# Git学习

## 1. 学习网站

菜鸟教程：<https://www.runoob.com/git/git-tutorial.html>

图1. 菜鸟-Git教程

## 2. Git介绍

Git 是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。适用于各种团队或个人协作项目。

Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

图2. Linus Torvalds

Git采用了分布式版本库的方式，不必服务器端软件支持。

## 3. Git安装配置

### 3.1 安装

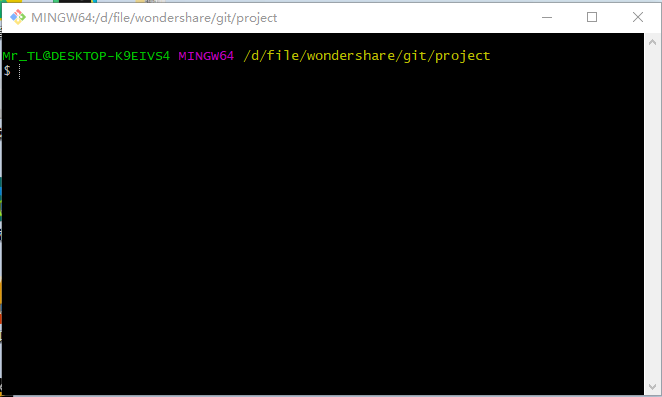
下载地址：<http://git-scm.com/downloads>

图3. 安装Git（包括GUI和BASH）

### 3.2 配置

#### 3.2.1 Git config

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：

1. /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
2. ~/.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（windows中主目录是$HOME 变量指定的目录）
3. 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。

#### 3.2.2 配置用户信息

配置个人的用户名称和电子邮件地址（如果要在某个特定的项目中使用其他名字或者电邮，只要去掉 --global 选项重新配置即可，新的设定保存在当前项目的 .git/config 文件里）：

$ git config --global user.name "TianYuemeng"

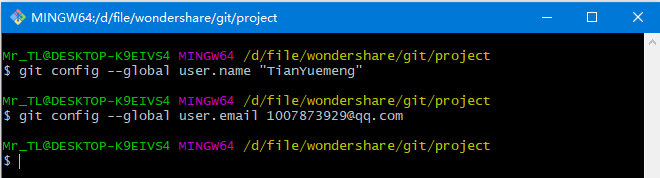
$ git config --global user.email 1007873929@qq.com

图4. 配置用户信息

#### 3.2.3 文本编辑器

Git默认使用Vi或者Vim作为编辑器，如果有其他偏好可以重新设置（如Emacs）：

$ git config --global core.editor emacs

#### 3.2.4 差异分析工具

在解决合并冲突时使用哪种差异分析工具。比如要改用 vimdiff 的话：

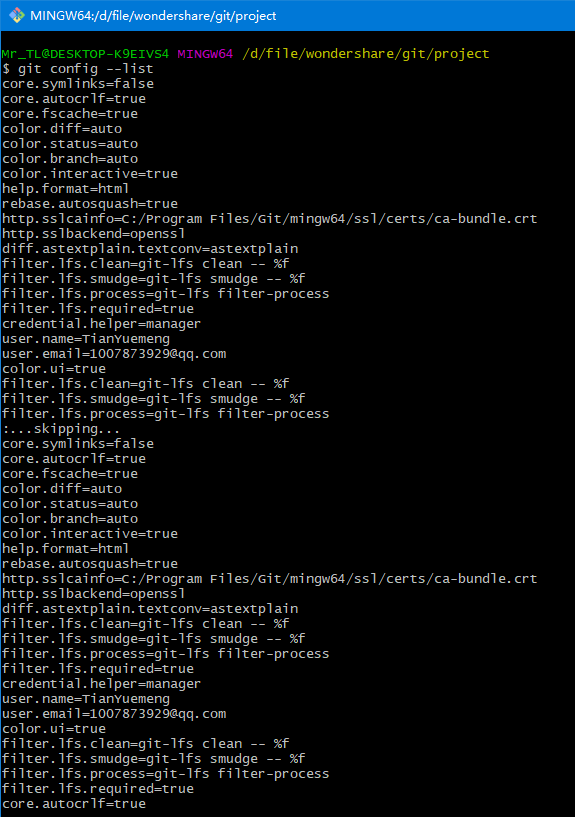
$ git config --global merge.tool vimdiff

