# Git 配置

Git 提供了一个叫做 git config 的工具，专门用来配置或读取相应的工作环境变量。

这些环境变量，决定了 Git 在各个环节的具体工作方式和行为。这些变量可以存放在以下三个不同的地方：

* /etc/gitconfig 文件：系统中对所有用户都普遍适用的配置。若使用 git config 时用 --system 选项，读写的就是这个文件。
* ~/.gitconfig 文件：用户目录下的配置文件只适用于该用户。若使用 git config 时用 --global 选项，读写的就是这个文件。
* 当前项目的 Git 目录中的配置文件（也就是工作目录中的 .git/config 文件）：这里的配置仅仅针对当前项目有效。每一个级别的配置都会覆盖上层的相同配置，所以 .git/config 里的配置会覆盖 /etc/gitconfig 中的同名变量。

# 基本概念

我们先来理解下Git 工作区、暂存区和版本库概念

**工作区**：就是你在电脑里能看到的目录。

**暂存区**：英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。

**版本库**：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

下面这个图展示了工作区、版本库中的暂存区和版本库之间的关系：



## git init

Git 使用 git init 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以 git init 是使用 Git 的第一个命令。在执行完成 git init 命令后，Git 仓库会生成一个 .git 目录，该目录包含了资源的所有元数据，其他的项目目录保持不变。

## git clone

我们使用 git clone 从现有 Git 仓库中拷贝项目。

## git add

git add 命令可将该文件添加到缓存。

可以使用 **git add .** 命令来添加当前项目的所有文件。

## git status

git status命令用于查看项目的当前状态。

## git diff

执行 git diff 来查看执行 git status 的结果的详细信息。

git diff 命令显示已写入缓存与已修改但尚未写入缓存的改动的区别。git diff 有两个主要的应用场景。

1. 尚未缓存的改动：git diff
2. 查看已缓存的改动： git diff --cached
3. 查看已缓存的与未缓存的所有改动：git diff HEAD
4. 显示摘要而非整个 diff：git diff –stat

## git commit

使用 git add 命令将想要快照的内容写入缓存区， 而执行 git commit 将缓存区内容添加到仓库中。

如果你觉得 git add 提交缓存的流程太过繁琐，Git 也允许你用 -a 选项跳过这一步。

## git remote

要查看当前配置有哪些远程仓库

## git push

推送你的新分支与数据到某个远端仓库命令

## git pull

从远端仓库提取数据并尝试合并到当前分支，该命令就是在执行 git fetch 之后紧接着执行 git merge 远程分支到你所在的分支。

## git fetch

从远程仓库下载新分支与数据，该命令执行完后需要执行git merge 远程分支到你所在的分支。

## git reset HEAD

git reset HEAD 命令用于取消已缓存的内容。

## git rm

git rm 会将条目从缓存区中移除。这与 git reset HEAD 将条目取消缓存是有区别的。 "取消缓存"的意思就是将缓存区恢复为我们做出修改之前的样子。

默认情况下，git rm file 会将文件从缓存区和你的硬盘中（工作目录）删除。

如果你要在工作目录中留着该文件，可以使用 **git rm –cached**

## git mv

git mv 命令做得所有事情就是重命名磁盘上的文件，然后再执行 git add 把新文件添加到缓存区。

## git branch (branchname)

创建分支命令, 没有参数时，git branch 会列出你在本地的分支。

使用git branch -d (branchname)来删除分支。

## git checkout (branchname)

切换分支命令，git checkout -b (branchname) 命令来创建新分支并立即切换到该分支下，从而在该分支中操作。注意：当前已是最新而误删的文件未提交前可以用此命令恢复文件。

## git merge

一旦某分支有了独立内容，你终究会希望将它合并回到你的主分支。 你可以使用该命令将任何分支合并到当前分支中去。合并失败时，需要手动修改合并失败的点(git diff)，然后重新提交(git add)修正合并。

## git log

列出历史提交记录，可以用 --oneline 选项来查看历史记录的简洁的版本。还可以用 --graph 选项，查看历史中什么时候出现了分支、合并。更清楚明了地看到何时工作分叉、又何时归并。可以用 –reverse 参数来逆向显示所有日志。

如果只想查找指定用户的提交日志可以使用命令：git log –author

$ git log --author=Linus --oneline -5

如果你要指定日期，可以执行几个选项：--since 和 --before，但是你也可以用 --until 和 --after。

$ git log --oneline --before={3.weeks.ago} --after={2010-04-18} --no-merges

git tag

Git的标签命令，使用的标签有两种类型：轻量级的（lightweight）和含附注的（annotated）。

轻量级标签就像是个不会变化的分支，实际上它就是个指向特定提交对象的引用。而含附注标签，实际上是存储在仓库中的一个独立对象，它有自身的校验和信息，包含着标签的名字，电子邮件地址和日期，以及标签说明，标签本身也允许使用 GNU Privacy Guard (GPG) 来签署或验证。一般我们都建议使用含附注型的标签，以便保留相关信息；当然，如果只是临时性加注标签，或者不需要旁注额外信息，用轻量级标签也没问题。

git tag 查看已有标签

git tag -a v1.0 创建带附注的标签v1.0

git tag -d v1.1 删除标签v1.1