**git笔记**

git是免费，开源的分布式版本控制软件系统

版本控制是一种记录文件内容变化，以便将来查阅特定版本修订情况的系统

版本控制其实最重要的是可以记录文件修改历史记录，从而让用户能够查看历史版本，方便版本切换

版本控制系统

本地版本控制系统

复制整个项目，好处是简单，坏处是容易混淆，写错文件后者覆盖意想外文件



RCS 的工作原理是在硬盘上保存补丁集（补丁是指文件修订前后的变化）；通过应用所有的补丁，可以重新计算出各个版本的文件内容

**集中化的版本控制系统CVCS**

CVS、Subversion 以及Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。

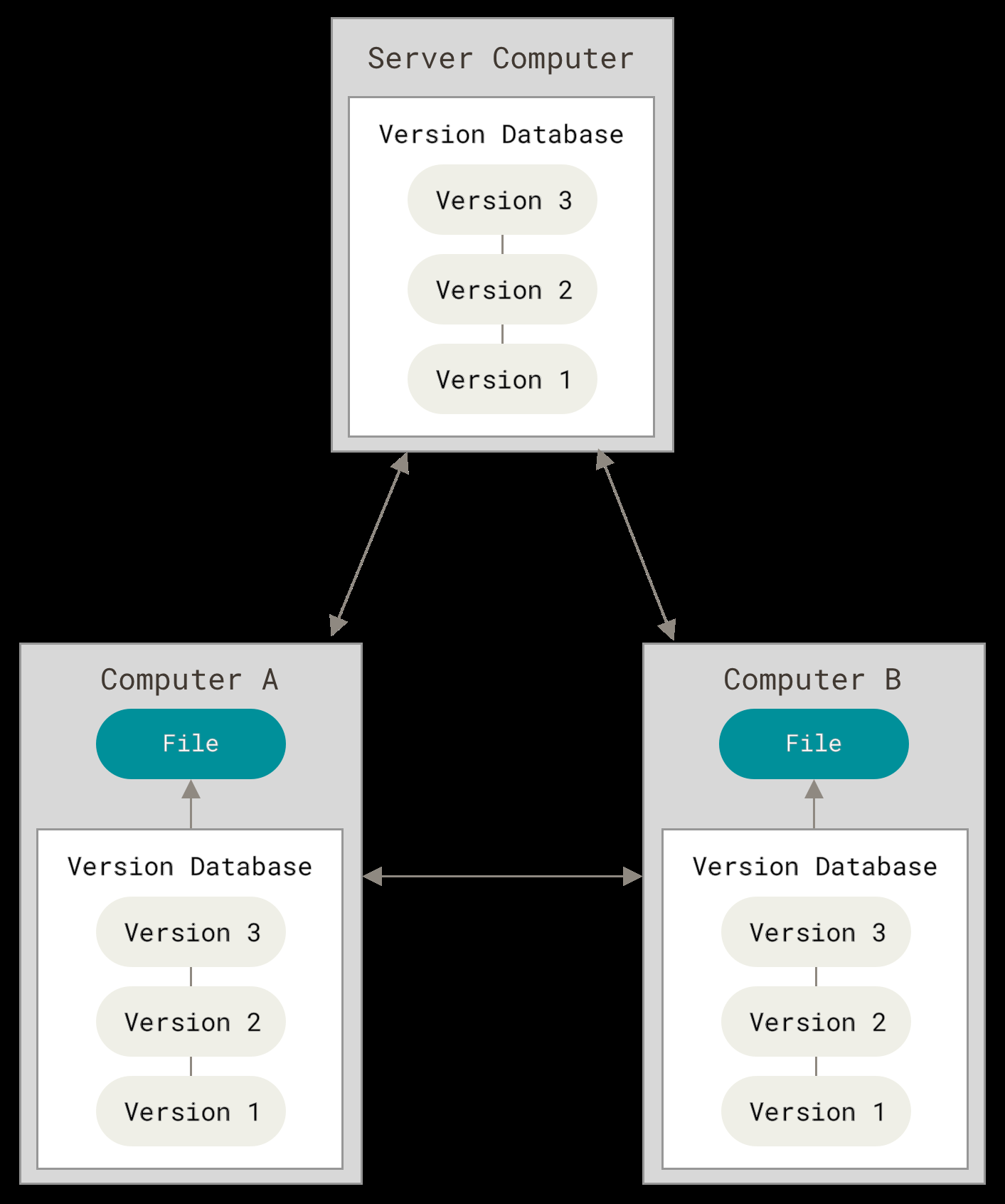


好处特别是相较于老式的本地 VCS 来说。 现在，每个人都可以在一定程度上看到项目中的其他人正在做些什么。 而管理员也可以轻松掌控每个开发者的权限，并且管理一个 CVCS 要远比在各个客户端上维护本地数据库来得轻松容易。

缺点是中央服务器的单点故障,只要整个项目的历史记录被保存在单一位置，就有丢失所有历史更新记录的风险。

**分布式版本控制系统DVCS**

Git、Mercurial、Bazaar 以及 Darcs 等客户端并不只提取最新版本的文件快照， 而是把代码仓库完整地镜像下来，包括完整的历史记录。 这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。 因为每一次的克隆操作，实际上都是一次对代码仓库的完整备份。



许多这类系统都可以指定和若干不同的远端代码仓库进行交互。籍此，你就可以在同一个项目中，分别和不同工作小组的人相互协作。 你可以根据需要设定不同的协作流程，比如层次模型式的工作流，而这在以前的集中式系统中是无法实现的。

直接记录快照，而非差异比较

Git 保证完整性

Git 中所有的数据在存储前都计算校验和，然后以校验和来引用。 这意味着不可能在 Git 不知情时更改任何文件内容或目录内容。 这个功能建构在 Git 底层，是构成 Git 哲学不可或缺的部分。 若你在传送过程中丢失信息或损坏文件，Git 就能发现。

Git 用以计算校验和的机制叫做 **SHA-1 散列（hash，哈希）**。 这是一个**由40个十六进制字符（0-9 和 a-f）组成的字符串**，基于Git中文件的内容或目录结构计算出来。 SHA-1哈希看起来是这样：24b9da6552252987aa493b52f8696cd6d3b00373

