# Git概述

## 什么是Github

GitHub是一个面向开源及私有软件项目的托管平台。

## 什么是GIT

Git是一个开源的分布式版本控制系统。

## Git中的相关概念

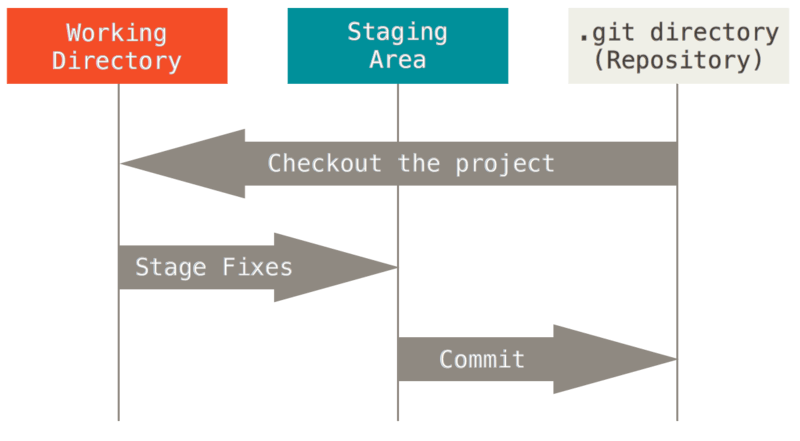
Git 有三种状态，文件可能处于其中之一： **已提交（committed）**、**已修改（modified）** 和 **已暂存（staged）**。

**已修改**表示修改了文件，但还没保存到数据库中。

**已暂存**表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。

**已提交**表示数据已经安全地保存在本地数据库中。

这会让我们的 Git 项目拥有三个阶段：工作区、暂存区以及 Git 目录。



**工作区**是对项目的某个版本独立提取出来的内容;

**暂存区**是一个文件，保存了下次将要提交的文件列表信息，一般在 Git 仓库目录中。

**Git 仓库目录**是 Git 用来保存项目的元数据和对象数据库的地方。 这是 Git 中最重要的部分，从其它计算机克隆仓库时，复制的就是这里的数据。

## 基本的 Git 工作流程：

1.在工作区中修改文件。

2.将你想要下次提交的更改选择性地暂存，这样只会将更改的部分添加到暂存区。

3.提交更新，找到暂存区的文件，将快照永久性存储到 Git 目录。

如果 Git 目录中保存着特定版本的文件，就属于 **已提交** 状态。 如果文件已修改并放入暂存区，就属于 **已暂存** 状态。 如果自上次检出后，作了修改但还没有放到暂存区域，就是 **已修改** 状态。

## Git 使用方式

Git有多种使用方式,可以使用原生的命令行模式，也可以使用 GUI 模式.

## GIT的安装

<https://git-scm.com/>

## 初次运行 Git 前的配置

## 设置你的用户名和邮件地址

## Git获取帮助

$ git help <verb>

$ git <verb> --help

$ man git-<verb>

实例：$ git help config

说明：获取config命令的手册

# GIT基础

# 获取 Git 仓库

### 获取 Git 项目仓库的方式：

1.将尚未进行版本控制的本地目录转换为 Git 仓库；

2.从其它服务器 **克隆** 一个已存在的 Git 仓库。

两种方式都会在你的本地机器上得到一个工作就绪的 Git 仓库。

### 在已存在目录中初始化仓库

如果你有一个尚未进行版本控制的项目目录，想要用 Git 来控制它，那么首先需要进入该项目目录中。 如果你还没这样做过，那么不同系统上的做法有些不同：

在 Linux 上：

$ cd /home/user/my\_project

在 macOS 上：

$ cd /Users/user/my\_project

在 Windows 上：

$ cd /c/user/my\_project

之后执行：

$ git init

该命令将创建一个名为 .git 的子目录，这个子目录含有你初始化的 Git 仓库中所有的必须文件，这些文件是 Git 仓库的骨干。 但是，在这个时候，我们仅仅是做了一个初始化的操作，你的项目里的文件还没有被跟踪

如果在一个已存在文件的文件夹（而非空文件夹）中进行版本控制，你应该开始追踪这些文件并进行初始提交。 可以通过 git add 命令来指定所需的文件来进行追踪，然后执行 git commit ：

$ git add \*.c

$ git add LICENSE

$ git commit -m 'initial project version'

