

git课程记录

前言：课程挺好的，原理+实践

1. Git 是什么

- 介绍版本控制的发展历史，为什么会出现Git
- 介绍Git的发展历史

分布式版本控制

1.1 版本控制



- **Git 是什么？**

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- **版本控制是什么？**

一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统
- **为什么需要版本控制？**

更好的关注变更，了解到每个版本的改动是什么，方便对改动的代码进行检查，预防事故发生；也能够随时切换到不同的版本，回滚误删误改的问题代码；

版本控制类型	代表性工具	解决的问题
本地版本控制	RCS	本地代码的版本控制
集中式版本控制	SVN	提供一个远端服务器来维护代码版本，本地不保存代码版本，解决多人协作问题
分布式版本控制	Git	每个仓库都能记录版本历史，解决只有一个服务器保存版本的问题

1.1 版本控制方式

- 最初：文件夹，v1-20220101, v2-20220303

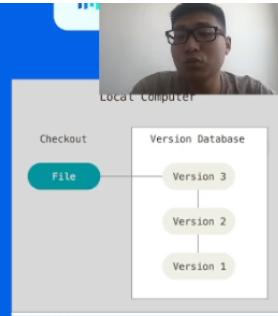
1.1.1 本地版本控制

最初的方式

通过本地复制文件夹，来完成版本控制，一般可以通过不同的文件名来区分版本

解决方案

开发了一些本地的版本控制软件，其中最流行的是 RCS



基本原理

本地保存所有变更的补丁集，可以理解成就是所有的 Diff，通过这些补丁，我们可以计算出每个版本的实际的文件内容

缺点

RCS 这种本地版本控制存在最致命的缺陷就是只能在本地使用，无法进行团队协作，因此使用的场景非常有限，因此衍生出了集中式版本控制

- 集中式版本控制

1.1.2 集中式版本控制

代表性工具: SVN

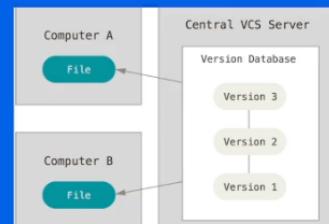


基本原理:

1. 提供一个远端服务来保存文件，所有用户的提交都提交到该服务器中
2. 增量保存每次提交的 Diff，如果提交的增量中和远端现存的文件存在冲突，则需要本地提前解决冲突

优点:

1. 学习简单，更容易操作
2. 支持二进制文件，对大文件支持更友好



缺点:

1. 本地不存储版本管理的概念，所有提交都只能联上服务器后才可以提交
2. 分支上的支持不够好，对于大型项目团队合作比较困难
3. 用户本地不保存所有版本的代码，如果服务端故障容易导致历史版本的丢失。

未经授权不得录制和转载

- 现在：分布式版本控制

1.1.3 分布式版本控制

代表性工具: Git



基本原理:

1. 每个库都存有完整的提交历史，可以直接在本地进行代码提交
2. 每次提交记录的都是完整的文件快照，而不是记录增量
3. 通过 Push 等操作来完成和远端代码的同步



优点:

1. 分布式开发，每个库都是完整的提交历史，支持本地提交，强调个体
2. 分支管理功能强大，方便团队合作，多人协同开发
3. 校验和机制保证完整性，一般只添加数据，很少执行删除操作，不容易导致代码丢失

缺点:

1. 相对 SVN 更复杂，学习成本更高
2. 对于大文件的支持不是特别好（git-lfs 工具可以弥补这个功能）

未经授权不得录制和转载

1.2 Git历史

1.2 Git 发展历史



作者

Linus Torvalds (就是 Linux 这个项目的作者，同时也是 Git 的作者)。

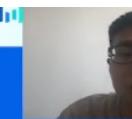
开发原因

怀疑 Linux 团队对 BitKeeper (另一种分布式版本控制系统，专有软件) 进行了逆向工程，BitKeeper 不允许 Linux 团队继续无偿使用。因此决定自己开发一个分布式版本控制系统。

开发时间

大概花了两周时间，就完成了 Git 的代码第一个版本，后续 Linux 项目就开始使用 Git 进行维护。

1.2 Git 发展历史



Github



全球最大的代码托管平台，大部分的开源项目都放在这个平台上。

Gitlab



全球最大的开源代码托管平台，项目的所有代码都是开源的，便于在自己的服务器上完成 Gitlab 的搭建。

Gerrit



由 Google 开发的一个代码托管平台，Android 这个开源项目就托管在 Gerrit 之上。

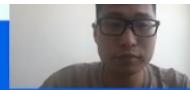
2. Git 基本使用方式

- Git的基本命令介绍，如何使用这些命令，以及命令的原理

02. Git 基本命令



02. 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？
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？

