（/）结尾指定目录。 |
| 要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号（!）取反。 |

|  |
| --- |
| 星号（\*）匹配零个或多个任意字符；[abc] 匹配任 何一个列在方括号中的字符（这个例子要么匹配一个 a，要么匹配一个 b，要么匹配一个 c）；问号（?）只匹配 一个任意字符；如果在方括号中使用短划线分隔两个字符，表示所有在这两个字符范围内的都可以匹配（比如 [0-9] 表示匹配所有 0 到 9 的数字）。使用两个星号（\*) 表示匹配任意中间目录，比如`a/\*\*/z` 可以匹配 a/z, a/b/z 或 `a/b/c/z`等。 |

例子：

|  |
| --- |
| # no .a files \*.a # but do track lib.a, even though you're ignoring .a files above !lib.a # only ignore the TODO file in the current directory, not subdir/TODO /TODO # ignore all files in the build/ directory build/ # ignore doc/notes.txt, but not doc/server/arch.txt doc/\*.txt # ignore all .pdf files in the doc/ directory doc/\*\*/\*.pdf |

|  |
| --- |
| github上的例子：https://github.com/github/gitignore |

## git log 格式化表示

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | %H | 提交对象（commit）的完整哈希字串 | | %h | 提交对象的简短哈希字串 | | %T | 树对象（tree）的完整哈希字串 | | %t | 树对象的简短哈希字串 | | %P | 父对象（parent）的完整哈希字串 | | %p | 父对象的简短哈希字串 | | %an | 作者（author）的名字 | | %ae | 作者的电子邮件地址 | | %ad | 作者修订日期（可以用 --date= 选项定制格式） | | %ar | 作者修订日期，按多久以前的方式显示 | | %cn | 提交者(committer)的名字 | | %ce | 提交者的电子邮件地址 | | %cd | 提交日期 | | %cr | 提交日期，按多久以前的方式显示 | | %s | 提交说明 | |