关于 Git

了解版本控制系统 Git 以及它如何与 GitHub 配合使用。

本文内容

关于版本控制和 Git

版本控制系统（VCS）跟踪人员和团队在项目上进行协作时的更改历史记录。 当开发人员对项目进行更改时，可以随时恢复项目的任何早期版本。

开发人员可以查看项目历史记录以找出：

- 进行了哪些更改?
- 谁进行了更改?
- 何时进行了更改?
- 为什么需要更改?

VCS 为每个贡献者提供统一且一致的项目视图，显示已经在进行中的工作。 查看透明的更改历史记录、谁进行了更改，以及它们如何为项目开发做出贡献，可帮助团队成员在独立工作时保持一致。

在分布式版本控制系统中，每个开发人员都有项目和项目历史记录的完整副本。 与曾经流行的集中式版本控制系统不同，DVCS 不需要与中央存储库的持续连接。 Git 是最流行的分布式版本控制系统。 Git 通常用于开源和商业软件开发，对个人、团队和企业都有明显的好处。

- Git 允许开发人员在一个地方查看任何项目的更改、决策和进度的整个时间线。 从他们访问项目历史记录的那一刻起，开发人员就拥有了理解它并开始参与所需的所有上下文。

- 开发人员在每个时区工作。 使用像 Git 这样的 DVCS，协作可以随时随地进行，同时保持源代码的完整性。 使用分支，开发人员可以安全地提出对生产代码的更改建议。
- 使用 Git 的企业可以打破团队之间的沟通障碍，让他们专注于做好最好的工作。 此外，Git 还可以让整个企业的专家协调一致，在重大项目上进行协作。

关于仓库

存储库或 Git 项目包含与项目关联的文件和文件夹的整个集合，以及每个文件的修订历史记录。 文件历史记录在时间上显示为快照，称为提交。 提交可以组织成多个开发行，称为分支。 由于 Git 是 DVCS，因此存储库是独立的单元，任何拥有存储库副本的人都可以访问整个代码库及其历史记录。 使用命令行或其他易用性接口，Git 存储库还允许：与历史记录交互、克隆存储库、创建分支、提交、合并、比较不同版本的代码更改等。

通过像 GitHub 这样的平台，Git 也为项目的透明度和协作提供了更多的机会。 公共存储库可帮助团队协同工作，以构建最佳的最终产品。

GitHub 的工作原理

GitHub 托管 Git 存储库，并为开发人员提供工具，通过命令行功能、议题（线程讨论）、拉取请求、代码审查或使用 GitHub Marketplace 中的一组免费和可购应用来交付更好的代码。 通过 GitHub 流程等协作层、拥有 1 亿开发人员的社区以及具有数百个集成的生态系统，GitHub 改变了软件的构建方式。

GitHub 将协作直接构建到开发过程中。 工作组织到存储库中，开发人员可以在其中概述要求或方向，并为团队成员设定期望。 然后，使用 GitHub 流程，开发人员只需创建一个分支来处理更新，提交更改以保存它们，打开拉取请求以建议和讨论更改，并在每个人都在同一页面上时合并拉取请求。 有关详细信息，请参阅“GitHub 流”。

有关 GitHub 计划和成本，请参阅 GitHub Pricing。 有关 GitHub Enterprise 与其他选项的比较信息，请参阅比较 GitHub 与其他 DevOps 解决方案。

GitHub 和命令行

基本 Git 命令

为使用 Git，开发人员使用特定命令来复制、创建、更改和合并代码。 这些命令可以直接从命令行执行，也可以使用 GitHub Desktop等应用程序执行。 以下是使用 Git 的一些常用命令：

