

Learn Git from zero

作者：徐子霖

时间：2023.05.23

参考：字节第五届后端青训营

为什么要学习git

- 协同工作
- 开源社区

为什么要设计这门课程

- 入职后按照git配置，拉取代码有问题，缺少自己排查配置问题的能力
- 一些异常操作，不符合研发规范，不清楚保护分支

课程目标

- 学会正确使用git

1. Git 是什么

分布式版本控制系统

版本控制是什么：

一种记录一个或若干文件内容的变化

为什么要版本控制

更好关注变更

及时回滚

- 本地版本控制
 - 最初的方式
 - 通过本地复制文件夹
 - 缺点：本地
- 集中式版本控制
 - 代表工具：SVN
 - 原理：增量保存每次提交的Diff
 - 缺点
 - 本地不会保存
 - 分支不够友好
 - 容易导致历史版本丢失
- 分布式版本控制
 - 原理
 - 每个库都有完整的提交历史
 - 每次提交记录都是完整的文件快照
 - 通过push等操作来完成和远端的同步

- 缺点
 - 学习成本更高
 - 对大文件不是很友好

1. git发展历史

作者：Linux的作者

开发原因

Bitkeeper不允许Linux团队继续无偿使用

开发时间

两周

github：全球最大的代码托管平台

gitlab：全球最大的开源代码托管平台

gerrit：Google开发的一个代码托管平台，安卓这个项目就托管在Gerrit之上

Git的基本使用方式

git命令

1. 配置
2. 提交代码
3. 远端同步
 - 拉取代码
 - 推送代码

常见问题

1. 为什么我配置了git，但依然没办法拉取代码
 - 配置权限的问题

项目初始化 `git init`：`--initial-branch` 初始化的分支 `--bare` 创建一个裸仓库

add和commit：存在暂存区

命令

```
mkdir study cd study ls git init
```

```
initialized existing Git repository in /Users/xzl/Desktop/study/.git/
```

```
(base) xzl@ersandeMacBook-Pro study % tree .git
.git
├── HEAD
├── config
├── description
└── hooks
    ├── applypatch-msg.sample
    ├── commit-msg.sample
    ├── fsmonitor-watchman.sample
    ├── post-update.sample
    ├── pre-applypatch.sample
    ├── pre-commit.sample
    ├── pre-merge-commit.sample
    ├── pre-push.sample
    ├── pre-rebase.sample
    ├── pre-receive.sample
    ├── prepare-commit-msg.sample
    ├── push-to-checkout.sample
    └── update.sample
├── info
│   └── exclude
└── objects
    ├── info
    └── pack
└── refs
    ├── heads
    └── tags
```

9 directories, 17 files

```
(base) xzl@ersandeMacBook-Pro study % cat .git/HEAD
ref: refs/heads/main
```

tree .git

tree .git 命令用于以树形结构显示当前目录下的 .git 文件夹及其内容。在版本控制系统 Git 中，.git 文件夹是用来存储仓库的所有历史记录、分支、标签和配置信息的地方。通过运行 tree .git 命令，您可以获得一个清晰的视图，显示 .git 文件夹中的子目录和文件，以及它们之间的层次关系。