Git 中使用这种哈希值的情况很多，你将经常看到这种哈希值。 实际上，Git 数据库中保存的信息都是以文件内容的哈希值来索引，而不是文件名。

初次运行git前的配置（变量）

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。）

每一个级别会覆盖上一级别的配置，所以 .git/config 的配置变量会覆盖 /etc/gitconfig 中的配置变量。

在 Windows 系统中，Git 会查找 $HOME 目录下（一般情况下是 C:\Users\$USER ）的 .gitconfig 文件。

Git 同样也会寻找 /etc/gitconfig 文件，但只限于 MSys 的根目录下，即安装 Git 时所选的目标位置。 如果你在 Windows 上使用 Git 2.x 以后的版本，那么还有一个系统级的配置文件，Windows XP 上在C:\Documents and Settings\All Users\Application Data\Git\config ，Windows Vista 及更新的版本在 C:\ProgramData\Git\config 。此文件只能以管理员权限通过 git config -f <file> 来修改。你可以通过以下命令查看所有的配置以及它们所在的文件：

$ git config --list --show-origin

查看版本信息

git -v

git –version

git区域分为三部分

工作区：人为编辑

暂存区：比对存储区域和工作区域的区域

存储区：用于存储资源得区域。一般指得就是.git文件夹

Git配置文件安装路径：../etc/gitconfig

查看配置信息

git config -l

名称和邮箱: Git软件必须区分不同用户的操作，区分的方式就是名称和邮箱

git config --global user.name daviesdeng

git config --global user.email [daviesdeng@google.com](mailto:daviesdeng@google.com)

--global为全局配置，配置文件位置：C:\Users\Administrator

初始化版本库

git init 版本库创建成功后，会创建.git目录，用于管理版本库，用此方法注册的主分支是master，而用克隆注册的主分支是main

查看版本库状态

git status

untracked files(未追踪文件)，文件显示为红色

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (master)