#### 3.2.5 查看配置信息

$ git config --list

有时候会看到重复的变量名，那就说明它们来自不同的配置文件（比如 /etc/gitconfig 和 ~/.gitconfig），不过**最终Git实际采用的是最后一个**。

这些配置我们也可以在 ~/.gitconfig 或 /etc/gitconfig 看到。

图4. 查看配置信息

## https://www.runoob.com/wp-content/uploads/2015/02/git-process.png4. Git工作流程

图5. Git工作流程

## 5. Git工作区、暂存区和版本库

* **工作区**：就是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 目录下的 index 文件（.git/index）中，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。

图6. 工作区、版本库中的暂存区和版本库之间的关系

* 图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage/index），标记为 "master" 的是 master 分支所代表的目录树。
* 图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。
* 图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。
* 当对工作区修改（或新增）的文件执行 git add 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。
* 当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。
* 当执行 git reset HEAD 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。
* 当执行 git rm --cached <file> 命令时，会直接从暂存区删除文件，工作区则不做出改变。
* 当执行 git checkout . 或者 git checkout -- <file> 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。
* 当执行 git checkout HEAD . 或者 git checkout HEAD <file> 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

## 6. Git创建仓库

### 6.1 git init

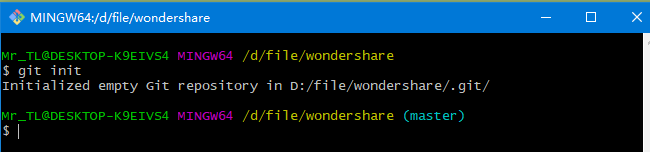
Git 使用 git init 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以 git init 是使用 Git 的第一个命令。在执行完成 git init 命令后，**Git 仓库会生成一个 .git 目录，该目录包含了资源的所有元数据，其他的项目目录保持不变**。

图7. 使用git init初始化Git仓库

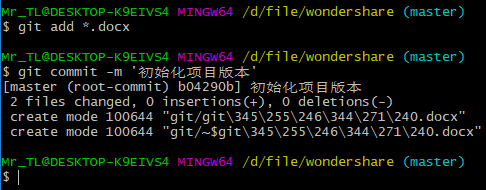
初始化后，会在目录下会出现一个名为 .git 的目录，所有 Git 需要的数据和资源都存放在这个目录中。如果当前目录下有几个文件想要纳入版本控制，需要先用 git add 命令告诉 Git 开始对这些文件进行跟踪，然后提交。

图8. 使用git add将文件纳入版本控制并提交

### 6.2 git clone

克隆仓库的命令格式为：

$ git clone <repo>

如果需要克隆到指定的目录，可以使用以下命令格式：

$ git clone <repo> <directory>

比如，要克隆 Ruby 语言的 Git 代码仓库 Grit，可以用下面的命令：

$ git clone git://github.com/schacon/grit.git

## 7. Git基本操作

### 7.1 Git常用的六个命令

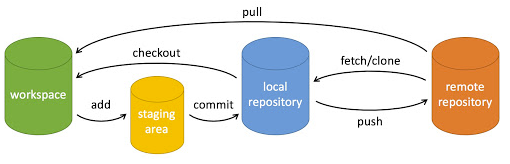
**git clone、git push、git add 、git commit、git checkout、git pull**

图9. Git常用的六个命令

### 7.2 Git其他命令

|  |  |
| --- | --- |
| git status | 查看仓库当前的状态，显示有变更的文件 |
| git diff | 比较文件的不同，即暂存区和工作区的差异 |
| git reset | 回退版本（将暂存区文件退回，add的反向操作） |
| git rm | 删除工作区文件 |
| git mv | 移动或重命名工作区文件 |
| git log | 查看历史提交记录（gitk可以图形化查看） |
| git blame | 以列表形式查看指定文件的历史修改记录 |

## 8. Git分支管理

|  |  |
| --- | --- |
| git branch (branchname) | 创建分支 |
| git checkout (branchname) | 切换分支 |
| git merge | 合并分支 |
| git branch | 列出分支 |