cat .git/HEAD

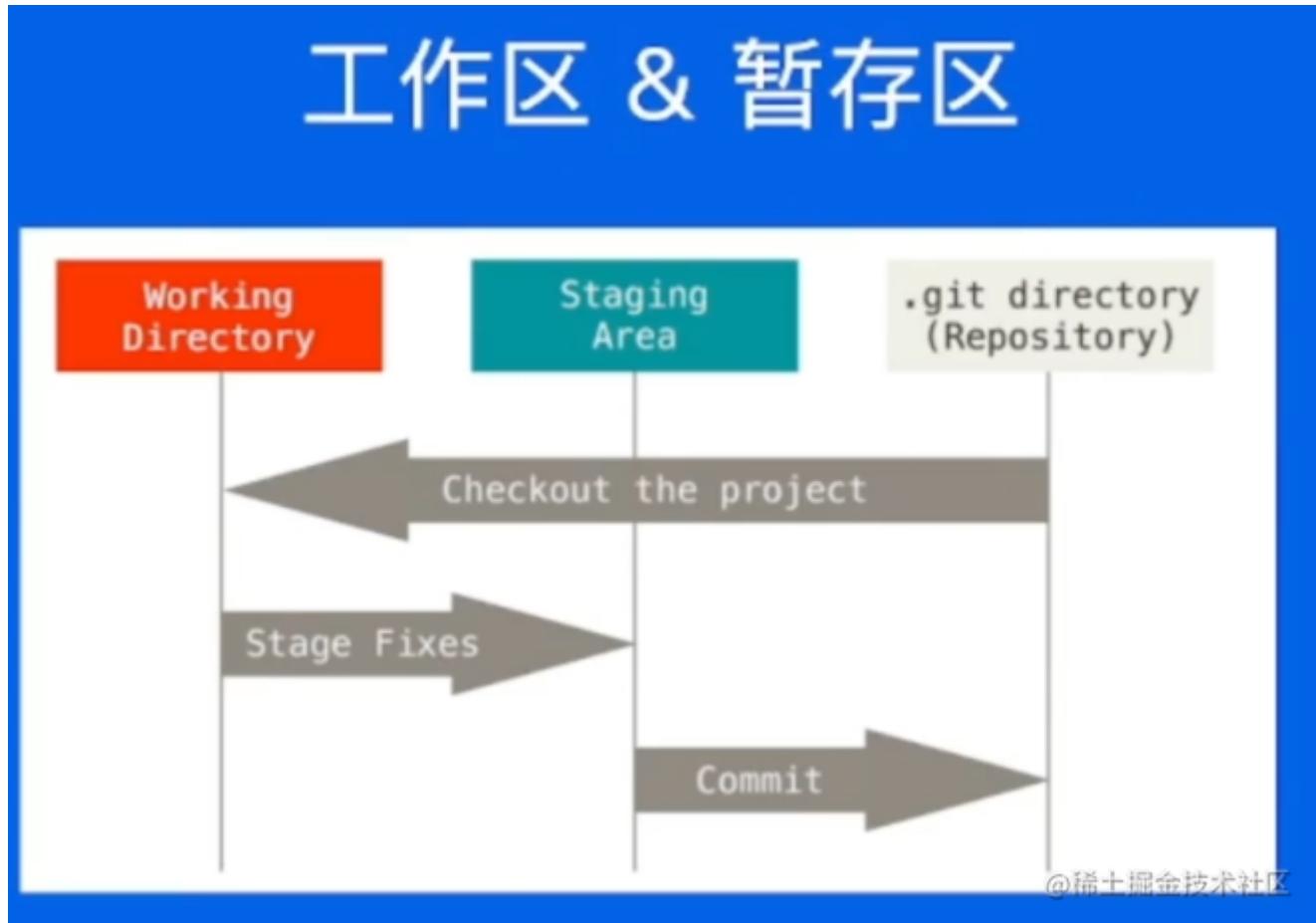
显示当前在的分支是main分支

config 一些配置

object 一些文件信息

refs 分支信息

后面的`操作都会修改这些目录里的内容`



工作区通过`add`把内容放到暂存区

我们的暂存区和最后通过`commit`提交其实都在我们刚才看到的`git`目录里

2.1.1 Git config

三种类型: 全局, 系统, 本地

--global	use global config file
--system	use system config file
--local	use repository config file

每个级别的配置可能重复, 但是低级别的配置会覆盖高级别的配置

系统 > 全局 > 本地

2.1.2 常见git配置（重要）

- 用户名配置

```
git config --global user.name "xxxx"
```

这个命令是用来配置Git的全局用户名（user.name）。当你在Git提交代码时，Git会记录下提交者的信息，包括用户名和电子邮件地址。通过设置全局用户名，你可以指定你在提交代码时所使用的用户名。

- Instead of 配置
- 别名

2.2 git remote和远端

查看 `git remote -v`

添加 `git remote add origin_ssh git@github.com:git/git.git`

这个命令是用来将一个远程仓库与你的本地仓库关联起来。它通过一个简短的名字（origin_ssh）来代表远程仓库的地址。在这个命令中，`origin_ssh`是一个自定义的远程仓库名称，你可以根据需要进行修改。`git@github.com:git/git.git`是远程仓库的地址，它遵循SSH协议的格式。具体来说，`git@github.com`是GitHub的主机名，它指示你将要将你的本地仓库关联到GitHub上。
`git/git.git`是远程仓库的路径，它指示你要关联的具体仓库。这个路径可以因你要关联的仓库而有所不同。执行这个命令后，你的本地仓库就会和远程仓库建立起连接。这样你就可以使用Git命令来推送（push）和拉取（pull）代码到远程仓库，并与其他开发者进行协作。通常情况下，你只需要执行这个命令一次，以后就可以使用关联的远程仓库进行代码交互。

```
git remote add origin_http git@github.com:git/git.git
```

```
(base) xzl@ersandeMacBook-Pro study % git remote -v
(base) xzl@ersandeMacBook-Pro study % git remote add origin_ssh git@github.com:git/git.git
(base) xzl@ersandeMacBook-Pro study % git remote add origin_http git@github.com:git/git.git
(base) xzl@ersandeMacBook-Pro study % git remote -v
origin_http      git@github.com:git/git.git (fetch)
origin_http      git@github.com:git/git.git (push)
origin_ssh       git@github.com:git/git.git (fetch)
origin_ssh       git@github.com:git/git.git (push)
(base) xzl@ersandeMacBook-Pro study % cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    ignorecase = true
    precomposeunicode = true
[remote "origin_ssh"]
    url = git@github.com:git/git.git
    fetch = +refs/heads/*:refs/remotes/origin_ssh/*
[remote "origin_http"]
    url = git@github.com:git/git.git
    fetch = +refs/heads/*:refs/remotes/origin_http/*

