



GitCode使用FAQ（OH版）

GitCode 产品团队



迁移五步走

鸿蒙开发者快速迁移 GitCode 指南



扫码查看更多

#1 登录注册

鸿蒙开发者 Guide <GitCode>

欢迎加入 GitCode 平台！本指南将帮助您快速熟悉 GitCode 的操作，让您高效开展开发协作。

立即加入

#2 gitee授权



GitCode 客户端

此第三方应用请求获得以下权限：

- ☒ 访问你的个人信息、最新动态等
- ☒ 查看你的个人邮箱信息

同意授权

拒绝

你可以随时在 帐号设置 > 第三方应用 中取消你的授权

#3 创建GitCode账号

设置用户名

GitCode 与华为云共同为用户提供代码托管服务，在使用代码托管相关服务时，将同步授权并开通华为云服务，并与重庆开源共创科技有限公司（GitCode的运营主体）关联及共享。

设置用户名

stealing-bear-plan

无需修改用户名

验证手机号

+86 请填写验证手机号

验证码

请填写验证码

获取验证码

- ☐ 我已阅读并同意《华为云用户协议》《华为云隐私政策声明》
- ☐ 我已阅读并同意《GitCode 用户协议》《GitCode 隐私政策》

创建账号并继续

取消

#4 签署协议

亲爱的鸿蒙开发者，欢迎您来到 OpenHarmony 大家庭，让我们一起用代码点亮世界！

欢迎回家！

您正式成为 OpenHarmony 社区的一员啦！从这一刻起，您不仅是 GitCode 社区的伙伴，更是 OpenHarmony 家庭中不可或缺的一部分。我们衷心感谢您为 OpenHarmony 社区的发展注入了新的活力，也感谢您对开源事业的支持与坚持。

在这个温暖的大家庭中，您将志同道合的开发者们，共同探索 OpenHarmony 系统的新篇章。无论是分享灵感、解决问题，还是探索未知，我们都将与您并肩前行。

☒ 下次不再提示

签署授权协议



使用该模板

添加标题*

ArkUI 用户数据迁移授权协议

无需修改标题

甲方

ArkUI 项目组

乙方*

请填写真实姓名

请填写你的姓名...

GitCode 账号*

前往gitee获取

请填写你的 GitCode 账号...

GitCode账号位置：

个人主页头像下方

#5 绑定电子邮件

⚠ 电子邮件是Commit、签署CLA的必要前提

迁移 GitLab

+ 新建



GitCode

@gitcode.com

开源探索者 Y

工作台

我的组织

我的项目

我的 Star

个人设置

操作路径

- 1、右上角个人头像
- 2、个人设置
- 3、电子邮件

个人设置

GitCode

@gitcode.com

用户资料

账号设置

电子邮件

偏好设置

电子邮件

请输入

请输入

添加

一、GitCode OpenHarmony 开发协作场景

我们为鸿蒙开发者在 **GitCode** 平台上的常见开发协作场景整理了一份使用指南，内容分为以下几部分：

1. 基础设置：GitCode 账号
2. 基础功能：个人工作台
3. 开发协作：Repo 开发工具下载
4. 开发协作：Fork + PR 协作模式
5. 开发协作：Issue
6. 开发协作：安全 issue 补丁修复
7. 管理配置：组织、项目管理员的配置
8. 寻求帮助：在线客服、内部交流群、帮助文档、热线电话4006868951

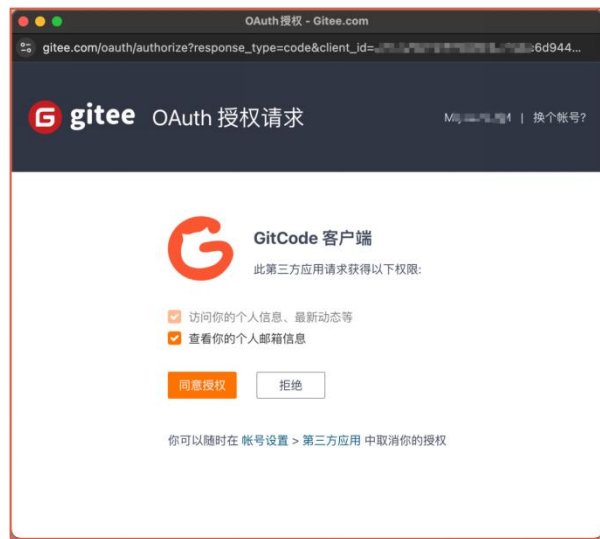
1.1、基础设置：GitCode 账号注册

首次访问 GitCode 时，你需要准备一个 GitCode 账号。对于曾在 Gitee 上参与鸿蒙开发的用户，我们**强烈建议使用 Gitee 第三方账号登录 GitCode**。

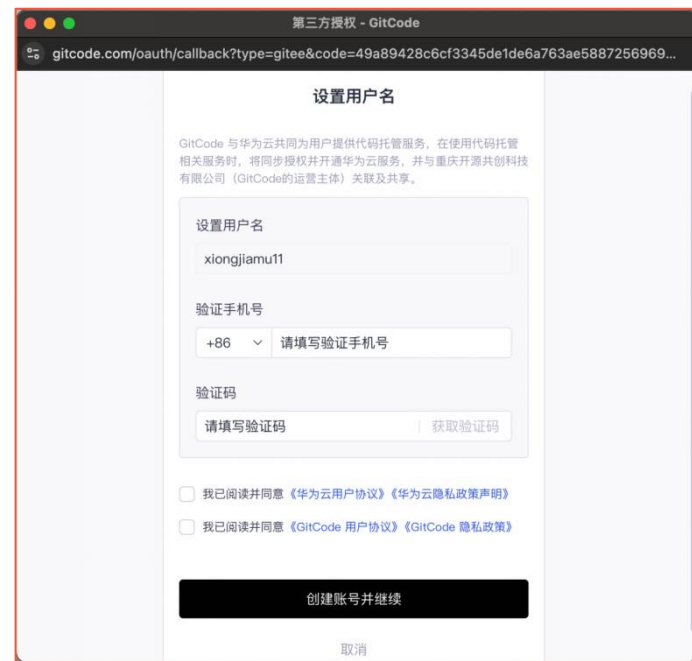
注意：GitCode 与 Gitee 是两个独立的平台，在你授权数据迁移之后，我们会尽可能的保留你之前的贡献数据（issue、pr 及评论内容）。



① 登录注册界面选择 Gitee 登录



② 登录 Gitee 并点击【同意授权】



③ 验证手机号并创建 GitCode注册账号

注意：你的用户名默认会使用 Gitee 的用户名，若你的用户命名被占用，此处会随机给你生成一个用户名。

1.1、基础设置：配置账号密码与绑定的邮箱（强烈建议）

在完成账号注册后，强烈建议你配置账号的密码和邮箱，配置完成后你可以使用密码登录的方式登录 **GitCode**。

- **配置密码（建议操作）**：首次登录后，点击右上角的头像--个人设置--账号设置。此处建议你配置一个复杂密码以保证账号安全，同时，请你牢记并储存你的密码。



GitCode

「开搜」：一搜即达，开发者的AI搜索

Coisini @StevenStark

账号管理

- 用户资料
- 账号设置**
- 电子邮件
- 偏好设置

账号设置

手机号

当前账号认证并绑定的手机号。

+86 185****3691 更换

密码设置

建议定期更换密码，确保安全且易记，避免泄露风险。

修改密码

- **绑定邮箱（必备操作）**：首次登录后，点击右上角的头像--个人设置--电子邮件。首次绑定电子邮件可能需要通过手机验证码验证你的身份，以确保账户安全。配置完成个人邮箱之后，你可以通过个人邮箱进行登录、Commit、签署CLA等操作。强烈建议你配置个人邮箱。



Coisini @StevenStark

账号管理

- 用户资料
- 账号设置
- 电子邮件**

电子邮件

电子邮箱用于控制与你账户关联的电子邮件，所有可用的电子邮件地址都可用于标识你的提交。

请输入邮箱 0/64

请输入 获取验证码 获取验证码

添加邮箱

1.1、基础设置：GitCode 账号登录

完成上述操作之后，未来你可以使用多种方式登录 GitCode，在此我们**建议你在登录界面选择用 Gitee 账号登录 GitCode**。当然，你也可以选择使用其它的方式来登录 GitCode 平台，其中：

