配置查找顺序 仓库目录-当前用户-git安装目录

git config 设置 仓库目录\.git\config

git config --global 命令设置 当前用户 C:\Users\ye\.gitconfig

git config --system 命令设置 在 git安装目录\etc\gitconfig

配置当前用户信息

git config --global user.name "John Doe"

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

配置文本编辑器

git config --global core.editor emacs

罗列配置信息

git config --list

配置文件 C:\Users\ye\.gitconfig

[user]

name = yesunboy

email = yebohuangsunboy@163.com

[diff]

tool = bc3

[difftool "bc3"]

path = E:\\tools\\Beyond Compare 4\\BCompare.exe

[merge]

tool=bcmerge

[mergetool "bcmerge"]

cmd = "\"E:/tools/Beyond Compare 4/BCompare.exe\" \"$LOCAL\" \"$REMOTE\" \"$BASE\" \"$MERGED\""

keepBackup = false

[core]

editor = notepad

[alias]

last = log -1 HEAD

st = status

ci = commit

br = branch

co = checkout

查找指定配置信息

git config user.name

获取帮助

git help <verb>

git <verb> --help

man git-<verb>

有两种方法建立 Git 仓库

1、初始化本地目录为git仓库

进入本地目录，然后执行以下命令

git init

2、克隆现有的仓库

git clone [url]

git clone https://github.com/libgit2/libgit2 mylibgit

添加到暂存区

Git add 文件名

生命周期



检查状态

**$** git status -s

新添加的未跟踪文件前面有 ?? 标记，新添加到暂存区中的文件前面有 A 标记，修改过的文件前面有 M标记。 你可能注意到了 M 有两个可以出现的位置，出现在右边的 M 表示该文件被修改了但是还没放入暂存区，出现在靠左边的 M 表示该文件被修改了并放入了暂存区。 例如，上面的状态报告显示：README 文件在工作区被修改了但是还没有将修改后的文件放入暂存区,lib/simplegit.rb 文件被修改了并将修改后的文件放入了暂存区。 而 Rakefile 在工作区被修改并提交到暂存区后又在工作区中被修改了，所以在暂存区和工作区都有该文件被修改了的记录。

### [忽略文件](https://git-scm.com/book/zh/v2/Git-%E5%9F%BA%E7%A1%80-%E8%AE%B0%E5%BD%95%E6%AF%8F%E6%AC%A1%E6%9B%B4%E6%96%B0%E5%88%B0%E4%BB%93%E5%BA%93#忽略文件)

一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。 通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。 在这种情况下，我们可以创建一个名为.gitignore 的文件，列出要忽略的文件模式。

创建.gitignore文件

touch .gitignore

对比工作目录当前文件与暂存区或快照的差异

**$** git diff

对比暂存区与快照的差异

**$** git diff --staged

用外部工具对比差异

Git difftool

对比两个版本之间的差异

git difftool 80ba6dc1133f1dfd3ff181f6e2c3ebeadf710085 3da36f88a5e95b78205df3321efe95edce49d999 -- 2.txt

提交暂存区里的修改

git commit -m "Story 182: Fix benchmarks for speed"

跳过暂存步骤

git commit -a

设置difftool采用BCompare对比差异

git config --global diff.tool bc3

git config --global difftool.bc3.path "E:\\tools\\Beyond Compare 4\\BCompare.exe"

移除文件

git rm PROJECTS.md

如果已经暂存就要加f

git rm -f PROJECTS.md

删除暂存区并保留本地文件

git rm --cached README

移动文件

git mv README.md README

查看提交历史

git log

显示最近两次提交

git log -p -2

查看某个文件历史提交

git log -p 2.txt

每次提交简易统计信息

git log --stat

展示格式

git log --pretty=oneline

git log --pretty=format:"%h - %an, %ar : %s"

| **选项** | **说明** |
| --- | --- |
| %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 |

| **选项** | **说明** |
| --- | --- |
| -p | 按补丁格式显示每个更新之间的差异。 |
| --stat | 显示每次更新的文件修改统计信息。 |
| --shortstat | 只显示 --stat 中最后的行数修改添加移除统计。 |
| --name-only | 仅在提交信息后显示已修改的文件清单。 |
| --name-status | 显示新增、修改、删除的文件清单。 |
| --abbrev-commit | 仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。 |
| --relative-date | 使用较短的相对时间显示（比如，“2 weeks ago”）。 |
| --graph | 显示 ASCII 图形表示的分支合并历史。 |
| --pretty | 使用其他格式显示历史提交信息。可用的选项包括 oneline，short，full，fuller 和 format（后跟指定格式）。 |