```

@稀土掘金技术社区

同一个origin设置不同的push和fetch url

比如说我希望从远端仓库拉取代码然后推送到我自己的仓库里

```
git remote add origin git@github.com:git/git.git
git remote set-url --add --push origin git@github.com:my_repo/git.git
[remote "origin"]
  url = git@github.com:git/git.git
  fetch = +refs/heads/*:refs/remotes/origin/*
[remote "origin_ssh"]
  url = git@github.com:git/git.git
  fetch = +refs/heads/*:refs/remotes/origin_ssh/*
[remote "origin_http"]
  url = git@github.com:git/git.git
  fetch = +refs/heads/*:refs/remotes/origin_http/*
[remote "my_repo"]
  url = git@github.com:my_repo/git.git
  pushurl = git@github.com:my_repo/git.git
(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

也可以直接用[vim](#)去改

2.2.1 HTTP remote

URL: <https://github.com/git/git.git>

正常来说不推荐用http，没那么安全，还是会使用ssh

2.2.2 SSH remote

URL: <git@github.com:git/git.git>

两种配置都有免密配置

ssh: 优先使用ed25519

```
(base) xzl@ersandeMacBook-Pro study % ssh-keygen -t ed25519 -C "zx112@duke.edu"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/xzl/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/xzl/.ssh/id_ed25519
Your public key has been saved in /Users/xzl/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:cdq91aDA0oD1UTUTp9fe0yEE3S23hZeStap2DITX3nE zx112@duke.edu
The key's randomart image is:
+---[ED25519 256]---+
|   . . . oBo=oo |
|   o . o. oOo+* |
|   . . o.+o.o+BE |
|   . *oo.o+=* |
|   S ..oo.o= |
|   . +o   . |
|   o.o   |
|   . .   |
+---[SHA256]---+
(base) xzl@ersandeMacBook-Pro study % cat /Users/xzl/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1ZD1NTE5AAAIKEr3jUUpfNLDJ8AnXbrbBiWvgAFVaIewFCB6RnDyzm3 zx112@duke.edu
(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

可以把这个public的key复制出来存到github上

当你在GitHub上添加新的SSH密钥时，它会提供以下几个作用：

1. 认证与授权：通过将SSH公钥添加到你的GitHub帐户中，你可以使用私钥进行身份验证，以证明你是该帐户的所有者。这样，你就可以执行与你帐户相关的操作，如克隆、推送和拉取代码等。
2. 安全连接：SSH密钥通过使用公钥加密和私钥解密的方式，确保了数据在传输过程中的安全性。当你使用SSH协议与GitHub进行通信时，数据将被加密，以防止未经授权的访问或数据篡改。
3. 简化访问：一旦你将SSH密钥添加到GitHub中，你就可以通过SSH协议轻松地与GitHub进行通信，而无需每次都输入用户名和密码。这样可以提高工作效率并提供更方便的访问方式。
4. 多设备共享：通过将SSH密钥添加到GitHub中，你可以在多个设备上轻松地访问和管理你的代码库。这意味着你可以在不同的计算机或服务器上使用私钥，而无需每次都生成新的密钥对。

总之，通过添加新的SSH密钥，你可以在GitHub上进行身份验证、建立安全连接、简化访问并方便地在多个设备上共享代码。这为你提供了更方便、更安全的代码管理和协作环境。

2.3 git add

这里我们新建一个文档，然后我们看git发现没什么变化

```
((base) xzl@ersandeMacBook-Pro study % ls
((base) xzl@ersandeMacBook-Pro study % touch readme.md
((base) xzl@ersandeMacBook-Pro study % tree .git
.git
├── HEAD
├── config
├── description
└── hooks
    ├── applypatch-msg.sample
    ├── commit-msg.sample
    ├── fsmonitor-watchman.sample
    ├── post-update.sample
    ├── pre-applypatch.sample
    ├── pre-commit.sample
    ├── pre-merge-commit.sample
    ├── pre-push.sample
    ├── pre-rebase.sample
    ├── pre-receive.sample
    ├── prepare-commit-msg.sample
    └── push-to-checkout.sample
        └── update.sample
├── info
│   └── exclude
└── objects
    ├── info
    └── pack
└── refs
    ├── heads
    └── tags

9 directories, 17 files
```

@稀土掘金技术社区

这是`git add`之前和之后的`git status`的变化

```
11 directories, 1 files
(base) xzl@ersandeMacBook-Pro study % git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md

nothing added to commit but untracked files present (use "git add" to track)
(base) xzl@ersandeMacBook-Pro study % git add .
(base) xzl@ersandeMacBook-Pro study % git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.md

(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

然后我们对readme进行修改，添加一行hello world进去，再add

就可以用`git cat-file -p <object id>`来看出区别