- 手机验证码登录：通过已经验证的手机号获取登录用的短信验证码；。
- 账号密码登录：通过账号和配置的密码登录。*注意：你需要先设置账号的登录密码（密码仅用于登录，不能用于HTTP Clone操作）。*
- 邮箱登录：使用邮箱账号和密码登录。*注意：需在设置登录密码的基础上完成邮箱验证*



登录界面选择 Gitee 登录
建议使用



登录界面选择手机号登录

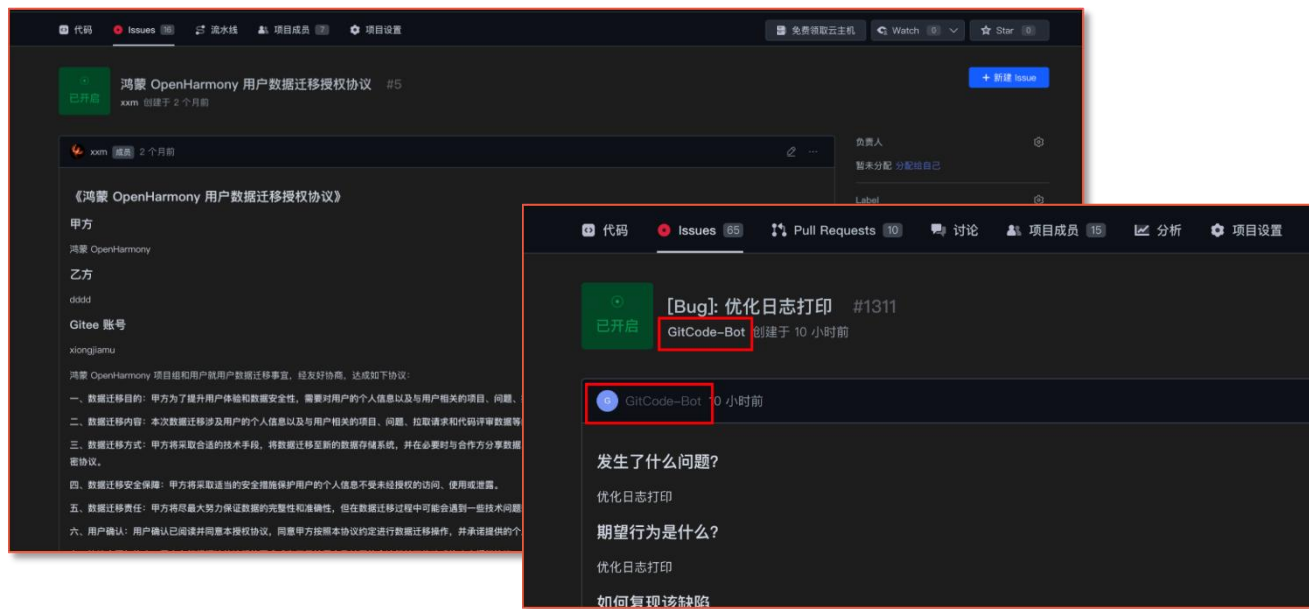


个人设置 - 账号设置/邮箱设置中设置密码和邮箱

1.1、鸿蒙数据迁移授权

为确保用户数据的连续性、操作一致性，并提升鸿蒙项目整体管理效率及维护开源社区的完整性，鸿蒙开发者需在 **GitCode** 上完成鸿蒙数据迁移授权：

- 授权完成：迁移的数据将可在 **GitCode** 平台上公开展示（包括 issue、pr 及评论内容）
- 未授权：未完成授权的数据将仅对项目成员可见，用户名统一显示为 **GitCode-Bot**



授权签署项目地址：<https://gitcode.com/openharmony/oh-agreements/issues>

1.1、GitCode 账号 FAQ

如果你在 OpenHarmony 项目中有贡献（如 issue、PR、评论），迁移过程中我们已为你预创建了一个未激活的 Gitee 第三方账号。只需通过 Gitee 三方登录并完成手机认证，在完成授权迁移的确认后，即可保留你的贡献数据。

Q：可以使用 Gitee 的账号密码登录吗？

A：GitCode 和 Gitee 是两个不同且独立的平台，账号信息不通用，你可以选择通过 Gitee 的三方账号授权登录 GitCode。但是你无法通过 Gitee 的账号密码登录 GitCode。

Q：手机号验证时提示手机号已绑定其他账号，怎么办？

A：请联系客服处理。确认手机号确实属于本人后，我们建议你注销之前的账号，然后使用 Gitee 账号重新授权并完成认证，以便保留之前的贡献数据。

Q：手机验证时提示创建的用户名是类似于“gitee11”，怎么办？

A：这表示你的 Gitee 用户名在 GitCode 上已被占用。如果你是该同名账号的所有者，可以先自行注销该同名账号，然后联系客服处理。在重新同步 Gitee 数据后，你的贡献数据将被恢复。

客服联系方式：1. 在线客服（推荐） 2. 客服邮箱 (kefu@gitcode.com) 3. 客服热线：4006868951

2.1、基础设置：个人设置（1）

在开始协作开发前，我们建议你进行以下个人账号的设置，以确保顺畅的开发体验并提升社区互动效果：

✓ 设置一：个人昵称

选择一个清晰、易识别的昵称，使社区用户更容易记住你并识别你的贡献。一个好的昵称可以帮助你社区中建立良好的个人品牌

- ① 点击右上角头像位置
- ② 下拉菜单中点击个人设置
- ③ 设置你的个人昵称
- ④ 补充你的个人简介（可选）
- ⑤ 启用个人 README（可选）
- ⑥ 点击保存

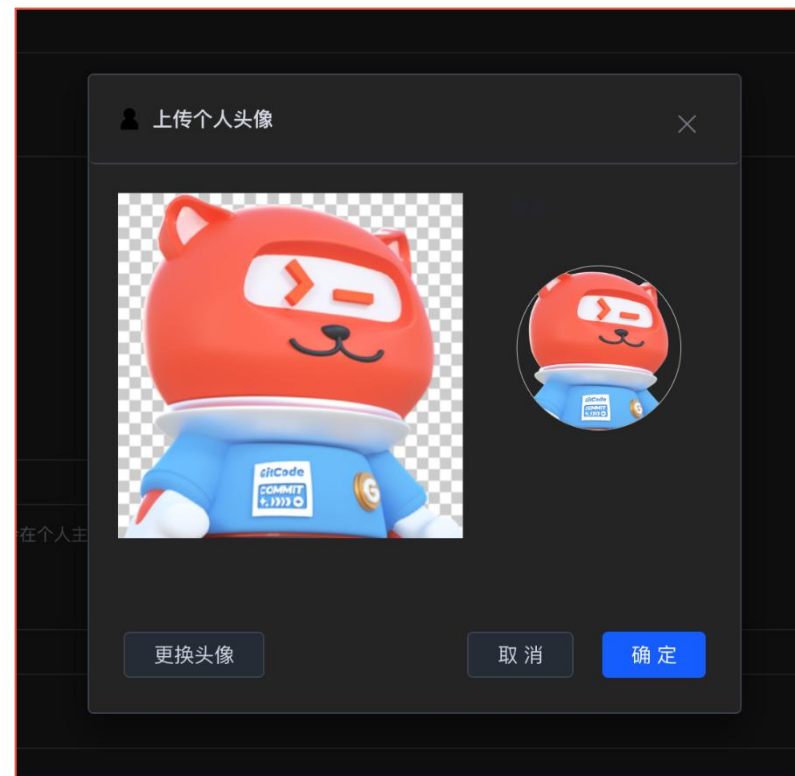
一些推荐的 README 个人模版：https://gitcode.com/gh_mirrors/aw/awesome-github-profile-readme

1.1、基础设置：个人设置（2）

✓ 设置二：个人头像

上传一个有趣或专业的头像，让你的账号更具辨识度。头像可以帮助其他开发者快速识别你的身份，增强交流的亲和力

- ① 点击右上角头像位置
- ② 下拉菜单中点击个人设置
- ③ 点击上传头像
- ④ 从你的电脑设备中选择你的头像图片
- ⑤ 用裁剪工具进行调整（可选）
- ⑥ 点击确定，完成个人头像更新