撤销操作

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

### 取消暂存的文件

git reset HEAD CONTRIBUTING.md

撤销修改

git checkout -- CONTRIBUTING.md

列出远程仓库

git remote -v

添加远程仓库

git remote add <shortname> <url>

git remote add pb https://github.com/paulboone/ticgit

获取远程仓库数据

git fetch [remote-name]

git fetch pb

推送到远程仓库

git push [remote-name] [branch-name]

git push origin master

查看远程仓库

git remote show [remote-name]

git remote show origin

远程仓库重命名

git remote rename pb paul

移除远程仓库

git remote rm paul

列出标签

git tag

git tag -l 'v1.8.5\*'

创建附注标签

git tag -a v1.4 -m 'my version 1.4'

通过使用 git show 命令可以看到标签信息与对应的提交信息

git show v1.4

创建轻量标签

git tag v1.4-lw

给特定的提交打标签

git tag -a v1.2 9fceb02

共享标签

git push origin v1.5

git push origin --tags

检出标签

git checkout -b [branchname] [tagname]

git checkout -b version2 v2.0.0

git别名

$ git config --global alias.co checkout

$ git config --global alias.br branch

$ git config --global alias.ci commit

$ git config --global alias.st status

Git 只是简单地将别名替换为对应的命令。 然而，你可能想要执行外部命令，而不是一个 Git 子命令。 如果是那样的话，可以在命令前面加入 ! 符号。 如果你自己要写一些与 Git 仓库协作的工具的话，那会很有用。 我们现在演示将 git visual 定义为 gitk 的别名：

$ git config --global alias.visual '!gitk'

创建分支

git branch testing

分支切换

git checkout testing

创建并切换分支

git checkout -b iss53

合并分支

git merge hotfix

删除分支

git branch -d hotfix

查看每一个分支的最后一次提交

git branch -v

查看哪些分支已经合并到当前分支

git branch --merged

查看所有包含未合并工作的分支

git branch --no-merged

推送分支

git push (remote) (branch)

推送本地分支serverfix到远程分支awesomebranch

git push origin serverfix:awesomebranch

跟踪远程分支

git checkout --track origin/serverfix

git checkout -b sf origin/serverfix

设置已有的本地分支跟踪一个刚刚拉取下来的远程分支

git branch -u origin/serverfix

查看设置的所有跟踪分支

git branch -vv

抓取所有远程

git fetch --all

删除远程分支

git push origin --delete serverfix

变基

git checkout mywork

git rebase origin

将 client 中的修改合并到主分支并发布，但暂时并不想合并 server 中的修改，因为它们还需要经过更全面的测试。 这时，你就可以使用 git rebase 命令的 --onto 选项，选中在 client 分支里但不在 server 分支里的修改（即 C8 和 C9），将它们在 master 分支上重放

git rebase --onto master server client

查看远程分支

git branch -r

查看所有分支

git branch -a

生成sshkey

ssh-keygen -t rsa -C [yebohuangsunboy@163.com](mailto:yebohuangsunboy@163.com)

合并工具p4merge配置

安装p4merge 并加入环境变量path

在.gitconfig加入

[merge]

tool=p4merge

[mergetool "p4merge"]

cmd="p4merge.exe $BASE $LOCAL $REMOTE $MERGED"

trustExitCode=true

keepBackup=false

Beyond Compare作为差异合并工具

[diff]

tool = bc3

[difftool "bc3"]

path = E:\\tools\\Beyond Compare 4\\BCompare.exe

[merge]

tool=bcmerge

[mergetool "bcmerge"]

cmd = "\"E:/tools/Beyond Compare 4/BCompare.exe\" \"$LOCAL\" \"$REMOTE\" \"$BASE\" \"$MERGED\""

keepBackup = false