git 教程

注：本教程引用了git官方文档的内容，只用于教学，不用于盈利

什么是git?

- Git 是一种分布式版本控制系统（dVCS）。顾名思义，版本控制系统就是控制和跟踪特定项目的不同版本。

git的历史

- Git 由芬兰软件工程师 Linus Torvalds 于 2005 年开发，他也因开发了 Linux 操作系统内核而受到赞誉。 Git 的创建是为了解决眼前的需求。 在其发明之前，世界各地的 Linux 开发人员都在使用专有软件 BitKeeper，它本身就是一个 dVCS。 由于该软件归公司所有，因此在 Linux 开发人员中引起了一些争论，其中大多数人都拥护开源精神。 作为免费使用该软件的回报，BitKeeper 背后的公司 BitMover 对 Linux 社区设置了限制。 据 Linux Journal 报道，这些限制之一是它们无法在竞争的版本控制项目上工作。 一位 Linux 开发人员开始对 BitKeeper 进行逆向工程，以努力创建开源产品，这一举动也许是不可避免的。 正如其承诺，BitMover 停止向 Linux 内核提供服务，分布式开发系统陷入了不确定性。 为了解决这个难题，Torvalds 自 1991 年以来首次停止了 Linux 上的工作，并创建了 Git，并在开始开发几个月后发布了稳定版本。 有趣的是，在 Linux 内核首先采用 BitKeeper 之前，开发人员独立向 Torvalds 发送他们的补丁（更改），而他则在需要时集成这些补丁。 2016 年，即 Git 发布 11 年后，BitKeeper **开源**了。
  （注意：开源万岁！。开假源罪该万死）

开始我们的git学习之旅吧！

安装

- 如果你是windows用户，请你到这个网站去安装git

> https://git-scm.com/

- 如果你是linux（ubuntu）用户，你应该是自带git的,如果你发现你可能没有，可以使用以下命令来安装你的个git:

> sudo apt-get install git

- 我认为目前应该没有mac用户，如果有，请移步macbook安装git

git配置

- Git 自带一个 git config 的工具来帮助设置控制 Git 外观和行为的配置变量。 这些变量存储在三个不同的位置：

> 1. /etc/gitconfig 文件: 包含系统上每一个用户及他们仓库的通用配置。 如果在执行 git config 时带上 --system 选项，那么它就会读写该文件中的配置变量。 （由于它是系统配置文件，因此你需要管理员或超级用户权限来修改它。）
> 2. ~/.gitconfig 或 ~/.config/git/config 文件：只针对当前用户。 你可以传递 --global 选项让 Git 读写此文件，这会对你系统上 所有 的仓库生效。
> 3. 当前使用仓库的 Git 目录中的 config 文件（即 .git/config）：针对该仓库。 你可以传递 --local 选项让 Git 强制读写此文件，虽然默认情况下用的就是它。。 （当然，你需要进入某个 Git 仓库中才能让该选项生效。）

- 如果你想知道你的git的具体的位置，你可以使用：

> git config --list --show-origin

- 安装完 Git 之后，要做的第一件事就是设置你的用户名和邮件地址。 这一点很重要，因为每一个 Git 提交都会使用这些信息

> git config --global user.name "username"
> git config --global user.email (your mail)

- 配置结束后你可以使用

> git config --list
> 来查看你的配置是是否成功

笔者配置完是这样的：

```
user.name=Bob0817912
user.email=2099572595@qq.com
credential.https://github.com.helper=
credential.https://github.com.helper=!/usr/bin/gh auth git-credential
```

你也可以通过**git config**命令来专门看某些内容
比如：

```
suibian@suibian-A35S:~$ git config user.email
2099572595@qq.com
```

git基础

获取 Git 仓库

通常有两种获取 Git 项目仓库的方式：

- 将尚未进行版本控制的本地目录转换为 Git 仓库
- 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。

本地建立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/the name of project

在 macOS 上：

> cd /Users/user/the name of project

在 Windows 上：

> cd /c/user/the name of project

之后执行：

> git init

执行这些命令之后你的仓库就会出现一个 **.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。

.git的秘密

> .git 文件夹包含项目所需的所有信息，以及与提交、远程仓库地址等相关的所有信息。它还包含一个保存提交历史的日志。该日志可以帮助你回滚到所需的代码版本。

以下是 .git 文件夹中的几个重要子目录和文件

- hooks --该文件夹包含脚本文件。Git hooks是在提交、推送等事件前后执行的脚本
- objects --该文件夹代表 Git 的对象数据库
- config --这是本地配置文件
- refs --该文件夹存储有关标记和分支的信息
- HEAD --该文件存储对当前分支的引用。默认情况下指向主分支
- index - 这是一个二进制文件，存储分期信息