1.1、基础设置：个人设置（3）

✓ 设置三：认证常用邮箱

认证你常用的邮箱，**特别是用于 commit 提交的邮箱**。确保提交记录中显示正确的贡献者信息，这对于在 git 仓库中清晰记录你的贡献尤为重要

- ① 点击右上角头像位置
- ② 下拉菜单中点击个人设置
- ③ 点击左侧【账号管理-电子邮件】选项
- ④ 输入你的常用邮箱
- ⑤ 点击获取验证码
- ⑥ 验证手机号（安全验证）
- ⑦ 输入邮箱验证码
- ⑧ 完成邮箱认证

说明：注册账号时，系统默认生成的 xxx@noreply 邮箱，将在你绑定常用邮箱后从邮箱列表中自动移除

1.1、基础设置：个人设置（4）

✓ 设置四：创建个人访问令牌（按需创建）

生成个人访问令牌，用于以 **HTTPS** 方式 **clone** 和 **push** 代码。**GitCode** 在 **git** 操作中 仅支持访问令牌认证（不支持密码验证），确保只有经过授权的操作可以访问你的项目

- ① 点击右上角头像位置
- ② 下拉菜单中点击个人设置
- ③ 点击左侧【安全设置-访问令牌】选项
- ④ 点击右上角新建访问令牌
- ⑤ 输入个人令牌名称
- ⑥ 设置令牌有效期、令牌的授权范围（可选）
- ⑦ 点击【新建访问令牌】

补充阅读：[GitHub 关于 clone 时移除密码验证的说明](#)

注意：个人访问令牌在成功创建后仅可见一次，请妥善保存你的个人访问令牌

1.1、基础设置：个人设置（5）

✓ 设置五：添加 SSH 公钥（建议）

在个人设置中添加常用设备的 SSH 公钥，以便通过 SSH 方式 clone 和 push 代码。使用 SSH 公钥不仅更加安全，还能避免频繁输入密码，提升开发效率

- ① 点击右上角头像位置
- ② 下拉菜单中点击个人设置
- ③ 点击左侧【安全设置-SSH公钥】选项
- ④ 点击右上角新建 SSH 公钥按钮
- ⑤ 输入你本机设备的 SSH 公钥（eg: id_rsa.pub 文件）
- ⑥ 点击【新建】，完成 SSH 公钥的添加

说明：添加完 SSH 公钥后，你可以在终端通过 `ssh -t git@gitcode.com` 来查看公钥是否生效

1.1、基础设置：个人设置（6）

✓ 设置六：启用多因素认证 (MFA)（建议）

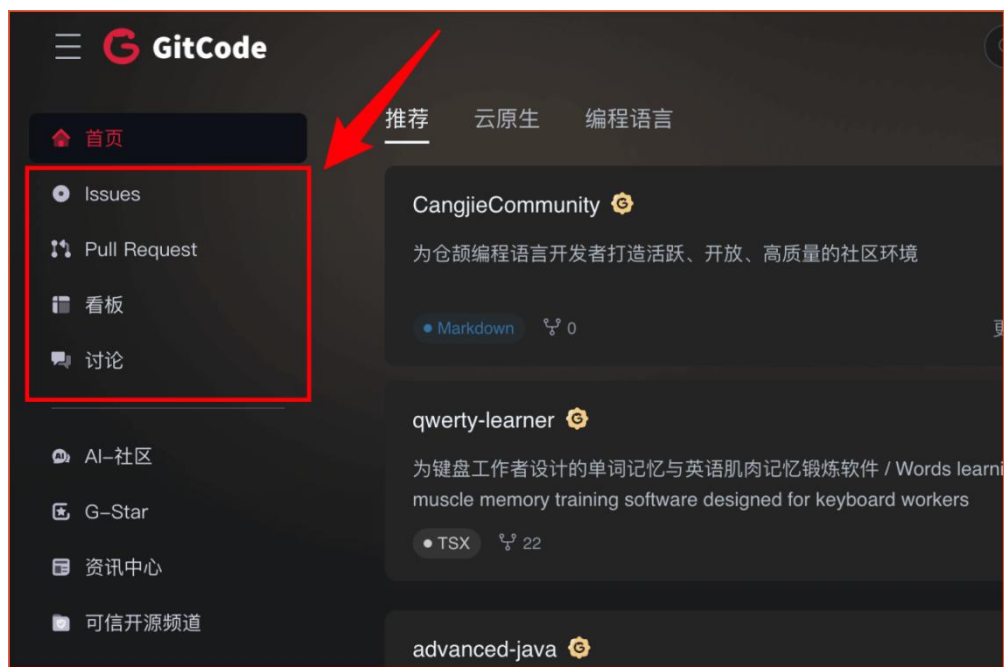
为账号开启多因素认证，增加安全性。**MFA** 可以有效防止未经授权的访问，保护你的账号和数据

- ① 点击右上角头像位置
- ② 下拉菜单中点击个人设置
- ③ 点击左侧【账号管理-账号设置】选项
- ④ 点击多因素认证（**MFA**）下方的【启用】按钮
- ⑤ 用认证程序扫描二维码
- ⑥ 完成账号密码验证（如果未设置密码，请先设置密码）
- ⑦ 输入认证程序中的 6 位数字验证码，并点击【下一步】
- ⑧ 复制并保存恢复码
- ⑨ 点击【下一步】，完成 **MFA** 的设置

1.2、基础设置：个人工作台

当你登录 **GitCode** 之后，你可以通过“个人工作台”查看你个人相关的内容，包括：

- ✓ 与你相关的 **issue**（包括你创建的、你负责的）
- ✓ 与你相关的 **PR**（包括你创建的，分配给你的，需要你评审的）



方式一（推荐）：通过左侧导航进入



方式二：点击个人头像、下拉菜单中点击【工作台】

1.3、开发协作：Repo 开发工具下载

我们提供了与 Gitee 平台功能一致的定制 Repo 工具，支持多代码仓的批量下载、更新和合入，确保鸿蒙开发者在开发流程中的连续性体验。使用方式与以前一致，具体步骤如下：

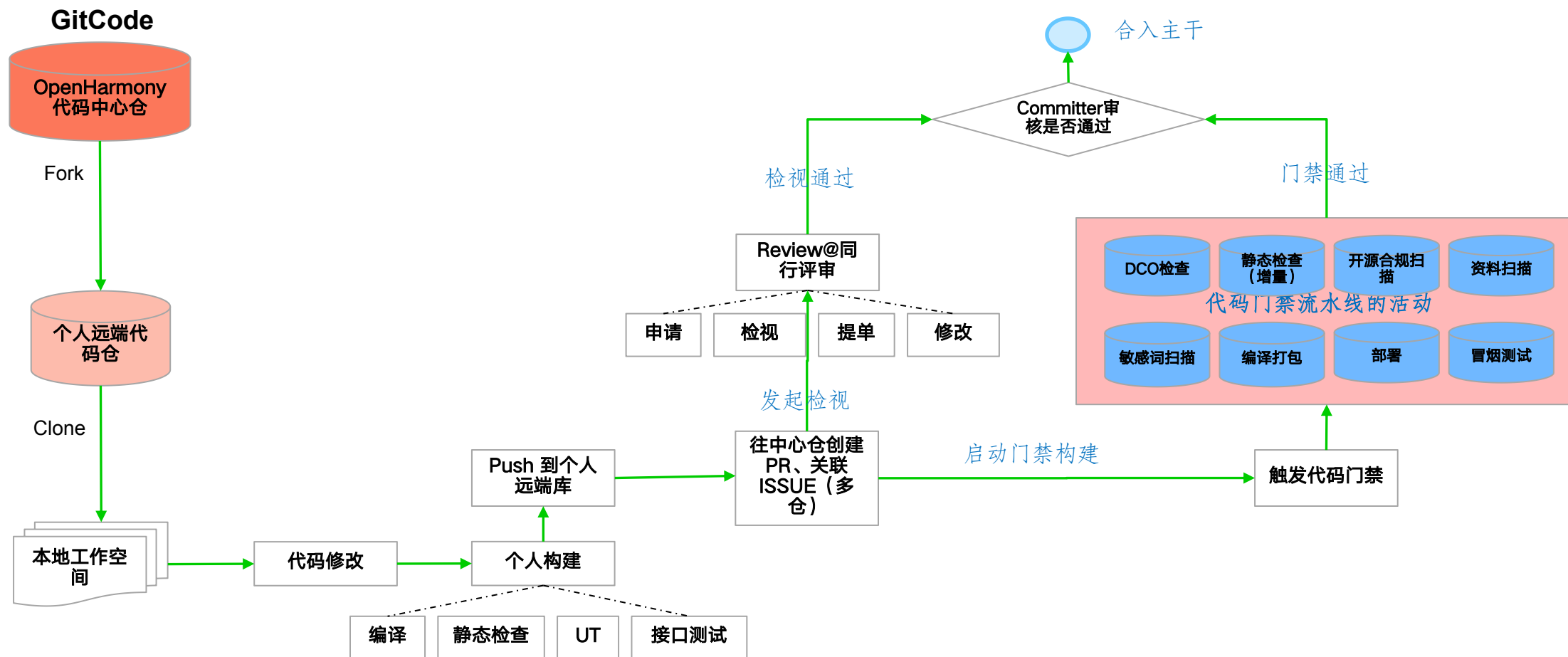
- ① manifest.xml 文件配置
- ② repo 引导命令下载
- ③ repo init 初始化
- ④ repo sync 仓库同步
- ⑤ repo start {BRANCH} [project1, project2] 进行批量分支切换开始开发.....
- ⑥ repo stage 或 repo forall -c git add . 存入文件 git 暂存区
- ⑦ repo config repo.token {ACCESS_TOKEN} 配置 gitcode 个人 API token
- ⑧ repo config repo.pullrequest {True/Fales} 配置是否开启 push 后，向指定分支进行的 PR 提交的特性
- ⑨ repo push -p --br={BRANCH} --d={DEST_BRANCH} --new_branch 用本地的指定分支向远程推送并关联，推送成功后向指定的分支进批量提交
- ⑩ repo sync 或 repo forall -c git pull 进行代码批量同步