```
update sample
├── index
├── info
│   └── exclude
└── objects
    ├── 3b
    │   └── 18e512dba79e4c8300dd08aeb37f8e728b8dad
    ├── e6
    │   └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
    └── info
        └── pack
    └── refs
        ├── heads
        └── tags

11 directories, 20 files
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 3b18e512dba79e4c8300dd08aeb37f8e728b8dad
hello world
(base) xzl@ersandeMacBook-Pro study % git cat-file -p e69de29bb2d1d6434b8b29ae775ad8c2e48c5391
(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

2.4 git commit

`git commit -m "add readme"` 再观察一下.git这个目录

```

    └── main
    ├── objects
    │   ├── 3b
    │   │   └── 18e512dba79e4c8300dd08aeb37f8e728b8dad
    │   ├── 62
    │   │   └── fd81834f7034844fb7fd4ea10f5fb25d39221b
    │   ├── 73
    │   │   └── 94b8cc9ca916312a79ce8078c34b49b1617718
    │   ├── e6
    │   │   └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        │   └── main
        └── tags

16 directories, 26 files
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 3b18e512dba79e4c8300dd08aeb37f8e728b8dad
hello world
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 62fd81834f7034844fb7fd4ea10f5fb25d39221b
tree 7394b8cc9ca916312a79ce8078c34b49b1617718
author zilin xu <zx112@duke.edu> 1684738716 +0800
committer zilin xu <zx112@duke.edu> 1684738716 +0800

add readme
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 7394b8cc9ca916312a79ce8078c34b49b1617718
100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad    readme.md
(base) xzl@ersandeMacBook-Pro study % git cat-file -p e69de29bb2d1d6434b8b29ae775ad8c2e48c5391
(base) xzl@ersandeMacBook-Pro study %

```

为什么会多两个文件呢

blob (目录树)

在Git中，Blob（Binary Large Object）对象用来表示文件内容的快照。每个文件在Git中都被存储为一个独立的Blob对象。

Blob对象保存着文件的数据，它们是Git存储中的基本数据单元。每个Blob对象都有一个唯一的哈希值，用于在Git中进行引用和查找。

Blob对象的作用包括：

1. 文件内容存储：Blob对象保存了文件的实际内容。它们充当了文件的快照，记录了特定版本的文件数据。
2. 内容比较和查找：Git使用Blob对象来比较文件的内容是否发生了更改，从而确定哪些部分需要在提交中进行保存。Blob对象的哈希值还用于快速查找特定文件版本。
3. 文件版本控制：Blob对象是Git进行版本控制的基础。通过记录文件内容的不同版本的Blob对象，Git可以轻松地进行版本回溯、分支合并等操作。

总之，Blob对象在Git中扮演着关键的角色，用于存储和管理文件的内容。它们使得Git能够高效地跟踪文件的修改历史，并提供强大的版本控制功能。

commit信息

还有一个可以结合git log一起看

```
(base) xzl@ersandeMacBook-Pro study % git log  
commit 62fd81834f7034844fb7fd4ea10f5fb25d39221b (HEAD -> main)  
Author: zilin xu <zx112@duke.edu>  
Date: Mon May 22 14:58:36 2023 +0800
```

```
    add readme
```

```
(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

2.5 objects

commit/ tree / blob 在git里面统称为object，除此之外还有个tag的object

- blob: 存储文件内容
- tree: 存储文件的目录信息
- commit: 提交信息

如何把这三个信息串联在一起呢？

1. 通过commit寻找tree信息，每个commit都会存储对应的tree id
2. 通过tree存储的信息，获取对应的目录树信息（可能存在多个blob）
3. 从tree中获得blob的id，通过blob id获取对应的文件内容

如下所示：

```
(base) xzl@ersandeMacBook-Pro study % git log  
commit 62fd81834f7034844fb7fd4ea10f5fb25d39221b (HEAD -> main)  
Author: zilin xu <zx112@duke.edu>  
Date: Mon May 22 14:58:36 2023 +0800
```

```
    add readme
```

```
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 62fd81834f7034844fb7fd4ea10f5fb25d39221b  
tree 7394b8cc9ca916312a79ce8078c34b49b1617718  
author zilin xu <zx112@duke.edu> 1684738716 +0800  
committer zilin xu <zx112@duke.edu> 1684738716 +0800
```

```
add readme
```

```
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 7394b8cc9ca916312a79ce8078c34b49b1617718  
100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad readme.md
```

```
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 3b18e512dba79e4c8300dd08aeb37f8e728b8dad  
hello world
```