实操环节

```
# 更新git bash
$ git update-git-for-windows
$ mkdir demo
$ cd demo
$ git init
# 在git bash中添加别名，实现类似原生使用windows自带的tree
# 在这个目录下
$ echo "# Set alias for tree command" >> ./etc/bash.bashrc
$ echo "alias tree='wimpy tree.com'" >> ./etc/bash.bashrc
$ source ./etc/bash.bashrc
# 重启git bash, 回到项目目录
$ tree .git
卷 Windows-SSD 的文件夹 PATH 列表
卷序列号为 0000007B DE72:9455
C:\USERS\14224\DESKTOP\STUDY\DEMO\.GIT
├─hooks
├─info
├─objects
| ├─info
| └─pack
└─refs
    ├─heads
    └─tags
$ cat .git/HEAD
ref: refs/heads/master
```

```
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = false
    bare = false
    logallrefupdates = true
    symlinks = false
    ignorecase = true
# 用户名配置
$ git config --global user.name "Baojiazhong"
$ git config --global user.email 1422401429@qq.com
14224@LAPTOP-BHE0I84M MINGW64 ~
$ cat .gitconfig
[credential]
    helper = store
[user]
    email = 1422401429@qq.com
    name = Baojiazhong
[color]
    ui = auto
[filter "lfs"]
    clean = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
    process = git-lfs filter-process
    required = true
# Instead of配置, 没做
$ git config --global url.git@github.com:.insteadOf https://github.com/
# Git 命令别名配置,没做
$ git config --global alias.cin "commit --amend --no-edit"
# Remote配置, http和https两种
# 查看Remote配置
$ git remote -v

# 添加Remote
$ git remote add origin_ssh git@github.com:git/git.git
$ git remote add origin_http https://github.com/git/git.git
# 再次查看Remote
$ git remote -v
origin_http     https://github.com/git/git.git (fetch)
origin_http     https://github.com/git/git.git (push)
origin_ssh      git@github.com:git/git.git (fetch)
origin_ssh      git@github.com:git/git.git (push)
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = false
    bare = false
    logallrefupdates = true
    symlinks = false
    ignorecase = true
[remote "origin_ssh"]
    url = git@github.com:git/git.git
    fetch = +refs/heads/*:refs/remotes/origin_ssh/*
[remote "origin_http"]
    url = https://github.com/git/git.git
    fetch = +refs/heads/*:refs/remotes/origin_http/*
$ git remote -h
usage: git remote [-v | --verbose]
```

```

or: git remote add [-t <branch>] [-m <master>] [-f] [--tags | --no-tags] [--mirror=<fetch|push>] <name> <url>
or: git remote rename [--[no-]progress] <old> <new>
or: git remote remove <name>
or: git remote set-head <name> (-a | --auto | -d | --delete | <branch>)
or: git remote [-v | --verbose] show [-n] <name>
or: git remote prune [-n | --dry-run] <name>
or: git remote [-v | --verbose] update [-p | --prune] [(<group> | <remote>)...]
or: git remote set-branches [--add] <name> <branch>...
or: git remote get-url [--push] [--all] <name>
or: git remote set-url [--push] <name> <newurl> [<oldurl>]
or: git remote set-url --add <name> <newurl>
or: git remote set-url --delete <name> <url>

-v, --verbose           be verbose; must be placed before a subcommand
# 同一个Origin设置不同的Push和Fetch URL
$ git remote add origin git@github.com:git/git.git
$ git remote set-url --add --push origin git@github.com:MY_REPOSITORY/git.git
$ git remote -v
origin  git@github.com:git/git.git (fetch)
origin  git@github.com:MY_REPOSITORY/git.git (push)
origin_http    https://github.com/git/git.git (fetch)
origin_http    https://github.com/git/git.git (push)
origin_ssh     git@github.com:git/git.git (fetch)
origin_ssh     git@github.com:git/git.git (push)
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = false
    bare = false
    logallrefupdates = true
    symlinks = false
    ignorecase = true
[remote "origin_ssh"]
    url = git@github.com:git/git.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[remote "origin_http"]
    url = https://github.com/git/git.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[remote "origin"]
    url = git@github.com:git/git.git
    fetch = +refs/heads/*:refs/remotes/origin/*
    pushurl = git@github.com:MY_REPOSITORY/git.git
# 直接用vim编辑上述文件也可以
# HTTP Remote 不推荐
# 免密配置
# 内存
$ git config --global credential.helper 'cache --timeout=3600'
# 硬盘, 不指定目录的情况默认是 ~/.git-credentials
$ git config --global credential.helper "store --file /path/to/credential-file"
# 将密钥信息存在指定文件中
# SSH Remote 推荐使用
$ ssh-keygen -t ed25519 -C "1422401429@qq.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/14224/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:

```

```
Your identification has been saved in /c/Users/14224/.ssh/id_ed25519
Your public key has been saved in /c/Users/14224/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:ow3It0slfgYEas09Xy3CIYph9ib4KlylVA9WyPVck94 1422401429@qq.com
The key's randomart image is:
+-- [ED25519 256] --+
| + o+=o. o. |
| + Bo*0oo...o |
| . =.*.+.oo+ o |
| o.+oo o o o E |
| .oo = S |
| ... o o . |
| o. = + |
| . . + |
| . . |
+--- [SHA256] ---+
$ touch readme.md
$ git status
On branch master

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)
$ echo hello > readme.md
$ git add .
warning: LF will be replaced by CRLF in readme.md.
The file will have its original line endings in your working directory

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/demo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:   readme.md
# ce + ce目录下的文件名
$ git cat-file -p ce013625030ba8dba906f756967f9e9ca394464a
hello
$ git commit -m "add readme"
[master (root-commit) 50e1a45] add readme
 1 file changed, 1 insertion(+)
 create mode 100644 readme.md
# 多了50, f0
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/demo (master)
$ git cat-file -p 50e1a4540074f0d1241c639c1e00382480129705
tree f071d59456a38f47c1bb65edbc1c11b405aed491
author Baojiazhong <1422401429@qq.com> 1653370649 +0800
committer Baojiazhong <1422401429@qq.com> 1653370649 +0800

add readme

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/demo (master)
```

```
$ git cat-file -p f071d59456a38f47c1bb65edbc1c11b405aed491
100644 blob ce013625030ba8dba906f756967f9e9ca394464a      readme.md
$ git log
commit 50e1a4540074f0d1241c639c1e00382480129705 (HEAD -> master)
Author: Baojiazhong <1422401429@qq.com>
Date:   Tue May 24 13:37:29 2022 +0800

    add readme
$ cat .git/refs/heads/master
50e1a4540074f0d1241c639c1e00382480129705
# 创建一个新的分支
$ git checkout -b test
Switched to a new branch 'test'
$ cat .git/refs/heads/test
50e1a4540074f0d1241c639c1e00382480129705
$ git tag v0.0.1
$ cat .git/refs/tags/v0.0.1
50e1a4540074f0d1241c639c1e00382480129705
# 附注标签
$ git tag -a v0.0.2 -m "add feature 1"
# objects 多了 f3
$ cat .git/refs/tags/v0.0.2
f36f2681c37ac8c4e6f62173c063f5bb165839d1
$ git cat-file -p f36f2681c37ac8c4e6f62173c063f5bb165839d1
object 50e1a4540074f0d1241c639c1e00382480129705
type commit
tag v0.0.2
tagger Baojiazhong <1422401429@qq.com> 1653371753 +0800

add feature 1
# 追溯历史版本
# 获取历史版本代码
$ vim readme.md
$ git add .
warning: LF will be replaced by CRLF in readme.md.
The file will have its original line endings in your working directory

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/demo (test)
$ git commit -m "update readme"
[test b0496a4] update readme
 1 file changed, 1 insertion(+), 1 deletion(-)
# 多了不少object啊。多了3个 (blob, commit, tree)
$ git log
commit b0496a40ae3c004340e708a0b43e573b07f713fb (HEAD -> test)
Author: Baojiazhong <1422401429@qq.com>
Date:   Tue May 24 14:02:16 2022 +0800

    update readme

commit 50e1a4540074f0d1241c639c1e00382480129705 (tag: v0.0.2, tag: v0.0.1,
master)
Author: Baojiazhong <1422401429@qq.com>
Date:   Tue May 24 13:37:29 2022 +0800

    add readme
$ git cat-file -p b0496a40ae3c004340e708a0b43e573b07f713fb
tree 386eb799bbfd0826f510b6587c5afaae2a962279
parent 50e1a4540074f0d1241c639c1e00382480129705
```

```
author Baojiazhong <1422401429@qq.com> 1653372136 +0800
committer Baojiazhong <1422401429@qq.com> 1653372136 +0800

update readme
$ git cat-file -p 386eb799bbfd0826f510b6587c5afaae2a962279
100644 blob 2be7c65ae93b54b988416f280298b0b8b5f20385      readme.md

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/demo (test)
$ git cat-file -p 2be7c65ae93b54b988416f280298b0b8b5f20385
# hello world
$ git commit --amend
[test 48d8662] update readme!!!!
Date: Tue May 24 14:02:16 2022 +0800
1 file changed, 1 insertion(+), 1 deletion(-)
$ git log
commit 48d8662ab123fffad6e96b7f2f5c9b100ddbe106 (HEAD -> test)
Author: Baojiazhong <1422401429@qq.com>
Date: Tue May 24 14:02:16 2022 +0800

update readme!!!!

commit 50e1a4540074f0d1241c639c1e00382480129705 (tag: v0.0.2, tag: v0.0.1,
master)
Author: Baojiazhong <1422401429@qq.com>
Date: Tue May 24 13:37:29 2022 +0800

add readme
$ tree .git
卷 Windows-SSD 的文件夹 PATH 列表
卷序列号为 000000B4 DE72:9455
C:\USERS\14224\DESKTOP\STUDY\DEMO\.GIT
├─hooks
├─info
├─logs
| └─refs
|   └─heads
├─objects
| ├─2b
| ├─38
| ├─48
| ├─50
| ├─b0
| ├─ce
| ├─f0
| ├─f3
| ├─info
| └─pack
└─refs
  ├─heads
  └─tags
# 就是新增了一个commit, 之前的commit悬空了, 没有用了
$ git fsck --lost-found
Checking object directories: 100% (256/256), done.
dangling commit b0496a40ae3c004340e708a0b43e573b07f713fb
# 删除悬空的commit
# Git GC
$ git reflog expire --expire=now --all
$ git gc --prune=now
```

```

Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (7/7), done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
$ tree .git/
卷 Windows-SSD 的文件夹 PATH 列表
卷序列号为 00000012 DE72:9455
C:\USERS\14224\DESKTOP\STUDY\DEMO\.GIT
├─hooks
├─info
├─logs
│ └─refs
│   └─heads
├─lost-found
│ └─commit
└─objects
  ├─info
  └─pack
└─refs
  ├─heads
  └─tags
$ git cat-file -p b0496a40ae3c004340e708a0b43e573b07f713fb
fatal: Not a valid object name b0496a40ae3c004340e708a0b43e573b07f713fb