更多 Repo 工具使用说明，请访问 <https://gitcode.com/gitcode-dev/repo/blob/main/README.md>

Repo 工具项目地址：<https://gitcode.com/gitcode-dev/repo>

1.4、开发协作：Fork + PR 协作模式

Pull Request 是提交两个项目仓库之间变更的一种方式，通常用于在 **fork** 项目和原始项目之间同步更新，适用于团队协作。与 **Gitee** 一样，参与 **GitCode** 上的仓库开发，也首选并推荐使用 “**Fork + Pull**” 模式。



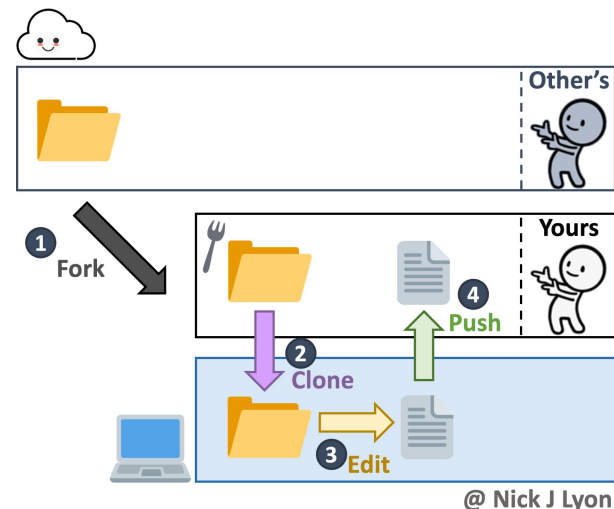
1.4、开发协作：Fork 项目

Fork 工作流程是开源项目中广泛采用的一种合作方式，以下是在 **GitCode** 平台上 **fork** 一个项目的具体步骤：

- ① 访问原始项目，然后点击右上角的「Fork」按钮
- ② 设置 fork 项目名称、项目归属、项目路径等基本信息
- ③ 选择 fork 模式：**fork 主分支(默认)** 或 **fork 完整的项目代码**
- ④ 点击【创建Fork项目】按钮
- ⑤ 等待 fork 完成

注意：

- 你可以选择将项目 **fork** 到个人或者组织，如果 **fork** 到组织，你需要拥有在组织中创建项目的权限
- 同一个 namespace（用户、组织）可以存在同一个项目的多个 **fork** 版本



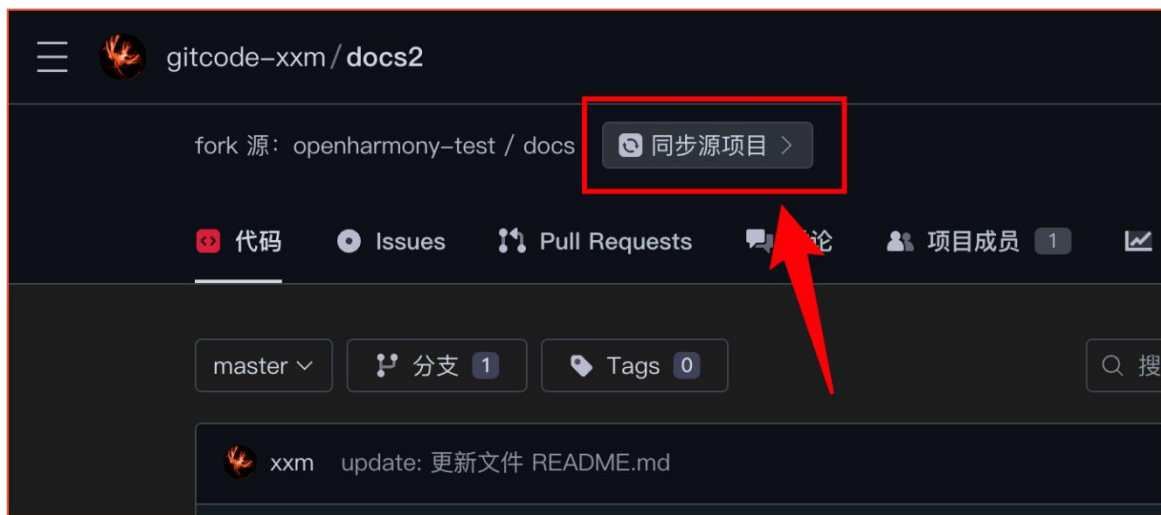
fork workflow: <https://docs.gitcode.com/docs/pulls/fork/>

1.4、开发协作：本地开发和提交

当完成 **fork** 后，接下来可以选择将你的 **fork** 项目克隆到本地计算机上，进行常规的代码编写和开发活动：

- ① 克隆到本地：通过命令 `git clone` 克隆项目，支持 `https`、`git` 两种克隆方式（具体配置方式参考 2.1、基础设置部分）
- ② 创建新分支：在本地仓库中创建一个新的分支来进行开发
- ③ 开发和提交：提交修改并在本地分支上进行推送。使用 `git commit` 来记录提交信息，并将更改推送到 `GitCode` 上的 `fork` 项目中
- ④ 同步更新：在开发过程中，原始项目可能会有新的提交，你可以通过 `GitCode` 提供的【同步源项目】功能进行代码更新