```
(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

2.6 refs

刚才我们的操作也修改了ref目录下面的内容

```

        └── main
objects
└── 3b
    └── 18e512dba79e4c8300dd08aeb37f8e728b8dad
└── 62
    └── fd81834f7034844fb7fd4ea10f5fb25d39221b
└── 73
    └── 94b8cc9ca916312a79ce8078c34b49b1617718
└── e6
    └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
└── info
└── pack
refs
└── heads
    └── main
└── tags

```

16 directories, 26 files
(base) xzl@ersandeMacBook-Pro study % cat .git/refs/heads/main
62fd81834f7034844fb7fd4ea10f5fb25d39221b
(base) xzl@ersandeMacBook-Pro study %

@稀土掘金技术社区

显示了我们的commit id

新建一个分支，用`git checkout -b test`，我们会发现和main分支的内容是一样的

```

└── info
└── pack
refs
└── heads
    └── main
        └── test
└── tags

```

16 directories, 28 files
(base) xzl@ersandeMacBook-Pro study % cat .git/refs/heads/test
62fd81834f7034844fb7fd4ea10f5fb25d39221b
(base) xzl@ersandeMacBook-Pro study %

@稀土掘金技术社区

ref文件储存的内容

内容就是对应的commit id 因此把ref当做指针，指向对应的commit来表示当前ref对应的版本

不同种类的ref

- ref/heads 表示的是分支
- ref/tags 表示的是标签

branch

用于开发阶段，可以不断添加commit进行迭代，是会变化的

tag

表示的是一个稳定的版本，指向的commit一般不变

`git tag v0.0.1`

```

main
└── test
    └── v0.0.1
tags
    └── v0.0.1

16 directories, 29 files
(base) xzl@ersandeMacBook-Pro study % cat .git/refs/tags/v0.0.1
62fd81834f7034844fb7fd4ea10f5fb25d39221b
(base) xzl@ersandeMacBook-Pro study %

```

@稀土掘金技术社区

2.7 annotation tag

一种特殊的tag，可以给tag提供一些额外的信息

```
git tag -a v0.0.2 -m "add feature 1"
```

```

study -- zsh -- 109x32
objects
├── 3b
│   └── 18e512dba79e4c8300dd08aeb37f8e728b8dad
├── 54
│   └── 0fade41216d017ac3103d21203fd951754adaf
├── 62
│   └── fd81834f7034844fb7fd4ea10f5fb25d39221b
├── 73
│   └── 94b8cc9ca916312a79ce8078c34b49b1617718
├── e6
│   └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
└── info
    └── pack
refs
└── heads
    ├── main
    └── test
└── tags
    ├── v0.0.1
    └── v0.0.2

17 directories, 31 files
(base) xzl@ersandeMacBook-Pro study % cat .git/refs/tags/v0.0.2
540fade41216d017ac3103d21203fd951754adaf
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 540fade41216d017ac3103d21203fd951754adaf
object 62fd81834f7034844fb7fd4ea10f5fb25d39221b
type commit
tag v0.0.2
tagger zilin xu <zx112@duke.edu> 1684740108 +0800

add feature 1
(base) xzl@ersandeMacBook-Pro study %

```

@稀土掘金技术社区

会发现它指向不同的object，并且有了一些额外的信息例如，是谁添加的tag

这里的object就是我们第四种object

追溯历史版本

获取当前版本代码

通过ref指向的commit可以获取唯一的代码版本

获取历史版本代码

commit里面会存有parent commit字段，通过commit的串联获取历史版本代码

回到我们刚才的md文件，修改文件添加一个#号，再add和commit

我们会发现多了三个文件，一个blob，一个commit，一个tree

看一下：

```
(base) xzl@ersandeMacBook-Pro study % git log
commit 1bd6526eda494b7e3894ece6adde5a79bc0a5891 (HEAD -> test)
Author: zilin xu <zx112@duke.edu>
Date:   Mon May 22 15:29:03 2023 +0800

    update readme

commit 62fd81834f7034844fb7fd4ea10f5fb25d39221b (tag: v0.0.2, tag: v0.0.1, main)
Author: zilin xu <zx112@duke.edu>
Date:   Mon May 22 14:58:36 2023 +0800

    add readme
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 1bd6526eda494b7e3894ece6adde5a79bc0a5891
tree 337d6ffe2ac9db3392f9a4f07fa15414d67a6202
parent 62fd81834f7034844fb7fd4ea10f5fb25d39221b
author zilin xu <zx112@duke.edu> 1684740543 +0800
committer zilin xu <zx112@duke.edu> 1684740543 +0800

update readme
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 337d6ffe2ac9db3392f9a4f07fa15414d67a6202
100644 blob 759f3c31e352fe4917e59f548ec26b40e5771d4f      readme.md
(base) xzl@ersandeMacBook-Pro study % git cat-file -p 759f3c31e352fe4917e59f548ec26b40e5771d4f
#hello world
(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

我们会发现我们的blob的id变了，因为我们修改了文件，这个时候访问这个blob id对应的文件就会变成我们更新的文件

这个时候我们看我们的test ref也指向新的commit