```

2.1 Git 目录介绍



项目初始化

```

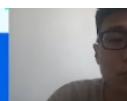
mkdir study
cd study
git init

```

其他参数

--initial-branch 初始化的分支
--bare 创建一个裸仓库(纯 Git 目录, 没有工作目录)
--template 可以通过模版来创建预先构建好的自定义 git 目录

2.1 Git 目录介绍



Git 仓库

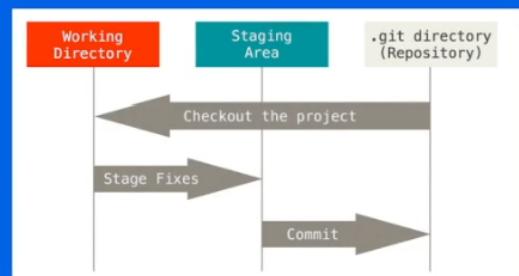
```

study git:(master) tree .git
.git
├── HEAD
├── config
├── hooks
│   └── commit-msg
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

7 directories, 3 files

```

工作区 & 暂存区



2.1.1 Git Config



不同级别的 Git 配置

```
--global
For writing options: write to global ~/.gitconfig file rather than the repository .git/config, write to $XDG_CONFIG_HOME/git/config file if this file exists and the ~/.gitconfig file doesn't.
For reading options: read only from global ~/.gitconfig and from $XDG_CONFIG_HOME/git/config rather than from all available files.
See also the section called "FILES".
--system
For writing options: write to system-wide ${prefix}/etc/gitconfig rather than the repository .git/config.
For reading options: read only from system-wide ${prefix}/etc/gitconfig rather than from all available files.
See also the section called "FILES".
--local
For writing options: write to the repository .git/config file. This is the default behavior.
For reading options: read only from the repository .git/config rather than from all available files.
See also the section called "FILES".
```

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

MINGW64:/c/Users/14224

```
14224@LAPTOP-BHE0I84M MINGW64 ~
$ cat .gitconfig
[credential]
    helper = store
[user]
    email = 1422401429@qq.com
    name = Baojiazhong
[color]
    ui = auto
[filter "lfs"]
    clean = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
    process = git-lfs filter-process
    required = true
```

2.1.2 常见 Git 配置



用户名配置

```
git config --global user.name "liaoxingju"
git config --global user.email liaoxingju@bytedance.com
```

```
[user]
    email = liaoxingju@bytedance.com
    name = liaoxingju
```

Instead of 配置

```
git config --global url.git@github.com:.insteadOf https://github.com/
[url "git@github.com:"]
    insteadOf = https://github.com/
```

Git 命令别名配置

```
git config --global alias.cin "commit --amend --no-edit"
```

```
[alias]
    cin = commit --amend --no-edit
```

将密钥信息存在指定文件中

具体格式: \${scheme}://\${user}:\${password}@github.com

2.2.1 HTTP Remote

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

免密配置

内存: git config --global credential.helper 'cache --timeout=3600'

硬盘: git config --global credential.helper "store --file /path/to/credential-file"
不指定目录的情况下默认是 ~/.git-credentials