### 克隆现有的仓库

如果你想获得一份已经存在了的 Git 仓库的拷贝，这时就要用到 git clone 命令。

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

会在当前目录下创建一个名为 “libgit2” 的目录，并在这个目录下初始化一个 .git 文件夹， 从远程仓库拉取下所有数据放入 .git 文件夹，然后从中读取最新版本的文件的拷贝。 如果你进入到这个新建的 libgit2 文件夹，你会发现所有的项目文件已经在里面了，准备就绪等待后续的开发和使用。

如果你想在克隆远程仓库的时候，自定义本地仓库的名字，你可以通过额外的参数指定新的目录名：

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

这会执行与上一条命令相同的操作，但目标目录名变为了 mylibgit。

Git 支持多种数据传输协议。 上面的例子使用的是 https:// 协议，不过你也可以使用 git:// 协议或者使用 SSH 传输协议，比如 user@server:path/to/repo.git 。

# 记录每次更新到仓库

## 记录每次更新到仓库

现在我们的机器上有了一个 **真实项目** 的 Git 仓库，并从这个仓库中检出了所有文件的 **工作副本**。 通常，你会对这些文件做些修改，每当完成了一个阶段的目标，想要将记录下它时，就将它提交到仓库。

请记住，你工作目录下的每一个文件都不外乎这两种状态：**已跟踪** 或 **未跟踪**。 已跟踪的文件是指那些被纳入了版本控制的文件，在上一次快照中有它们的记录，在工作一段时间后， 它们的状态可能是未修改，已修改或已放入暂存区。简而言之，已跟踪的文件就是 Git 已经知道的文件。

工作目录中除已跟踪文件外的其它所有文件都属于未跟踪文件，它们既不存在于上次快照的记录中，也没有被放入暂存区。 初次克隆某个仓库的时候，工作目录中的所有文件都属于已跟踪文件，并处于未修改状态，因为 Git 刚刚检出了它们， 而你尚未编辑过它们。

编辑过某些文件之后，由于自上次提交后你对它们做了修改，Git 将它们标记为已修改文件。 在工作时，你可以选择性地将这些修改过的文件放入暂存区，然后提交所有已暂存的修改，如此反复。



## 检查当前文件状态

可以用 git status 命令查看哪些文件处于什么状态。

$ git status

## 跟踪新文件

使用命令 git add 开始跟踪一个文件。

$ git add README

此时再运行 git status 命令，会看到 README 文件已被跟踪，并处于暂存状态。

$ git status

## 暂存已修改的文件

修改一个已被跟踪的文件。 如果你修改了一个名为 CONTRIBUTING.md 的已被跟踪的文件，然后运行 git status 命令，会看到已跟踪文件的内容发生了变化，但还没有放到暂存区。 要暂存这次更新，需要运行 git add 命令。

$ git status

## 状态简览

git status 命令的输出十分详细，但其用语有些繁琐。 Git 有一个选项可以帮你缩短状态命令的输出，这样可以以简洁的方式查看更改。 如果你使用 git status -s 命令或 git status --short 命令，你将得到一种格式更为紧凑的输出。

$ git status -s

M README

MM Rakefile

A lib/git.rb

M lib/simplegit.rb

?? LICENSE.txt

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。 输出中有两栏，左栏指明了暂存区的状态，右栏指明了工作区的状态。

## 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，列出要忽略的文件的模式。

.classpath

.project

.settings

Target

文件 .gitignore 的格式规范如下：

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，它会递归地应用在整个工作区中。

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，可以在模式前加上叹号（!）取反。

### 查看已暂存和未暂存的修改

## Git的使用步骤

1. 注册Github账号
2. 在Github上创建仓库
3. 在本地安装Git、初始化和配置Git
4. 将文件提交到github中

# 初始化和配置Git

## 1.初始化git

语法：git init;

说明:初始化git

## 2.配置Git账号信息(名称和邮箱)

1.语法：git config user.name "名称"

   说明：配置用户名

   实例：git config user.name "wangtao-luse"

 2.语法：git config user.email "邮箱"

   说明：配置邮箱

   实例：git config user.email "wangtao@163.com"

2.查看配置的信息

语法：git cofig user.name

说明：查看名称

语法：git config user.email

说明：查看邮箱;

# Git添加和提交文件

Git分为工作区，缓存区，版本库;