```

└── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
    └── info
    └── pack
└── refs
    ├── heads
    │   └── main
    │       └── test
    └── tags
        └── v0.0.1
            └── v0.0.2

```

20 directories, 34 files

```
(base) xzl@ersandeMacBook-Pro study % cat .git/refs/heads/test
1bd6526eda494b7e3894ece6adde5a79bc0a5891
(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

2.9 修改历史版本

1. commit --amend

通过这个命令可以修改最近一次commit的信息，修改之后commit id会变

如下所示，我们改commit信息添加1111

```
(base) xzl@ersandeMacBook-Pro study % git commit --amend
[test d05ada1] update readme1111
Date: Mon May 22 15:29:03 2023 +0800
1 file changed, 1 insertion(+), 1 deletion(-)
(base) xzl@ersandeMacBook-Pro study % git log
commit d05ada1f11cbbababd378bd60694a2268b33c0b8 (HEAD -> test)
Author: zilin xu <zx112@duke.edu>
Date: Mon May 22 15:29:03 2023 +0800

update readme1111

commit 62fd81834f7034844fb7fd4ea10f5fb25d39221b (tag: v0.0.2, tag: v0.0.1, main)
Author: zilin xu <zx112@duke.edu>
Date: Mon May 22 14:58:36 2023 +0800

add readme
```

@稀土掘金技术社区

并且，操作新增了一个**commit object**，老的并没有删除

2. rebase

通过git rebase -i HEAD~3 可以实现对最近三个commit的修改：

- 合并 commit
- 修改具体的commit message
- 删除某个commit

3. filter --branch

该命令可以指定删除所有提交中的某个文件或者全局修改邮箱地址等操作

2.10 object

上述操作修改了ref指向的commit，但是之前的commit没有被删掉，所以我们引出一个概念叫**悬空object**

用git fsck --lost-found查看

```
(base) xzl@ersandeMacBook-Pro study % cat .git/refs/heads/test
d05ada1f11cbbababd378bd60694a2268b33c0b8
(base) xzl@ersandeMacBook-Pro study % git fsck --lost-found
Checking object directories: 100% (256/256), done.
dangling blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391
dangling commit 1bd6526eda494b7e3894ece6adde5a79bc0a5891
(base) xzl@ersandeMacBook-Pro study %
```

@稀土掘金技术社区

新的问题： 我们肯定想删掉这些多余的commit

2.11 git gc

因为我们发现，git的很多操作会新生成很多文件，git优化后我们可以处理掉

GC

用git gc命令，可以删除一些不需要的object 依旧会对object进行一些打包压缩来减少仓库的体积

reflog

reflog 适用于记录操作日志，防止误操作后数据丢失

通过reflog来找丢失的数据，手动将日志设置为过期

指定时间

git gc prune = now 指定的是修剪多久之前的对象，默认是两周前

```
git reflog expire --expire=now --all
```

```
git gc --prune=now
```