将密钥信息存在指定文件中

具体格式: \${scheme}://\${user}:\${password}@github.com

2.2.2 SSH Remote

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



免密配置

SSH 可以通过公私钥的机制，将生成公钥存放在服务端，从而实现免密访问

目前的 Key 的类型四种，分别是 dsa、rsa、ecdsa、ed25519

默认使用的是 rsa，由于一些安全问题，现在已经不推荐使用 dsa 和 rsa 了，优先推荐使用 ed25519

ssh-keygen -t ed25519 -C "your_email@example.com" 密钥默认存在 ~/.ssh/id_ed25519.pub

The screenshot shows two parts of the GitHub account settings interface. On the left, a modal window titled 'Generating a new SSH key' provides instructions for generating an Ed25519 key. It includes steps: 1. Open Terminal (终端). 2. Paste the text below, substituting in your GitHub email address. Below these are command examples for ssh-keygen. A note at the bottom says: 'Note: If you are using a legacy system that doesn't support the Ed25519 algorithm, use: \$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"' On the right, the main account settings page shows the 'SSH keys' section. It lists a single key (labeled 'SSH') with a 'Delete' button. Below it is the 'GPG keys' section, which currently has no keys.

2.3 Git Add

```
study git:(master) touch readme.md
study git:(master) × tree .git
.git
├── HEAD
├── config
├── hooks
│   └── commit-msg
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

7 directories, 3 files
study git:(master) × git status
位于分支 master

尚无提交

未跟踪的文件：
(使用 "git add <文件>..." 以包含要提交的内容)
readme.md

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）
```

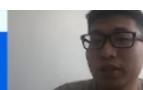
```
study git:(master) × tree .git
.git
├── HEAD
├── config
├── hooks
│   └── commit-msg
├── index
├── objects
│   ├── e6
│   │   └── 7de29bb2d1d6434b8b29ae775ad8c2e48c5391
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

8 directories, 5 files
study git:(master) × git status
位于分支 master

尚无提交

要提交的变更：
(使用 "git rm --cached <文件>..." 以取消暂存)
新文件： readme.md

study git:(master) ×
```



名称	修改日期	类型
50	2022/5/24 13:37	文件夹
ce	2022/5/24 13:33	文件夹
f0	2022/5/24 13:37	文件夹
info	2022/5/24 11:23	文件夹
pack	2022/5/24 11:23	文件夹

2.5 Objects

commit / tree / blob 在 git 里面都统一称为 Object, 除此之外还有个 tag 的 object.

Blob

存储文件的内容

Tree

存储文件的目录信息

Commit

存储提交信息, 一个 Commit 可以对应唯一版本的代码

2.5 Objects

如何把这三个信息串联在一起呢 ?

1. 通过 Commit 寻找到 Tree 信息, 每个 Commit 都会存储对应的 Tree ID.

```
[→ study git:(master) git cat-file -p 373137d6db9018d97279384fb5644e880a9c1b1b
tree 239ec593c6a2192e76c005435f748b2ad28be832
author liaoicingju <liaoicingju@bytedance.com> 1648371702 +0800
committer liaoicingju <liaoicingju@bytedance.com> 1648371702 +0800

add readme
```

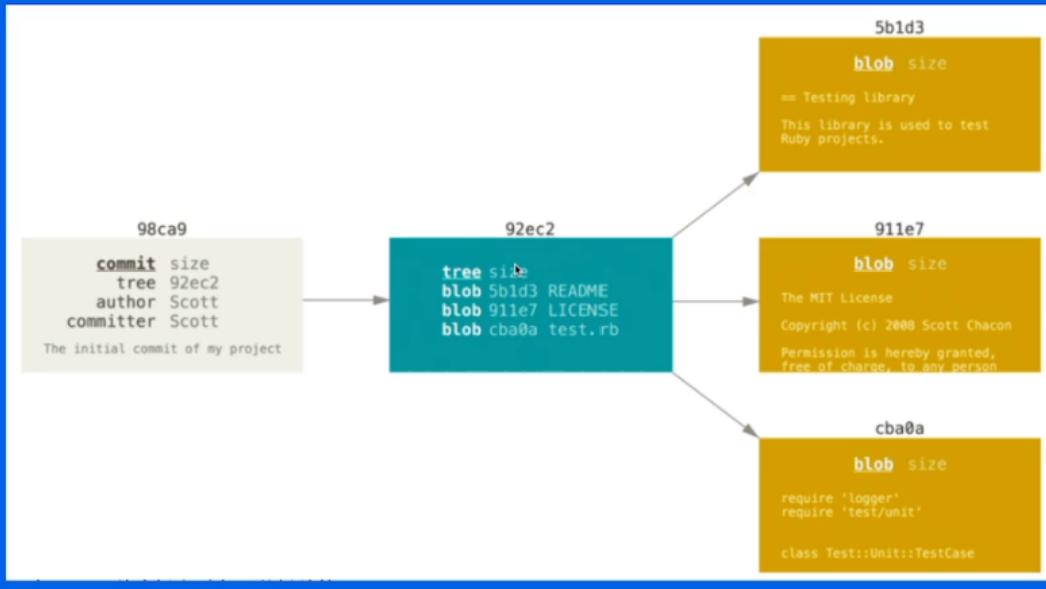
2. 通过 Tree 存储的信息, 获取到对应的目录树信息。

```
[→ study git:(master) git cat-file -p 239ec593c6a2192e76c005435f748b2ad28be832
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    readme.md
```

3. 从 tree 中获得 blob 的 ID, 通过 Blob ID 获取对应的文件内容。

```
[→ study git:(master) git cat-file -p e69de29bb2d1d6434b8b29ae775ad8c2e48c5391
[→ study git:(master)]
```

2.5 Objects



2.6 Refs



Refs 文件存储的内容

```
[→ study git:(test) cat .git/refs/heads/master  
373137d6db9018d97279384fb5644e880a9c1b1b  
[→ study git:(test) cat .git/refs/heads/test  
373137d6db9018d97279384fb5644e880a9c1b1b
```

refs 的内容就是对应的 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

refs/heads 前缀表示的是分支，除此之外还有其他种类的 ref，比如 refs/tags 前缀表示的是标签。

2.6 Refs

Branch

git checkout -b 可以创建一个新分支

分支一般用于开发阶段，是可以不断添加 Commit 进行迭代的

Tag

标签一般表示的是一个稳定版本，指向的 Commit 一般不会变更

通过 git tag 命令生成 tag.