$ git status

On branch master 当前分支为master

No commits yet 尚未提交

**Untracked files**: 未追踪文件

(use "git add <file>..." to include in what will be committed)

a.txt

nothing added to commit but untracked files present (use "git add" to track)

没有添加任何要提交的内容，但存在未跟踪的文件（使用“git add”进行跟踪）

Git提交到暂存区

git add file 提交单独文件

git add \* 提交所有文件

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (master)

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --**cached <file>**..." to unstage)

new file: a.txt

cached file是暂存文件，还没有提交到存储区

git提交到存储区

git commit -m "日志信息" 文件名

# commit表示真正地纳入到版本库中

# -m 表示提交时的信息（日志信息message），是必须输入的。用于描述不同版本之间的差别信息

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (master)

$ git commit -m "first git file"

[master (root-commit) 63f6e17] first git file

1 file changed, 0 insertions(+), 0 deletions(-)

**1个文件已更改，0个插入（+），0个删除（-）**

create mode 100644 a.txt

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (master)

$ git status

On branch master

nothing to commit, working tree clean

提交后，Git会对当前的操作进行Hash计算，通过计算后的值将数据保存下来，保存的位置为版本库.git文件目录的objects中，我们可以通过指令查看当前提交

修改版本库文件

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: a.txt

**modified表示文件已经修改**

直接提交到存储区

git commit -a -m "update file"

查看版本库文件历史

Git reflog

git log

git log –graph 操作日志

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (master)

$ git log

commit cf87e62518a16f74d30ff6f3f712ea125080bd3e (HEAD -> master)

Author: daviesdeng <daviesdeng@gmail.com>

Date: Sat Apr 22 11:13:31 2023 +0800

Second file modification

commit 63f6e17d56ee0115020cc6e4e6d7b191abd2f19c

Author: daviesdeng <daviesdeng@gmail.com>

Date: Sat Apr 22 10:35:15 2023 +0800

first git file

从下向上是从第一版到最新版

显示美化

git log --pretty=oneline

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (master)

$ git log --pretty=oneline

f30ecfc63db33b8185aa46c36c5595aed3063ea1 (HEAD -> master) 第三次文件修改

cf87e62518a16f74d30ff6f3f712ea125080bd3e Second file modification

63f6e17d56ee0115020cc6e4e6d7b191abd2f19c first git file

简单显示

git log --oneline

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (master)

$ git log --oneline

f30ecfc (HEAD -> master) 第三次文件修改

cf87e62 Second file modification

63f6e17 first git file

恢复删除文件

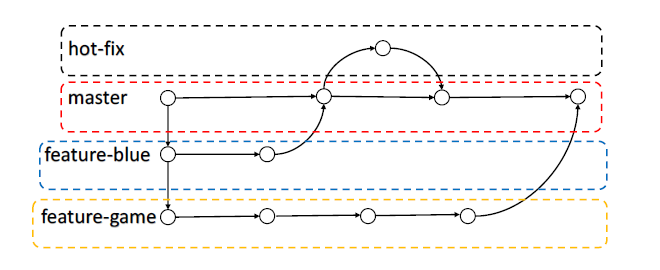
git restore \* 恢复所有文件

版本穿梭

git reset –hard 版本号

分支介绍

同时推进多个任务，为每个任务，我们就可以创建每个任务的单独分支。使用分支意味着程序员可以把自己的工作从开发主线上分离开来，开发自己分支的时候，不会影响主线分支的运行。分支可以简单理解为副本，一个分支就是一个单独的副本。



好处：同时并行推进多个功能开发，提高开发效率。

各个分支在开发过程中，如果某一个分支开发失败，不会对其他分支有任何影响。失败的分支删除重新开始即可。

分支命令

查看分支 git branch -v

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (hot-fix)