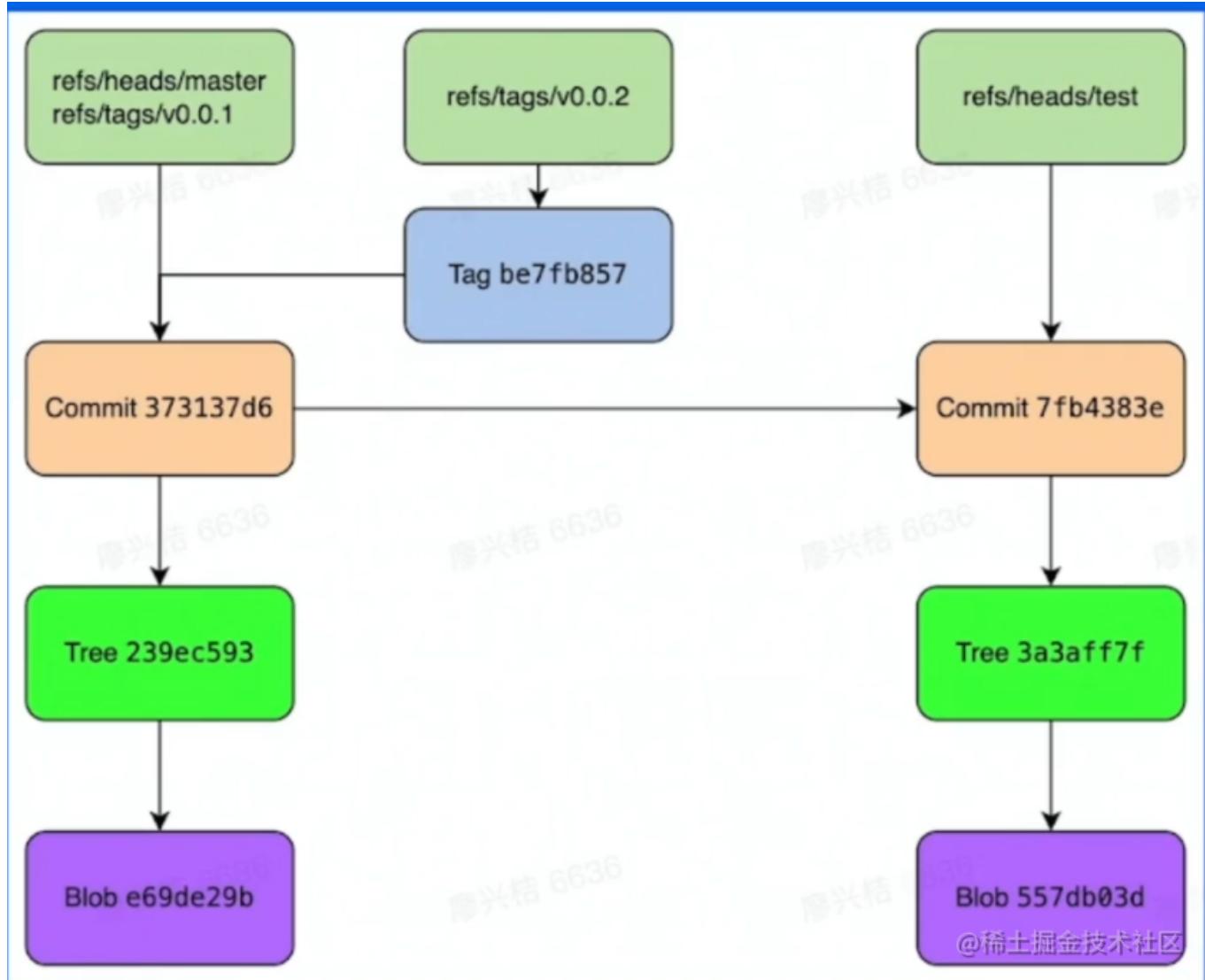
```
info
  └── exclude
  └── refs
logs
  └── HEAD
    └── refs
      └── heads
        ├── main
        └── test
lost-found
  └── commit
    └── 1bd6526eda494b7e3894ece6adde5a79bc0a5891
  └── other
    └── e69de29bb2d1d6434b8b29ae775ad8c2e48c5391
objects
  └── info
    └── commit-graph
  └── packs
    └── pack-bd07af6fae83a78a8de783dd33a1d795de070239.idx
    └── pack-bd07af6fae83a78a8de783dd33a1d795de070239.pack
packed-refs
refs
  └── heads
  └── tags

15 directories, 30 files
```

@稀土掘金技术社区

我们发现去掉了许多没用的文件，并把之前的东西打包了

2.12 完整的git视图



很多ref, branch或者tag, tag可能是附注标签, 指向一个tag, 每个commit都会有一个tree和blob, 每个commit都会有一个仓库

2.13 git clone & pull & fetch

clone

拉取完整的仓库到本地目录, 可以指定分支, 深度

fetch

将远端某些分支最新的代码拉取到本地, 不会执行merge操作

会修改refs/remote内的分支信息, 如果需要和本地代码合并需要手动操作

pull

拉取远端某分支, 并和本地代码进行合并, 操作等于`git fetch + git merge`, 也可以通过`git pull --rebase`完成`git fetch + git rebase`操作

可能存在冲突, 需要解决冲突

2.14 git push

常用命令 `git push origin main`

常见问题

1. 为什么我明明配置了git配置，但是依然没有办法拉取代码

- 免密认证没有配
- instead of配置没有配，配的ssh免密配置，但是使用的还是http协议访问

2. 为什么我fetch了远端分支，但是我看本地当前的分支历史还是没有变化

- fetch会把代码拉取到本地的远端分支，但是并不会合并到当前分支，所以当前分支历史没有变化，要手动merge或者rebase

Git 研发流程

03常见问题

1. 在gerrit平台上使用merge的方式合代码
2. 不了解保护分支，code review，ci等概念，研发流程不规范
3. 代码历史混乱，合并方式不清晰

3.1 不同的工作流

类型	代表平台	特点	合入方式
集中式工作流	Gerrit/ svn	只依托于主干分支进行开发，不存在其他分支	fast-forward
分支管理 workflow	github/ gitlab	可以定义不同特性的开发分支，上线分支，在开发分支完成开发后再通过MR/PR合入主干分支	自定义，fast-forward or three-way merge都可以