2.7 Annotation Tag

什么是附注标签？

一种特殊的 Tag，可以给 Tag 提供一些额外的信息。

如何创建附注标签？

通过 git tag -a 命令来完成附注标签的创建。

```
study git:(test) cat .git/ref/tags/v0.0.2  
be7fb8574500354cd5887443552e92134c4c2486
```

查看该 tag object 的内容。

```
study git:(test) git cat-file -p be7fb8574500354cd5887443552e92134c4c2486  
object 373137d6db9018d97279384fb5644e880a9c1b1b  
type commit  
tag v0.0.2  
tagger liaoicingju <liaoicingju@bytedance.com> 1648374424 +0800  
my version v0.0.2
```

```
study git:(test) git tag -a v0.0.2 -m  
study git:(test) tree .git  
.git  
├── COMMIT_EDITMSG  
├── HEAD  
├── config  
├── hooks  
│   └── commit-msg  
├── index  
├── logs  
│   └── HEAD  
│       └── refs  
│           └── heads  
│               └── master  
│                   └── test  
└── objects  
    ├── 23  
    │   └── 9ec593c6a2192e76c005435f748b2ad28be832  
    ├── 37  
    │   └── 3137d6db9018d97279384fb5644e880a9c1b1b  
    ├── be  
    │   └── 7fb8574500354cd5887443552e92134c4c2486  
    ├── e6  
    │   └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391  
    ├── info  
    └── pack  
        └── refs  
            ├── heads  
            │   └── master  
            │       └── test  
            └── tags  
                └── v0.0.1  
                    └── v0.0.2  
14 directories, 16 files
```

2.8 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。

2.9 修改历史版本

1. commit --amend

通过这个命令可以修改最近的一次 commit 信息，修改之后 commit id 会变

2. rebase

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

1. 合并 commit
2. 修改具体的 commit message
3. 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 Objects

新增的 Object

修改 Commit 后我们可以发现 git object 又出现了变化

新增 commit object 7f

但是之前的 commit object 63 并没有被删除

悬空的 Object

顾名思义就是没有 ref 指向的 object

```
study git:(test) git fsck --lost-found
正在检查对象目录：100% (256/256), 完成。
悬空 commit 63ae840dfe33df2975f9653903cb533f2912bc1c
```

```
study git:(test) tree .git
.git
├── COMMIT_EDITMSG
├── HEAD
├── config
├── hooks
│   └── commit-msg
├── index
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
│               └── test
└── lost-found
    └── commit
        └── 63ae840dfe33df2975f9653903cb533f2912bc1c
            ├── 23
            │   └── 9ec593c6a2192e76c005435f748b2ad28be832
            ├── 37
            │   └── 3137d6db9018d97279384fb5644e880a9c1b1b
            ├── 3a
            │   └── 3aff7fa9639da674465c43fac566c1291f952b
            ├── 55
            │   └── 7db03de997c86a4a028e1ebd3a1ceb225be238
            ├── 63
            │   └── ae840dfe33df2975f9653903cb533f2912bc1c
            ├── 7f
            │   └── b4383e68df781422df87876c30af884aeae86e
            ├── b4
            ├── b6
            │   └── 7fb8574500354cd5887443552e92134c4c2486
            ├── e6
            │   └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
            └── info
                └── pack
                    └── refs
                        ├── heads
                        │   ├── master
                        │   └── test
                        └── tags
                            ├── v0.0.1
                            └── v0.0.2
```

2.11 Git GC



GC

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

Reflog

reflog 是用于记录操作日志，防止误操作后数据丢失，通过 reflog 来找到丢失的数据，手动将日志设置为过期。

指定时间

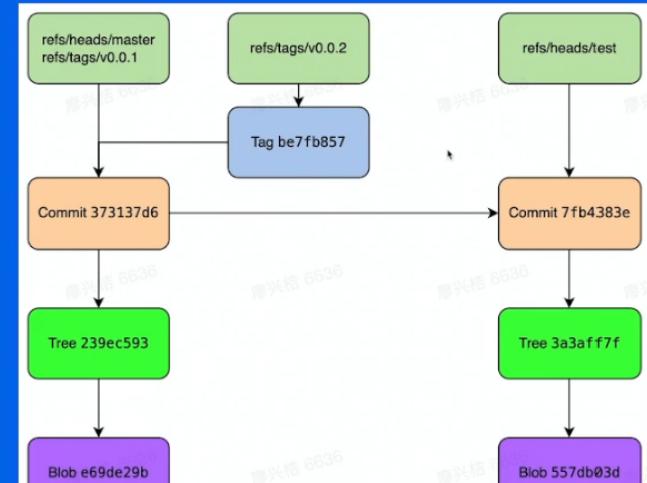
git gc prune=now 指定的是修剪多久之前的对象，
默认是两周前

```
study git:(test) git reflog expire --expire=now --prune=now
对象计数中: 7, 完成.
使用 12 个线程进行压缩
压缩对象中: 100% (3/3), 完成.
写入对象中: 100% (3/3), 完成.
总共 7 (差异 0) , 复用 7 (差异 0) , 包复用 0
study git:(test)
study git:(test)
study git:(test) tree .git
.git
├── COMMIT_EDITMSG
├── HEAD
├── config
├── hooks
│   └── commit-msg
├── index
├── info
│   └── refs
└── logs
    └── HEAD
        └── refs
            └── heads
                └── master
                    └── test
└── lost-found
    └── commit
        └── 63ae840dfe33df2975f9653903cb533f2912bc1c
├── objects
│   ├── info
│   │   └── commit-graph
│   └── pack
│       ├── pack-af720db47eedb7c6cc85f962be3b1fc93eb7ee.idx
│       └── pack-af720db47eedb7c6cc85f962be3b1fc93eb7ee.pack
└── packed-refs
└── refs
    └── heads
        └── tags
13 directories, 15 files
study git:(test) git cat-file -p 63ae840dfe33df2975f9653903cb533f2912bc1c
fatal: Not a valid object name 63ae840dfe33df2975f9653903cb533f2912bc1c
```