$ git branch -v

\* hot-fix f30ecfc 第三次文件修改 \*在哪显示，那个为当前分支

master f30ecfc 第三次文件修改

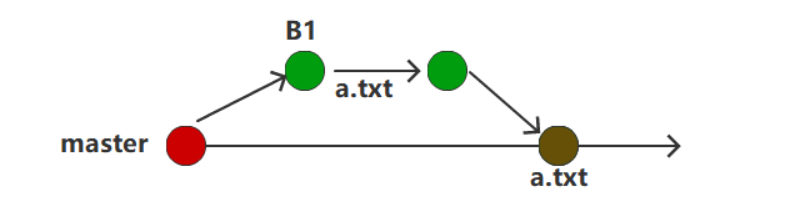
创建分支 git branch 分支名

切换分支 git checkout 分支名

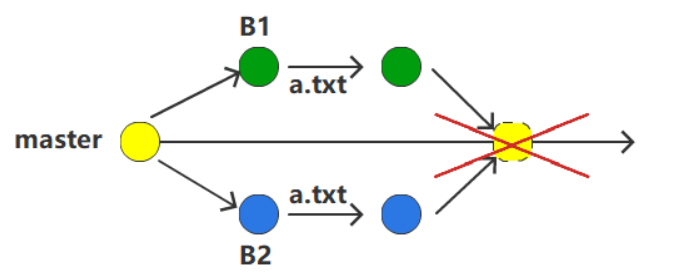
在主干分支创建分支并切换 git checkout -b new\_branch

删除分支 git branch -d 分支名

合并分支 git merge 分支名



产生冲突



冲突产生的表现：后面状态为MERGING

Administrator@DESKTOP-5IBDOLB MINGW64 /e/git/test/gittest (hot-fix)(master| MERGING）

<<<<<<< HEAD

hello git! hello atguigu! master test

hello git! hello atguigu!

=======

hello git! hello atguigu!