克隆现有的仓库

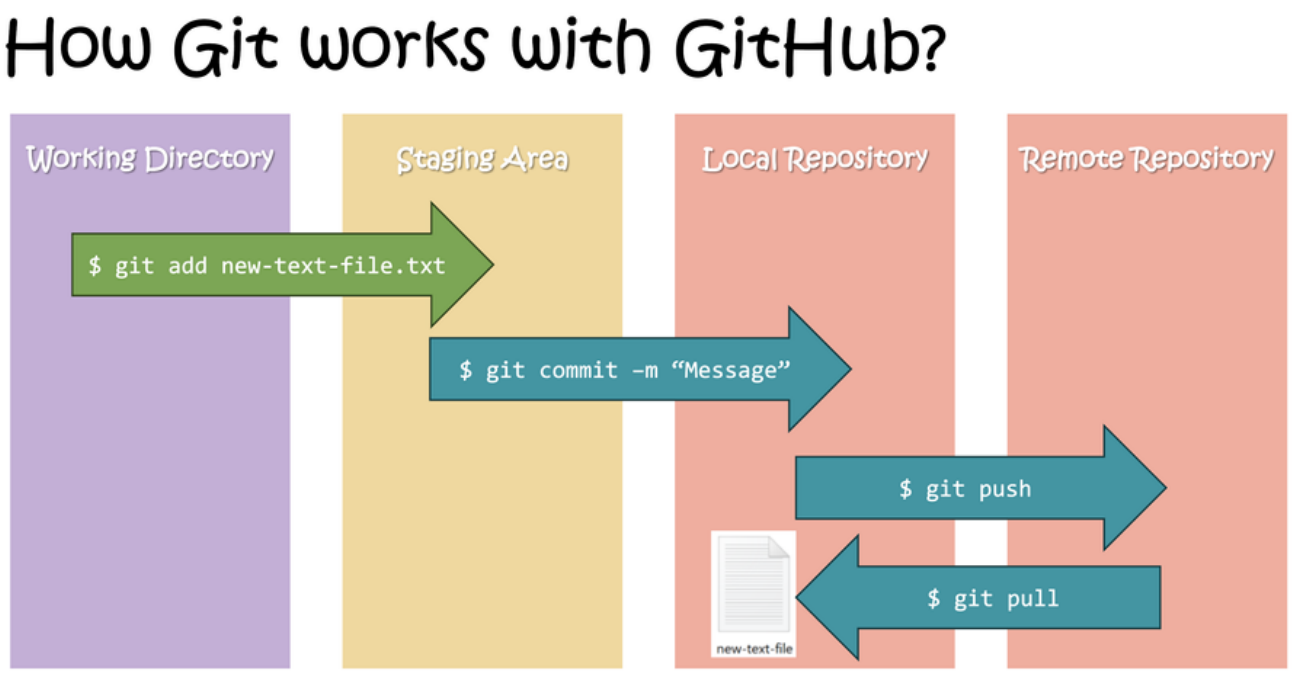
如果你想要git别人的仓库，或者你在远端的仓库，你就可以使用git clone

> git clone https://github.com/knot41/python1.git

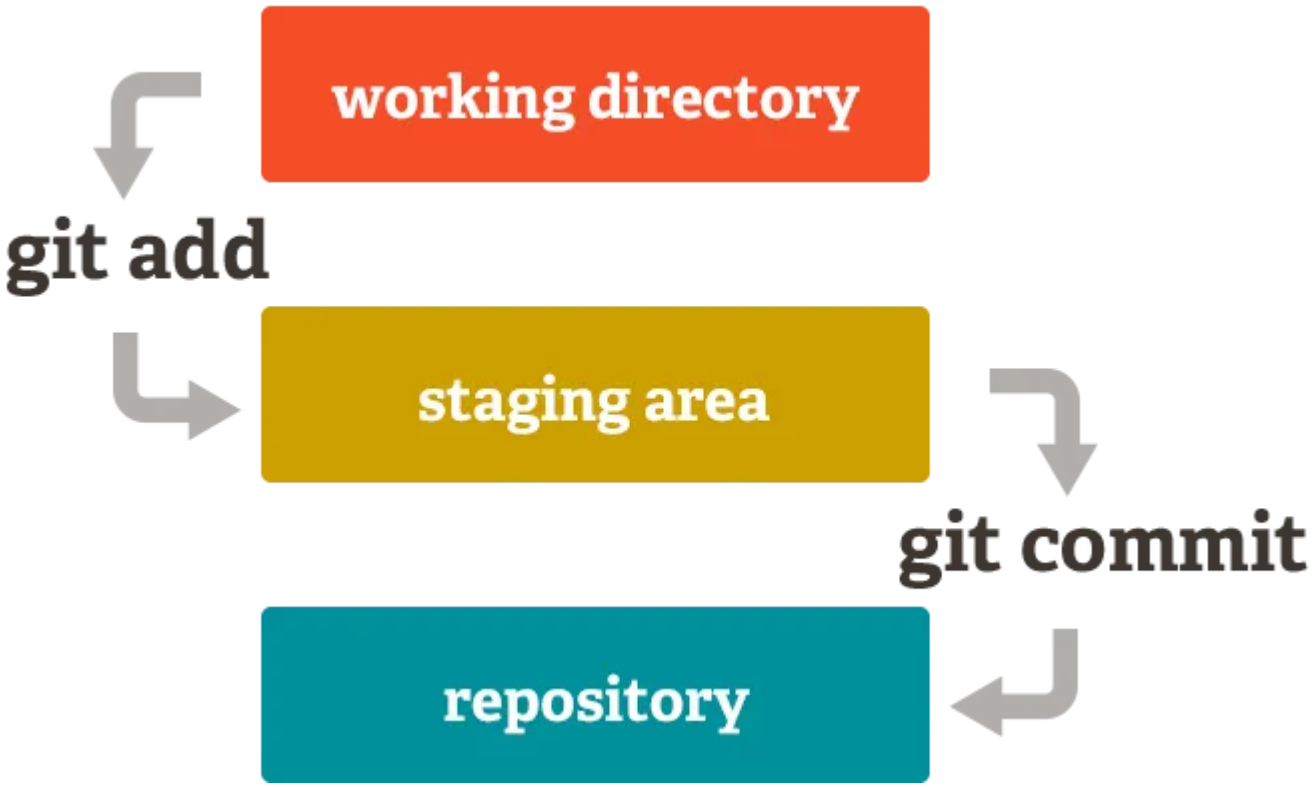
如果你想在克隆远程仓库的时候，自定义本地仓库的名字，你可以通过额外的参数指定新的目录名：

> git clone https://github.com/knot41/python1.git Tetris

我们该如何使用git应用到实际工作呢？



还记得我们上面提到的创建仓库吗？当我们使用git init初始化之后，我们就要开始将自己仓库里内容提交上去了。
git 总共有三个区域：



- 第一步我们要把我们本地执行的更改使用**git add** 命令提交到暂存区

> git add *.c
> git add README
> git commit -m 'Comment'
> **注意：在 Linux 系统中，commit 信息使用单引号 '，Windows 系统，commit 信息使用双引号 "。**

这样你的更改就被提交到远端的仓库了

常见的git 命令

增加删除文件

> 添加指定文件到暂存区
> git add [file1] [file2] ...
> 添加指定目录到暂存区，包括子目录
> git add [dir]
> 添加当前目录的所有文件到暂存区
> git add .
> 添加每个变化前，都会要求确认
> 对于同一个文件的多处变化，可以实现分次提交
> git add -p
> 删除工作区文件，并且将这次删除放入暂存区
> git rm [file1] [file2] ...