2.12 完整的 Git 视图

```
study git:(test) tree .git
.git
├── COMMIT_EDITMSG
├── HEAD
├── config
├── hooks
│   └── commit-msg
├── index
└── logs
    └── HEAD
        └── refs
            └── heads
                └── master
                    └── test
└── objects
    ├── 23
    │   └── 9ec593cde2192e76c005435f748b2ad28be832
    ├── 37
    │   └── 3137d6db9818d97279384fb5644e880a9c1b1b
    ├── 3a
    │   └── 3aff7fa9639d674465c43fac565c1291f952b
    ├── 55
    │   └── 7db63de997c86a4a028e1ebd3a1ccb225be238
    ├── 63
    │   └── aa840dfe33df2975f9653903cb533f2912bc1c
    ├── be
    │   └── 7fb8574500354cd5887443552e92134c4c2486
    ├── e6
    │   └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
    └── info
        └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
        └── v0.0.1
        └── v0.0.2
17 directories, 19 files
```

未经授权不得录制和转载



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。

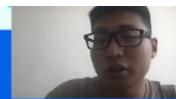
Pull

拉取远端某分支, 并和本地代码进行合并, 操作等同于 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

Push 是将本地代码同步至远端的方式。



常用命令

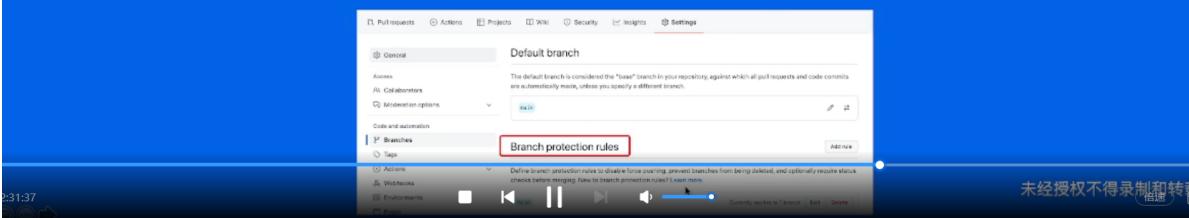
一般使用 git push origin master 命令即可完成

冲突问题

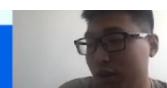
- 如果本地的 commit 记录和远端的 commit 历史不一致, 则会产生冲突, 比如 git commit --amend 或 git rebase 都有可能导致这个问题。
- 如果该分支就自己一个人使用, 或者团队内确认过可以修改历史则可以通过 git push origin master -f 来完成强制推送, 一般不推荐主干分支进行该操作, 正常都应该解决冲突后再进行推送。

推送规则限制

可以通过保护分支, 来配置一些保护规则, 防止误操作, 或者一些不合规的操作出现, 导致代码丢失。



02. 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。

3. Git 研发流程

- 依托代码管理平台 Gitlab/Github/Gerrit介绍我们如何进行代码的开发及团队合作

03. 常见问题

1. 在 Gerrit 平台上使用 Merge 的方式合入代码
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，代码合并方式不清晰

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
分支管理工作流	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

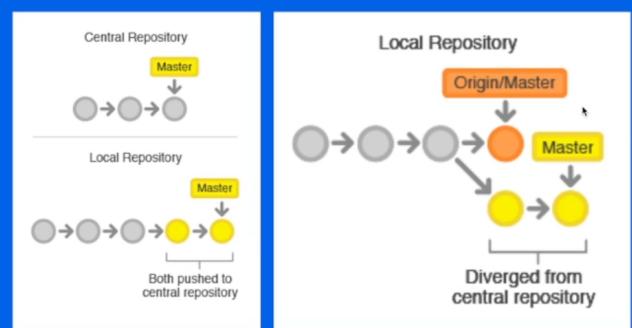


什么是集中式工作流？

只依托于 master 分支进行研发活动

工作方式

1. 获取远端 master 代码
2. 直接在 master 分支完成修改
3. 提交前拉取最新的 master 代码和本地代码进行合并（使用 rebase），如果有冲突需要解决冲突
4. 提交本地代码到 master



3.2.1 集中式工作流-Gerrit



Gerrit 是由 Google 开发的一款代码托管平台，主要的特点就是能够很好的进行代码评审。在 aosp (android open source project) 中使用的很广，Gerrit 的开发流程就是一种集中式工作流。

基本原理

1. 依托于 Change ID 概念，每个提交生成一个单独的代码评审。
2. 提交上去的代码不会存储在真正的 refs/heads/ 下的分支中，而是存在一个 refs/for/ 的引用下。
3. 通过 refs/meta/config 下的文件存储代码的配置，包括权限，评审等配置，每个 Change 都必须要完成 Review 后才能合入。

3.2.1 集中式工作流-Gerrit



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，可以针对分支做细粒度的权限管控
3. 保证 master 的历史整洁性
4. Aosp 多仓的场景支持更好

缺点

1. 开发人员较多的情况下，更容易出现冲突
2. 对于多分支的支持较差，想要区分多个版本的线上代码时，更容易出现问题
3. 一般只有管理员才能创建仓库，比较难以在项目之间形成代码复用，比如类似的 fork 操作就不支持。

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之上构建环境分支，版本分支，满足不同发布 or 环境的需要

3.3.1 分支管理工作流-GitFlow

Git Flow 时比较早期出现的分支管理策略。

- 包含五种类型的分支

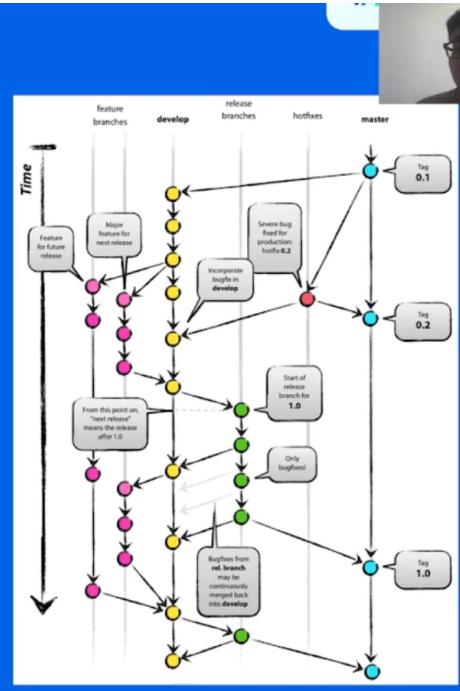
Master: 主干分支
Develop: 开发分支
Feature: 特性分支
Release: 发布分支
Hotfix: 热修复分支