hello git! hello atguigu! Hot-fix test

>>>>>>> hot-fix

冲突产生的原因：

合并分支时，两个分支在同一个文件的同一个位置有两套完全不同的修改。Git无法替我们决定使用哪一个。必须人为决定新代码内容。

解决冲突

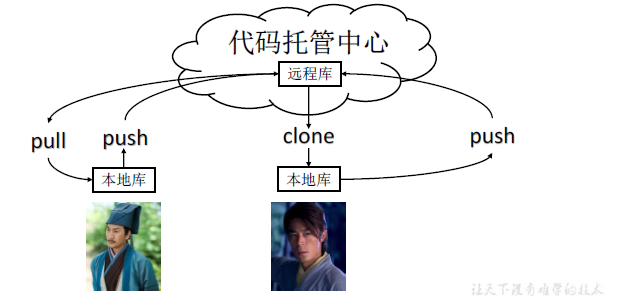
编辑有冲突的文件，删除特殊符号，决定要使用的内容

特殊符号：<<<<<<< HEAD 当前分支的代码======= 合并过来的代码>>>>>>> hot-fix

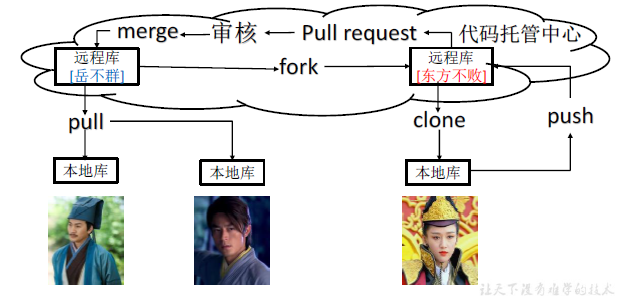
此时提交存储区时不可以有文件名

团队协作机制

团队内协作



跨团队协作



Github

官网：<https://github.com/>

创建新仓库

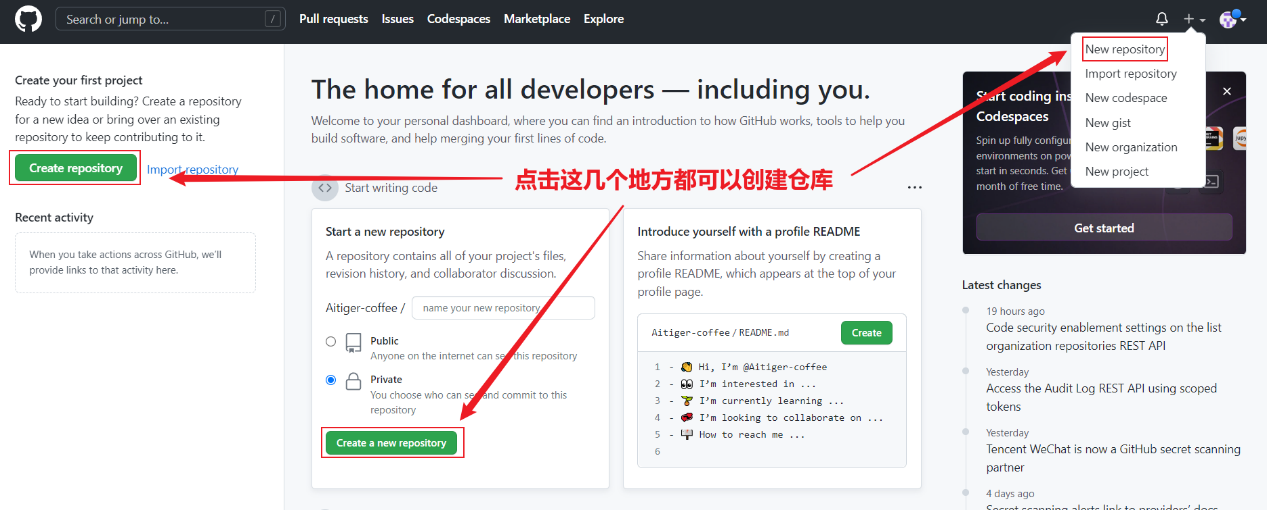
New repository:创建远程仓库

Import repository导入存储库

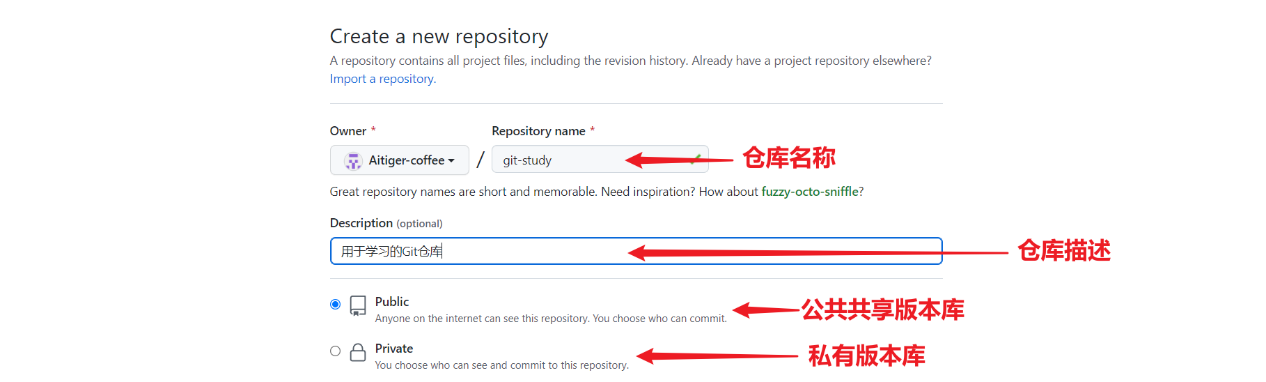
New codespace新建代码空间

New gist新的要点

New organization新组织



输入仓库的相关信息



远程仓库操作

查看当前所有远程地址别名 git remote -v

起别名 git remote add 别名 远程地址

推送本地分支上的内容到远程仓库 git push 别名 分支

克隆到本地 git clone 远程地址

将远程仓库对于分支最新内容拉下来后与当前本地分支直接合并 git pull 远程库地址别名 远程分支名