注意：同步源项目会覆盖PR中提交的代码，可能导致空提交，请确保PR合入后再覆盖同步。



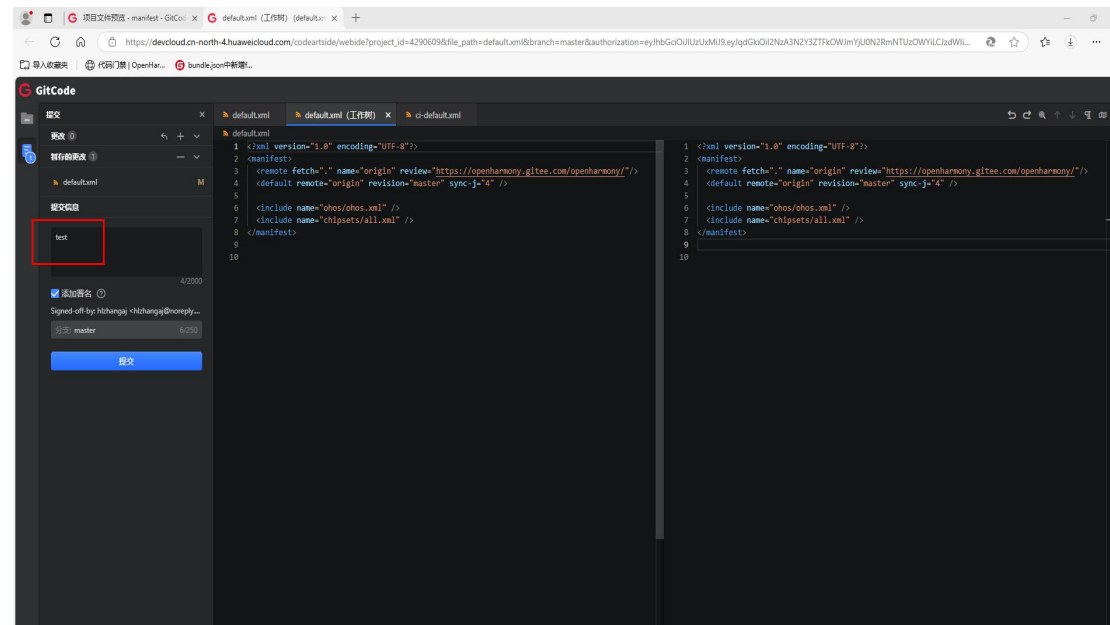
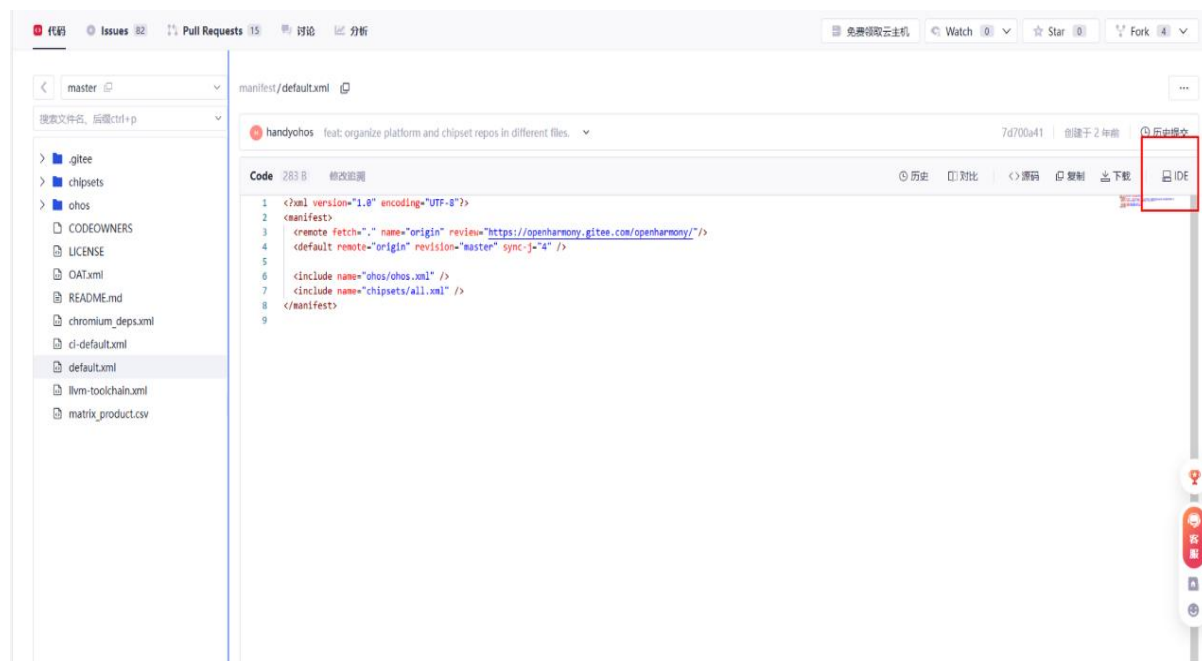
git 常用命令：<https://git-scm.com/docs/git/zh> HANS-CN# git 命令

1.4、开发协作：webIDE方式提交

webIDE的方式顾名思义是直接在网页上进行代码编辑，适合修改较为简单的轻量化提交：

- ① 在网页中直接选中需要修改的代码；
- ② 选中IDE，并进入编辑界面；
- ③ 文件修改完成之后，输入对应的上库描述，点击提交；

注意：同步源项目会覆盖PR中提交的代码，可能导致空提交，请确保PR合入后再覆盖同步。

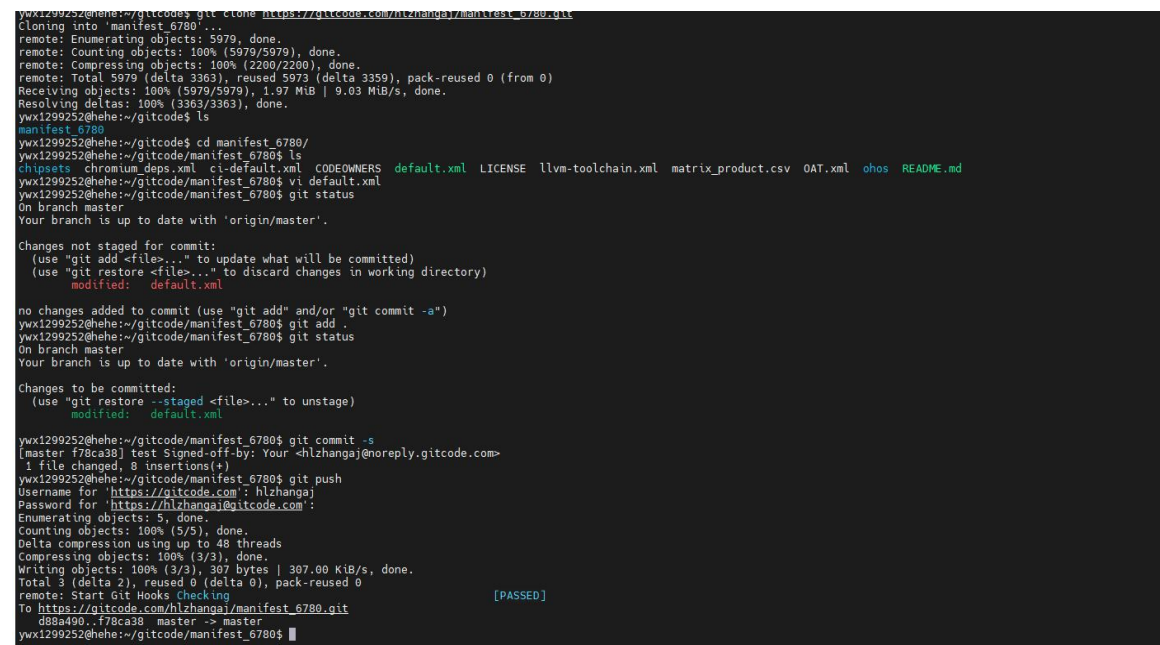
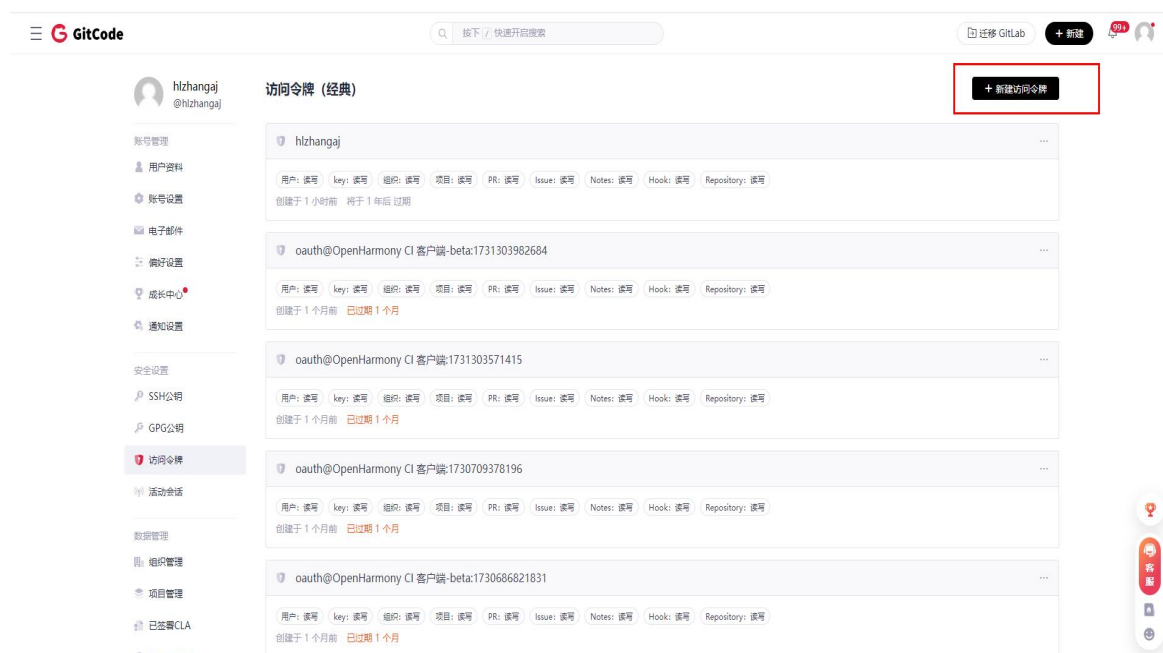