- 优点

如果能按照定义的标准严格执行，
代码会很清晰，并且很难出现混乱。

- 缺点

流程过于复杂，上线的节奏会比较慢。
由于太复杂，研发容易不按照标准执行，
从而导致代码出现混乱。

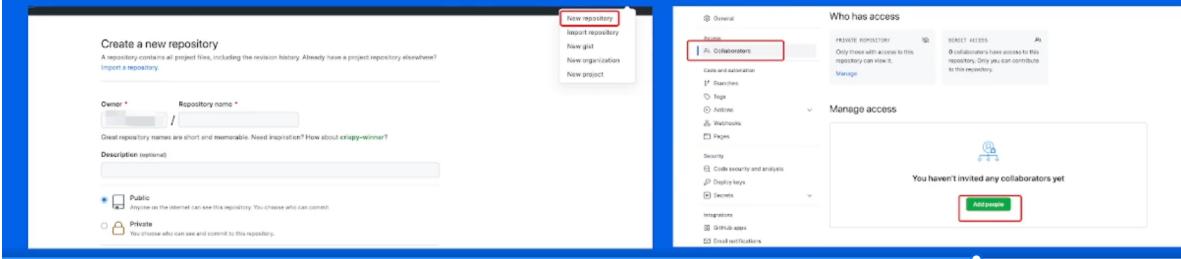


3.3.2 分支管理工作流-Github Flow

Github 的工作流，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

- 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
- owner 创建好仓库后，统一给团队内成员分配权限，直接在同一个仓库内进行开发



```
# 克隆仓库
$ git clone git@github.com:Baojiazhong/demo.git
Cloning into 'demo'...
warning: You appear to have cloned an empty repository.

# push
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ vim readme.md

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git add .
warning: LF will be replaced by CRLF in readme.md.
The file will have its original line endings in your working directory

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git commit -m "add readme"
[main (root-commit) a52c521] add readme
 1 file changed, 1 insertion(+)
 create mode 100644 readme.md
```

```
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 224 bytes | 224.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:Baojiazhong/demo.git
 * [new branch]      main -> main
# 创建新分支
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git checkout -b feature
Switched to a new branch 'feature'

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (feature)
$ vim readme.md

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (feature)
$ git add .
warning: LF will be replaced by CRLF in readme.md.
The file will have its original line endings in your working directory

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (feature)
$ git commit -m "update readme"
[feature 9423187] update readme
 1 file changed, 1 insertion(+), 1 deletion(-)

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (feature)
$ git push origin feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 259 bytes | 259.00 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/Baojiazhong/demo/pull/new/feature
remote:
To github.com:Baojiazhong/demo.git
 * [new branch]      feature -> feature
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (feature)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git pull origin main
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 623 bytes | 124.00 KiB/s, done.
From github.com:Baojiazhong/demo
 * branch            main      -> FETCH_HEAD
   a52c521..2b4b0fd  main      -> origin/main
Updating a52c521..2b4b0fd
Fast-forward
 readme.md | 2 ++
 1 file changed, 1 insertion(+), 1 deletion(-)

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
```

```
$ git log
commit 2b4b0fd3308d79f9b9eaa14f56134ac5ae534eba (HEAD -> main, origin/main)
Merge: a52c521 9423187
Author: Baojiazhong <50916042+Baojiazhong@users.noreply.github.com>
Date:   Tue May 24 14:52:34 2022 +0800

Merge pull request #1 from Baojiazhong/feature

update readme

commit 94231871408c38228c09e2d02397166da3435787 (origin/feature, feature)
Author: Baojiazhong <1422401429@qq.com>
Date:   Tue May 24 14:47:18 2022 +0800

update readme

commit a52c5213032a568648a6368067c3ff60009573e3
Author: Baojiazhong <1422401429@qq.com>
Date:   Tue May 24 14:43:35 2022 +0800

add readme
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ vim readme.md

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git add .

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git commit -m "test rules"
[main f48fd27] test rules
 1 file changed, 1 insertion(+)

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 274 bytes | 274.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote: error: GH006: Protected branch update failed for refs/heads/main.
remote: error: Changes must be made through a pull request.
To github.com:Baojiazhong/demo.git
 ! [remote rejected] main -> main (protected branch hook declined)
error: failed to push some refs to 'github.com:Baojiazhong/demo.git'
# 代码合并
# 1 fast-forward
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git checkout -b test
Switched to a new branch 'test'

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
$ ls
readme.md

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
$ vim readme.md

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
$ git add .
```

```
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
$ git commit -m "test"
[test 6429ec1] test
 1 file changed, 1 insertion(+)

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git merge test --ff-only
Updating f48fd27..6429ec1
Fast-forward
 readme.md | 1 +
 1 file changed, 1 insertion(+)

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git log
commit 6429ec1597ccbd643e4057aea4bbbbdb6fca59c5 (HEAD -> main, test)
Author: Baojiazhong <1422401429@qq.com>
Date:   Tue May 24 15:05:24 2022 +0800

test

commit f48fd2713979dc8efd5f4ab6229f60404de844a6
Author: Baojiazhong <1422401429@qq.com>
Date:   Tue May 24 14:58:58 2022 +0800

test rules

commit 2b4b0fd3308d79f9b9eaa14f56134ac5ae534eba (origin/main)
Merge: a52c521 9423187
Author: Baojiazhong <50916042+Baojiazhong@users.noreply.github.com>
Date:   Tue May 24 14:52:34 2022 +0800

Merge pull request #1 from Baojiazhong/feature

update readme

commit 94231871408c38228c09e2d02397166da3435787 (origin/feature, feature)
Author: Baojiazhong <1422401429@qq.com>
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
$ git checkout test
Switched to branch 'test'

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
$ vim readme.md

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
$ git add .

14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
$ git commit -m "test third party"
[test 799f8f5] test third party
 1 file changed, 1 insertion(+)
```

```
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (test)
```

```
$ git checkout main
```

```
Switched to branch 'main'
```

```
Your branch is ahead of 'origin/main' by 2 commits.
```

```
(use "git push" to publish your local commits)
```

```
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
```

```
$ git merge test --no-ff
```

```
Merge made by the 'ort' strategy.
```

```
readme.md | 1 +
```

```
1 file changed, 1 insertion(+)
```

```
14224@LAPTOP-BHE0I84M MINGW64 ~/Desktop/study/study/demo (main)
```

```
$ git log
```

```
commit 15a7085eb07ccab2704904edaca90093606ccb30 (HEAD -> main)
```

```
Merge: 6429ec1 799f8f5
```

```
Author: Baojiazhong <1422401429@qq.com>
```

```
Date: Tue May 24 15:12:34 2022 +0800
```

```
Merge branch 'test'
```

```
commit 799f8f5446bfd06e1f95aaf0c4795a9cbca3f6ff (test)
```

```
Author: Baojiazhong <1422401429@qq.com>
```

```
Date: Tue May 24 15:10:58 2022 +0800
```

```
test third party
```

```
commit 6429ec1597ccbd643e4057aea4bbbbdb6fca59c5
```

```
Author: Baojiazhong <1422401429@qq.com>
```

```
Date: Tue May 24 15:05:24 2022 +0800
```

```
test
```

```
commit f48fd2713979dc8efd5f4ab6229f60404de844a6
```

```
Author: Baojiazhong <1422401429@qq.com>
```

```
Date: Tue May 24 14:58:58 2022 +0800
```