> 停止追踪指定文件，但该文件会保留在工作区
> git rm --cached [file]
> 改名文件，并且将这个改名放入暂存区
> git mv [file-original] [file-renamed]

代码提交

> 提交暂存区到仓库区
> git commit -m [message]
> 提交暂存区的指定文件到仓库区
> git commit [file1] [file2] ... -m [message]
> 提交工作区自上次commit之后的变化，直接到仓库区
> git commit -a
> 提交时显示所有diff信息
> git commit -v

> 使用一次新的commit，替代上一次提交
> 如果代码没有任何新变化，则用来改写上一次commit的提交信息
> git commit --amend -m [message]
> 重做上一次commit，并包括指定文件的新变化
> git commit --amend [file1] [file2] ……

远程同步

> 下载远程仓库的所有变动
> git fetch [remote]
> 显示所有远程仓库
> git remote -v
> 显示某个远程仓库的信息
> git remote show [remote]
> 增加一个新的远程仓库，并命名
> git remote add [shortname] [url]
> 取回远程仓库的变化，并与本地分支合并
> git pull [remote] [branch]
> 上传本地指定分支到远程仓库
> git push [remote] [branch]
> 强行推送当前分支到远程仓库，即使有冲突
> git push [remote] --force
> 推送所有分支到远程仓库
> git push [remote] --all

参考文档

git官方文档
菜鸟教程