1.4、开发协作：git方式提交

在涉及到较为复杂提交的时候，就需要使用到git方式：

- ① 首先申请个人令牌(和gitee的提交方式有不同，gitee是使用用户名和密码，gitcode使用的是用户名和个人令牌)；
- ② 使用基础的git命令上库，git status->git add->git commit->git push(在git push的最后一步 输入的是刚才申请的个人令牌)

注意：同步源项目会覆盖PR中提交的代码，可能导致空提交，请确保PR合入后再覆盖同步。



git 常用命令：<https://git-scm.com/docs/git/zh> HANS-CN# git 命令

1.4、开发协作：大文件处理

在提交过程中，**gitcode**将单个文件提交的大小限制在**10MB**，强行推送，会存在下图所示的报错，处理方法：

- ① 首先使用**find**命令将所推送文件中大于**10MB**的部分检索出来，记录它们的路径；
- ② 可以使用**git lfs track+文件路径**，将大文件走**lfs track**通道；
- ③ 此时，需要重新对文件重新添加一遍再进行**git push**，那么该文件就会被重新推送了。

注意：同步源项目会覆盖**PR**中提交的代码，可能导致空提交，请确保**PR**合入后再覆盖同步。

```
Uploading LFS objects: 100% (1/1), 17 MB | 0 B/s, done.
Enumerating objects: 11968, done.
Counting objects: 100% (11968/11968), done.
Delta compression using up to 144 threads
Compressing objects: 100% (9644/9644), done.
Writing objects: 100% (11965/11965), 817.65 MiB | 80.12 MiB/s, done.
Total 11965 (delta 2163), reused 11601 (delta 2036), pack-reused 0
remote: Resolving deltas: 100% (2163/2163), completed with 2 local objects.
remote: warning: Repository size (2.5 GiB) exceeds limit (1.0 GiB). Further push might be rejected. Please delete unnecessary objects in your repository.
remote: hint: In the project settings page, use [Project Management]->[Compact]->[Compact] to compact your repository.
remote: Start Git Hooks Checking [FAILED]
remote: Error: Deny by project hooks setting 'default': size of the file 'checkstyle-all.jar', is 16 MiB, which has exceeded the limited size (10 MiB) in commit 'a5e9aa2fca132f3b4193e98c1669ee4de7033062'.
To https://gitcode.com/hlzhangaj/chromium_src.git
! [remote rejected] 132_trunk -> 132_trunk (pre-receive hook declined)
error: failed to push some refs to 'https://gitcode.com/hlzhangaj/chromium_src.git'
mwx1118640@huawei:~/zhanghanlin/0306/chromium_src$ git status
On branch 132_trunk
Your branch is ahead of 'origin/132_trunk' by 3 commits.
(use "git push" to publish your local commits)

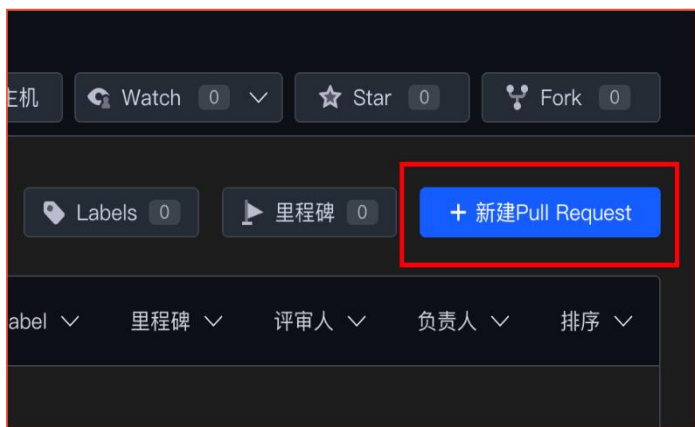
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  (commit or discard the untracked or modified content in submodules)
    modified:   third_party/catapult (modified content)
    modified:   third_party/depot_tools (modified content)
    modified:   third_party/devtools-frontend/src (modified content)

no changes added to commit (use "git add" and/or "git commit -a")
mwx1118640@huawei:~/zhanghanlin/0306/chromium_src$ git lfs track ./third_party/checkstyle/cipd/checkstyle-all.jar
"third_party/checkstyle/cipd/checkstyle-all.jar" already supported
mwx1118640@huawei:~/zhanghanlin/0306/chromium_src$
```

1.4、开发协作：创建 Pull Request

在完成代码开发和提交后，就可以开始通过创建跨项目的 **Pull Request** 向原始项目提交代码修改内容：

- ① 查看 **fork** 项目：查看 GitCode 上的 fork 项目，并点击【Pull Requests】标签
- ② 新建 **Pull Request**：点击右上角的【新建Pull Request】按钮，并选择要合并的分支和目标分支，然后点击【下一步】
- ③ 填写 **PR** 信息：添加 PR 的标题和描述，描述你的代码更改的目的和内容
- ④ 提交 **Pull Request**：点击【创建】按钮，完成 PR 的创建

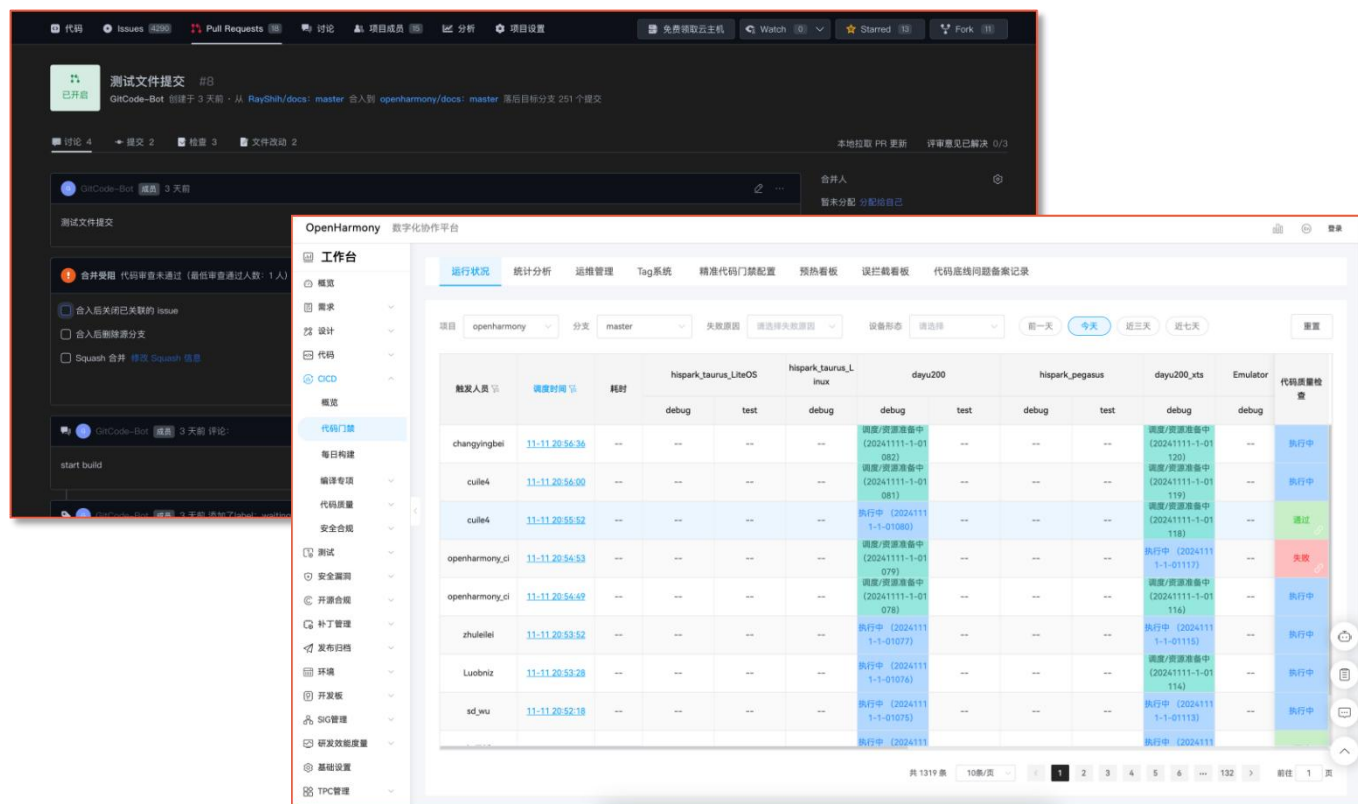


PR 描述中的 **issue** 地址，在 PR 创建成功后，PR 会默认关联上该 **issue**

1.4、开发协作：启动门禁构建

与 Gitee 类似，在完成 PR 创建后，需要通过 `start build` 命令触发编译构建测试。构建门禁通过后，CI Bot 会提醒审核人员进行 PR 检视。常用的 PR 命令包括：