3.3.2 分支管理工作流–Github Flow

可以通过进行一些保护分支设置，来限制合入的策略，以及限制直接的 push 操作。

The screenshot shows the 'Branch protection rules' configuration page for a GitHub repository. It includes sections for 'Branch name pattern' (empty), 'Protect matching branches' (with several checkboxes for pull requests, status checks, conversations, signed commits, linear history, and administrators), and 'Rules applied to everyone including administrators' (allowing force pushes and deletions).

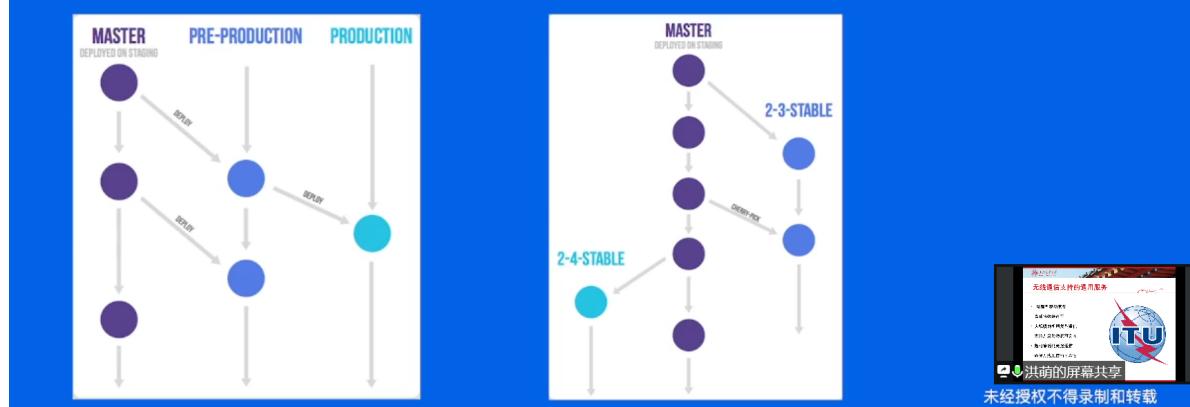
3.3.2 分支管理工作流–Gitlab Flow



Gitlab 推荐的工作流是在 GitFlow 和 Github Flow 上做出优化，既保持了单一主分支的简便，又可以适应不同的开发环境。

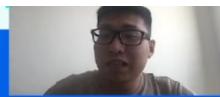
原则: upstream first 上游优先

只有在上游分支采纳的代码才可以进入到下游分支，一般上游分支就是 master.



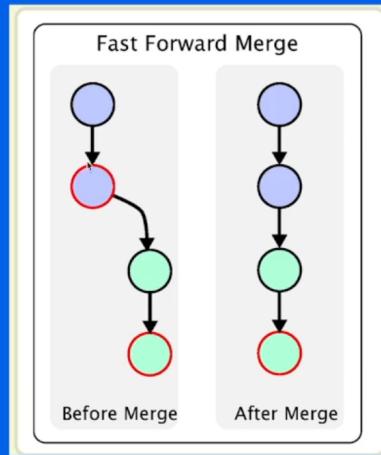
未经授权不得录制和转载

3.4 代码合并



Fast-Forward

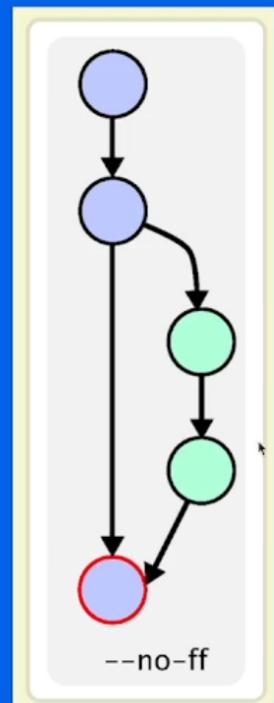
不会产生一个 merge 节点，合并后保持一个线性历史，如果 target 分支有了更新，则需要通过 rebase 操作更新 source branch 后才可以合入。



3.4 代码合并

Three-Way Merge

三方合并，会产生一个新的 merge 节点



3.5 如何选择合适的工作流



选择原则

没有最好的，只有最合适的

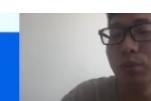
针对小型团队合作，推荐使用 Github 工作流即可

1. 尽量保证少量多次，最好不要一次性提交上千行代码
2. 提交 Pull Request 后最少需要保证有 CR 后再合入
3. 主干分支尽量保持整洁，使用 fast-forward 合入方式，合入前进行 rebase

b

大型团队合作，根据自己的需要指定不同的工作流，不需要局限在某种流程中。

03. 常见问题



1. 在 Gerrit 平台上使用 Merge 的方式合入代码。

Gerrit 是集中式工作流，不推荐使用 Merge 方式合入代码，应该是在主干分支开发后，直接 Push。

2. 不了解保护分支，Code Review，CI 等概念，研发流程不规范。

保护分支：防止用户直接向主干分支提交代码，必须通过 PR 来进行合入。

Code Review，CI：都是在合入前的检查策略，Code Review 是人工进行检查，CI 则是通过一些定制化的脚本来进行一些校验。

3. 代码历史混乱，代码合并方式不清晰。

不理解 Fast Forward 和 Three Way Merge 的区别，本地代码更新频繁的使用 Three Way 的方式，导致生成过多的 Merge 节点，使提交历史变得复杂不清晰。

课后作业



1. 熟悉一个开源项目，学习开源代码，整理成学习笔记，提交到 Github 上。

2. 尝试向开源项目提一个 Pull Request。