- `git init` 初始化一个全新的 Git 存储库并开始跟踪现有目录。 它在现有目录中添加一个隐藏的子文件夹，该子文件夹包含版本控制所需的内部数据结构。

- `git clone` 创建远程已存在的项目的本地副本。 克隆包括项目的所有文件、历史记录和分支。
- `git add` 暂存更改。 Git 跟踪对开发人员代码库的更改，但有必要暂存更改并拍摄更改的快照，以将其包含在项目的历史记录中。 此命令执行暂存，即该两步过程的第一部分。 暂存的任何更改都将成为下一个快照的一部分，并成为项目历史记录的一部分。 通过单独暂存和提交，开发人员可以完全控制其项目的历史记录，而无需更改其编码和工作方式。
- `git commit` 将快照保存到项目历史记录中并完成更改跟踪过程。 简言之，提交就像拍照一样。 任何使用 `git add` 暂存的内容都将成为使用 `git commit` 的快照的一部分。
- `git status` 将更改的状态显示为未跟踪、已修改或已暂存。
- `git branch` 显示正在本地处理的分支。
- `git merge` 将开发线合并在一起。 此命令通常用于合并在两个不同分支上所做的更改。 例如，当开发人员想要将功能分支中的更改合并到主分支以进行部署时，他们会合并。
- `git pull` 使用远程对应项的更新来更新本地开发线。 如果队友已向远程上的分支进行了提交，并且他们希望将这些更改反映到其本地环境中，则开发人员将使用此命令。
- `git push` 使用本地对分支所做的任何提交来更新远程存储库。

有关详细信息，请参阅 Git 命令的完整参考指南。

示例：参与现有存储库

```
# download a repository on GitHub to our machine
# Replace `owner/repo` with the owner and name of the repository to clone
git clone https://github.com/owner/repo.git

# change into the `repo` directory
cd repo

# create a new branch to store any new changes
git branch my-branch

# switch to that branch (line of development)
git checkout my-branch

# make changes, for example, edit `file1.md` and `file2.md` using the text editor

# stage the changed files
git add file1.md file2.md

# take a snapshot of the staging area (anything that's been added)
git commit -m "my snapshot"
```

```
# push changes to github
git push --set-upstream origin my-branch
```

示例：启动新存储库并将其发布到 GitHub

首先，你需要在 GitHub 上创建一个新的存储库。 有关详细信息，请参阅“Hello World”。 不要使用 README、.gitignore 或 License 文件初始化存储库。 这个空存储库将等待您的代码。

```
# create a new directory, and initialize it with git-specific functions
git init my-repo

# change into the `my-repo` directory
cd my-repo

# create the first file in the project
touch README.md

# git isn't aware of the file, stage it
git add README.md

# take a snapshot of the staging area
git commit -m "add README to initial commit"

# provide the path for the repository you created on github
git remote add origin https://github.com/YOUR-USERNAME/YOUR-REPOSITORY-NAME.git

# push changes to github
git push --set-upstream origin main
```

示例：为 GitHub 上的现有分支做出贡献

此示例假定计算机上已有一个名为 `repo` 的项目，并且自上次在本地进行更改以来，已将新分支推送到 GitHub。

```
# change into the `repo` directory
cd repo

# update all remote tracking branches, and the currently checked out branch
git pull

# change into the existing branch called `feature-a`
git checkout feature-a

# make changes, for example, edit `file1.md` using the text editor

# stage the changed file
git add file1.md
```

```
# take a snapshot of the staging area
git commit -m "edit file1"

# push changes to github
git push
```

协作开发模型

人们在 GitHub上主要有两种协作方式：

1. 共享存储库
2. 复刻和拉取

使用共享存储库，个人和团队被显式指定为具有读取、写入或管理员访问权限的参与者。 这种简单的权限结构与受保护的分支等功能相结合，可帮助团队在采用 GitHub 时快速取得进展。

对于开源项目，或者对于任何人都可以参与的项目，管理个人权限可能具有挑战性，但复刻和拉取模型允许任何可以查看项目的人做出贡献。 复刻是开发人员个人帐户下项目的副本。 每个开发人员都可以完全控制他们的分支，并可以自由地实现修复或新功能。 在复刻中完成的工作要么保持独立，要么通过拉取请求返回到原始项目。 在那里，维护者可以在合并之前查看建议的更改。 有关详细信息，请参阅“参与项目”。