- ❑ `start build`：触发编译构建测试
- ❑ `stop build`：停止编译构建测试
- ❑ `submit`：自动合入异常时，手工触发合入
- ❑ `check dco`：人工触发检查 DCO
- ❑ `static-check`：手工触发静态检查



The screenshot displays the OpenHarmony digital collaboration platform interface. The top section shows a Pull Request for '测试文件提交' (Test File Submission) with a 'start build' button. The main area is a dashboard titled 'OpenHarmony 数字化协作平台' with a sidebar menu. The '运行状况' (Run Status) tab is active, showing a table of build results for various components.

项目	分类	失败原因	设备形态	运行状况	统计	分析	运维管理	Tag系统	精准代码门禁配置	预检看板	误报看板	代码底线问题备案记录
hispark_taurus_LiteOS	debug	test	debug	debug	test	debug	test	debug	test	debug	debug	代码质量检查
changyingbei	11-11 20:56:36	执行中
cuile4	11-11 20:56:00	执行中
cuile4	11-11 20:55:52	通过
openharmoy.ci	11-11 20:54:53	失败
openharmoy.ci	11-11 20:54:49	执行中
zhuleilei	11-11 20:53:52	执行中
Luobniz	11-11 20:53:28	执行中
sd_yu	11-11 20:52:18	执行中

更多常用命令参考：https://gitcode.com/openharmony/community/blob/master/zh/infrastructure/build_command.md

1.4、开发协作：下载构建结果

除门禁构建外，每日构建操作也与之前保持一致。你可以通过每日构建和滚动构建来获取版本的下载链接：

项目openharmony分支master日期2024-11-10至2024-11-10前一天今天前一月本月重置

版本级流水线

100%成功率

8运行次数

平均耗时13

26Mini

154Small

Standard

每日构建

滚动构建

流水线名称	设备类型	构建失败次数	构建总数	构建成功率	冒烟成功率	耗时
hispark_pegasus	Mini	0	1	100%	0%	13
hispark_taurus_LiteOS	Small	0	1	100%	--	25
ohos-sdk-public	Standard	0	1	100%	--	67
hispark_taurus_Linux	Small	0	1	100%	--	27
ohos-sdk-full	Standard	0	1	100%	--	80
dayu200	Standard	0	1	100%	100%	50
mac-sdk-m1-public	Standard	0	1	100%	--	42

共 8 条

每日构建

项目openharmony分支master日期2024-11-10至2024-11-10前一天今天前一月本月重置

版本级流水线

68%成功率

170运行次数

平均耗时12

17Mini

61Small

Standard

每日构建

滚动构建

流水线名称	设备类型	构建失败次数	构建总数	构建成功率	冒烟成功率	耗时	运行周期	版本下载地址
mac-sdk-m1-public-test	Standard	0	3	100%	--	67min	0次/天	下载链接
hispark_taurus_Linux_22.04	Small	0	3	100%	--	23min	6次/天	下载链接
hispark_pegasus_22.04	Mini	0	3	100%	0%	17min	5次/天	全量包: 下载链接 全量包.sha256: 下载链接 镜像包: 下载链接 签名文件: 下载链接
hispark_pegasus	Mini	0	2	100%	0%	10min	5次/天	下载链接
weex-loader	Standard	5	5	0%	--	--	7次/天	下载链接
hispark_taurus_LiteOS_22.04	Small	0	3	100%	--	20min	5次/天	下载链接
hispark_taurus_LiteOS	Small	0	2	100%	--	13min	5次/天	下载链接

共 77 条50条/页12>前往1页

滚动构建

1.4、开发协作：PR 审查、测试和评审

当有用户向你的项目提交 **PR** 后，你会收到系统通知。你可以对提交的代码进行审查，确保其符合项目规范并达到代码质量要求。在完成审查后，可以选择是否合并这些更改：

- ✓ **代码审查**：仔细检查提交的代码，确保其功能、逻辑和风格符合项目要求
- ✓ **CODEOWNERS**：确认项目中的 CODEOWNERS（责任田）已参与审查，如果该文件中指定了代码所有者，他们需要进行审核并批准变更
- ✓ **代码测试**：运行测试来验证代码的正确性，确保新代码不会引入新的 bug 或影响现有功能
- ✓ **代码评审**：根据项目的 Code Review 标准进行全面评审，确保代码遵循最佳实践

完成上述审查流程后，你可以将 **PR** 合并到目标分支。这意味着 **PR** 作者正式成为项目的代码贡献者，并记录在项目的贡献历史中。

注意：

- 项目维护者可以指定 **PR** 的默认审查人员、测试人员
- 项目维护者可以指定 **PR** 的评审人员，但社区上的任何用户都可以对公开项目的 **PR** 进行 CodeReview 评审
- **PR** 合入都由门禁管控，由 CI 自动合入，仅支持管理员合入（注：**OH_Committer** 不再提供管理员权限）

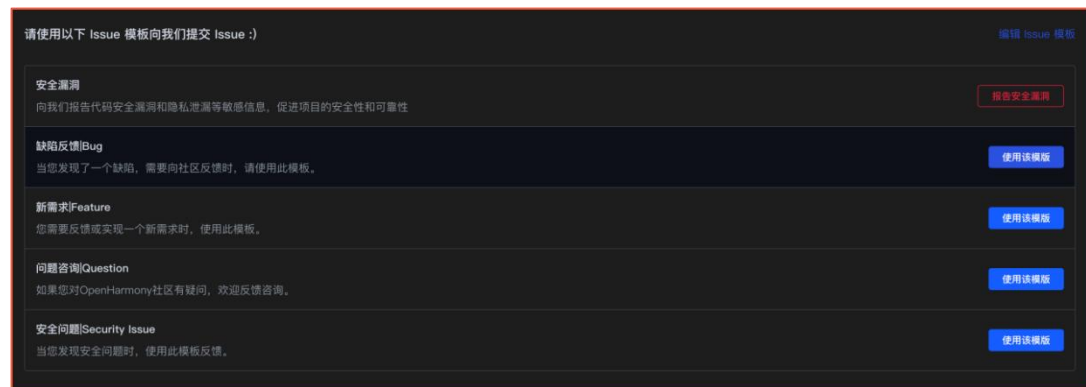
1.5、开发协作：Issues

Issue 可以用于跟踪和管理项目中的问题、**bug**、任务和改进请求，它是开发者在 **GitCode** 上进行协作和解决问题的关键部分：

- ① 在项目导航栏中，点击【Issues】标签
- ② 单击【+新建 Issue】按钮
- ③ 选择对应的 issue 模板，并点击【使用该模板】
- ④ 填写 Issue 标题和描述，确保描述足够清晰，以便其他人能够理解问题或任务
- ⑤ 点击【提交 Issue】按钮，创建 Issue

1、与 Gitee 的 issue 编号不同，**GitCode** 项目中的 issue 使用与 **PR** 类似的自增 iid（在同一项目内唯一），要定位到某个具体的 issue，需结合 iid 和项目信息一起使用

2、**GitCode** 限制了非项目成员 @项目成员的功能（非项目成员无法查看项目成员列表），但允许手动输入 @用户名



更多关于 issue 的介绍：https://docs.gitcode.com/docs/help/home/org_project/issue/

1.5、开发协作：Issues 模板

与 GitHub 和 Gitee 类似，GitCode 也支持 issue 模板（包括项目模板和组织模板）。GitCode 的 issue 模板是预先填充的表单，用于引导用户提供所有必要的细节，从而提高问题处理效率，确保维护者获得解决问题所需的完整信息：