3.2 集中式工作流

只依托于main分支进行研发活动

工作方式

1. 获取远端main代码
2. 直接在main上完成修改
3. 提交前拉取最新的代码和本地的代码合并，并解决冲突

4. 提交

3.2.1 集中式工作流- Gerrit

安卓开发用的多

基本原理

1. 依托于 change id概念
2. 提交上去的代码要review

优点

1. 提供强制的代码评审机制，保证代码的质量
2. 提供更丰富的权限功能，可以针对分支权限的管控
3. 保证main历史的整洁性

缺点

1. 开发人员较多会有冲突

3.3 分支管理工作流

分支管理工作流 特点

git flow	分支类型丰富，规范严格
github flow	只有主干分支和开发分支，规则简单
gitlab flow	在主干分支和开发分支上构建环境分支，版本分支，满足不同发布or环境需要

3.3.1 分支管理工作流- gitflow

早期出现的分支管理策略

五种类型分支：

- master
- develop
- feature
- release
- hotfix

优点：

- 代码会很清晰

缺点：

- 流程复杂

3.3.2 github flow

只有一个主干分支，基于pull request往主干分支中提交代码

选择团队合作的方式

1. owner创建好仓库后，其他用户通过fork的方式来创建自己的仓库，并在fork的仓库上进行开发
2. owner创建好仓库后，统一给团队内成员分配权限，直接在同一仓库内进行开发

我们按照第二种方式来创建一个仓库：

- new 一个repository
- 复制ssh的key
- 在本地 `git clone git@github.com:Bruce-XUZILIN/demo.git`
- 查看tree，里面什么都没没有
- 新建一个readme，add再commit再push

新切一个feature分支，修改readme文件，再push

```
(base) xzl@ersandeMacBook-Pro demo % git checkout -b feature
Switched to a new branch 'feature'
(base) xzl@ersandeMacBook-Pro demo % vim readme.md
(base) xzl@ersandeMacBook-Pro demo % git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
      add
(base) xzl@ersandeMacBook-Pro demo % git add .
(base) xzl@ersandeMacBook-Pro demo % git commit -m "update readme"
[feature 70dea13] update readme
 1 file changed, 1 insertion(+), 1 deletion(-)
(base) xzl@ersandeMacBook-Pro demo % git push prigin feature
fatal: 'prigin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
(base) xzl@ersandeMacBook-Pro demo % git push origin feature
Enter passphrase for key '/Users/xzl/.ssh/id_ed25519':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 253 bytes | 253.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature' on GitHub by visiting:
remote:     https://github.com/Bruce-XUZILIN/demo/pull/new/feature
remote:
To github.com:Bruce-XUZILIN/demo.git
 * [new branch]      feature -> feature
```

@稀土掘金技术社区

这里面多了个http的链接，打开就会有pull request

create pull request之后再merge，就会发现main分支上更新了我们修改的内容

之后，我们切回main分支，再运行`git pull origin main`拉取最新的内容

我们再看git log会发现多了一个commit

```

remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 618 bytes | 309.00 KiB/s, done.
From github.com:Bruce-XUZILIN/demo
 * branch           main      -> FETCH_HEAD
   f77542e..402fadb main      -> origin/main
Updating f77542e..402fadb
Fast-forward
 README.md | 2 ++
 1 file changed, 1 insertion(+), 1 deletion(-)
(base) xzl@ersandeMacBook-Pro demo % git log
commit 402fadba28acf3aa7fe4ecafcc609c395ecf3862 (HEAD -> main, origin/main)
Merge: f77542e 70dea13
Author: Zilin Xu <zx112@duke.edu>
Date: Mon May 22 16:57:38 2023 +0800

  Merge pull request #1 from Bruce-XUZILIN/feature

  update README

commit 70dea1337c83987da6c69db7ae06778a2904eaba (origin/feature, feature)
Author: zilin xu <zx112@duke.edu>
Date: Mon May 22 16:53:06 2023 +0800

  update README

commit f77542e2c1e62950236a71c2fb8a29cdf32a118b
Author: zilin xu <zx112@duke.edu>
Date: Mon May 22 16:51:30 2023 +0800

  add README
(base) xzl@ersandeMacBook-Pro demo %

```

可以去**branch**设置一些保护分支的规范

3.3.2 gitlab flow

在gitflow 和 github flow上做出优化，既保持了单一住分支的渐变，又可以适应不同的开发环境

原则：upstream first 上游优先

只有在上游分支才拿的代码才可以进入到下游分支，一般上游分支就是main

3.4 代码合并

fast-forward: `git merge test --ff-only`

three-way (会生成一个merge节点) : `git merge test --no-ff`

3.5 如何选择合适的工作流

小型团队合作: github

- 少量滴哦次
- 提交pull request最少保证cr后再merge
- 主干分支整洁，ff的方式合并，合并前rebase

常见问题

1. 在gerrit平台上使用merge的方式合代码：不推荐
2. 不了解保护分支，code review，ci等概念，研发流程不规范

3. 代码历史混乱，合并方式不清晰