- ✓ 项目 issue 模版，依次支持 .github、.gitcode、.gitee 目录
- ✓ 组织 issue 模版，仅支持组织下的 .gitcode 项目中的 .gitcode 目录
- ✓ 支持模版类型： markdown 格式模版、yaml 模版

```
---
name: "Bug 报告"
about: "报告一个问题帮助我们改进"
title: "【BUG】:"
labels: ["BUG"]
assignees: 'username'
---

### BUG 类型

<!--
请在这里描述你所遇到的 BUG 类型，以便我们更快定位问题，比如 UI、功能、体验等
-->

### 复现步骤

<!--
请在这里描述你遇到该 BUG 时的页面及步骤
-->
```

```
name: Bug 报告
description: 报告一个问题帮助我们改进
title: "【BUG】 "
labels: ["bug"]
assignees:
  - username
body:
  - type: markdown
    attributes:
      value: |
        ## 感谢您的报告!
        请在提交之前，检查该问题是否已经被报告过。

  - type: input
    id: what-happened
    attributes:
      label: 发生了什么?
      description: 请尽可能详细地描述问题。
      placeholder: 请在这里输入详细信息
    validations:
      required: true
```

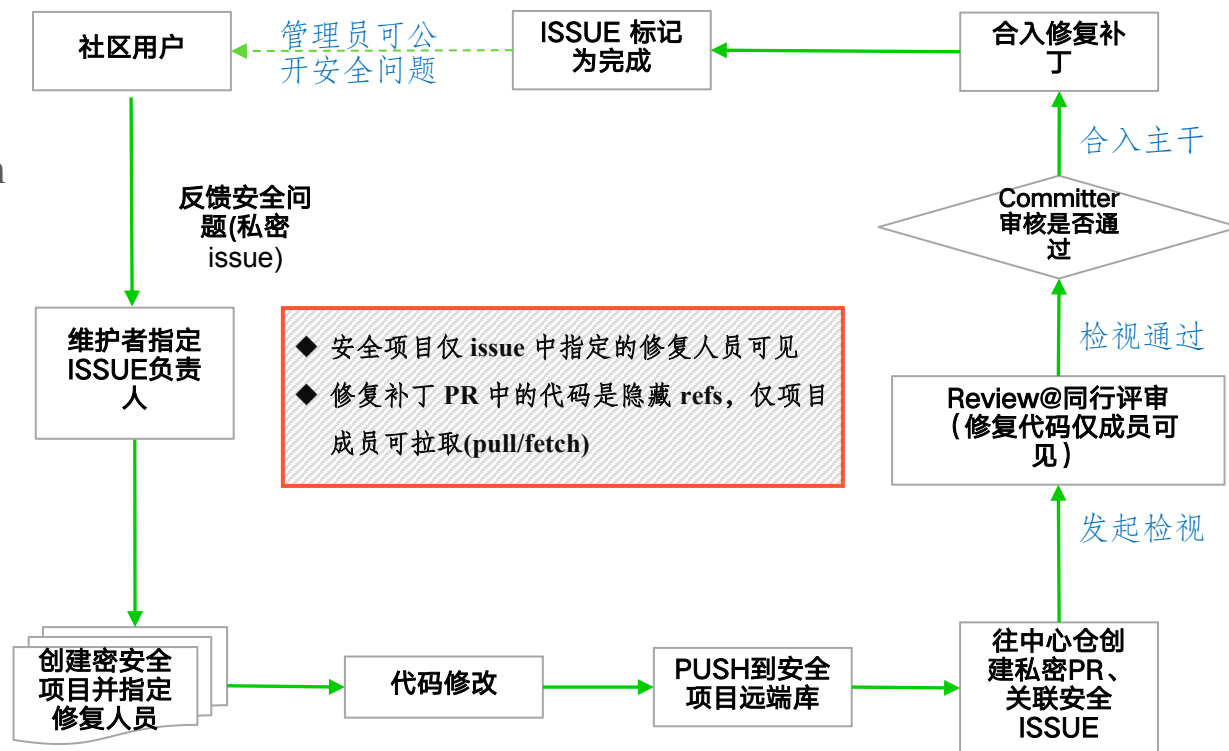
更多关于 issue 模版的介绍：https://docs.gitcode.com/docs/help/home/org_project/issue/issue_template/

1.6、开发协作：安全 issue 修复补丁（白名单）

GitCode 内置了安全 issue 功能。当项目管理员启用该功能后，用户提交的安全 issue 仅项目成员和 issue 作者可见，其他非成员无法查看。对于白名单组织，平台还提供了修复补丁的功能流程：


- ① 指派安全 issue 的处理人
- ② 处理人可选择需要修复补丁的项目，并创建安全修复的 fork 项目（该 fork 项目仅对指派人员同步）
- ③ 指派人员在 fork 项目中对安全问题进行修复并提交代码
- ④ 修复完成后，提交 PR，PR 仅项目成员可见
- ⑤ PR 提交的代码带有特殊的 refs，仅成员可以 clone 和 fetch
- ⑥ 补丁修复 PR 被合并
- ⑦ 安全 issue 标记为已完成，管理员可以选择将该 issue 公开

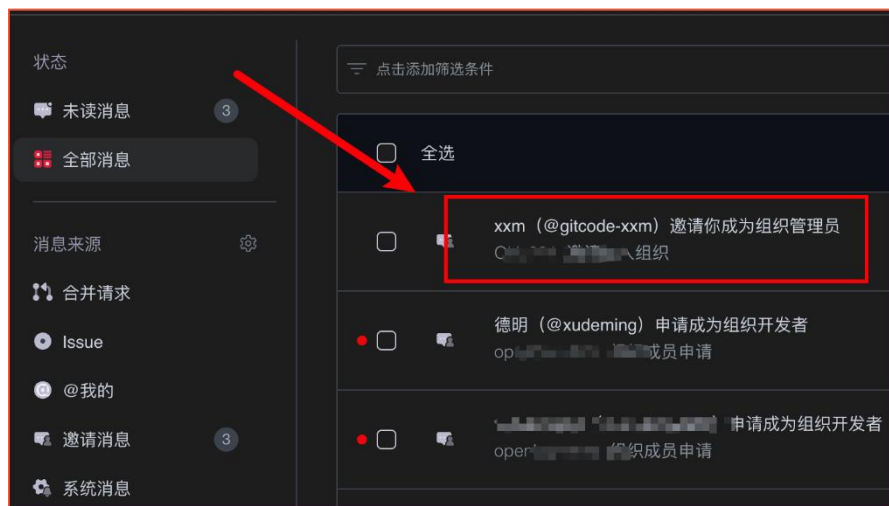
注：如果需要启用安全 issue 修复补丁功能，请务必在【项目设置】中开启项目中的【安全漏洞】模块



1.7、加入组织/项目：接受邀请（1）

当 **GitCode** 用户被邀请加入组织/项目后，系统会通过通知中心、邮件等方式通知用户，用户可以通过通知中的链接来看邀请信息，并选择是否接受邀请。

- ① 访问 **GitCode** 网站，点击右上角【通知中心】
- ② 筛选【邀请消息】，并查看邀请链接
- ③ 在邀请页面中查看邀请信息，包括被邀请加入的组织/项目信息、角色信息等
- ④ 被邀请用户可以选择【加入】或【拒绝】



① 进入通知中心查看邀请消息



② 查看邀请信息并作出选择

1.7、加入组织/项目：加入申请（2）

当 GitCode 用户点击邀请链接后，用户将会通过邀请链接生成一个加入组织/项目的申请，当组织/项目管理员、维护者审核通过后，即可成为组织/项目的成员。

- ① 登录 GitCode 网站，并点击【邀请链接】，将自动提交加入申请
- ② 等待管理员、维护者审核申请，通过后即可加入组织/项目



① 通过邀请链接提交申请



② 管理员、维护者查审核申请，并作出选择

1.8、寻求帮助

以上是 **GitCode** 为鸿蒙开发者准备的使用指南，在使用过程中如果你遇到任何问题或疑问，欢迎通过以下方式联系我们，我们将竭诚为鸿蒙开发者提供全力支持！

在线客服

通过网站右下角的在线客服寻求帮助

- 在线客服（推荐）
- issue 反馈
- 客服热线4006868951
- 邮件联系（kefu@gitcode.com）

帮助文档

浏览 **GitCode** 帮助文档获取帮助

- <https://docs.gitcode.com>