Business Economics

Record

Author: Player_He

Publisher: Angel

August 26, 2025

目　录

0.1 Git 项目提交远程仓库全流程指南

核心要义：通过五步标准化操作完成本地项目到 GitHub 的完整提交，涵盖仓库初始化、文件追踪、分支管理、远程关联与推送策略，解决常见错误场景。

0.1.1 基础操作流程

1. **初始化本地仓库**
   创建项目目录并进入，执行初始化命令：
   ```
   mkdir project-name && cd project-name
   git init  # 生成.git 隐藏目录
   ```
2. **添加文件到暂存区**
   创建项目文件后执行追踪操作：
   ```
   touch README.md  # 创建示例文件
   git add .  # 添加所有文件（或指定文件名）
   ```
3. **提交到本地仓库**
   记录版本快照并添加描述：
   ```
   git commit -m " 初始化项目"  # 提交信息需明确
   ```
4. **关联远程仓库**
   复制 GitHub 仓库 HTTPS/SSH 链接后执行：
   ```
   git remote add origin https://github.com/HR689hHN/Computer.git
   ```
5. **推送到远程分支**
   首次推送需建立分支关联：
   ```
   git push -u origin main  # -u 设置默认上游分支
   ```

0.1.2 关键配置说明

1. **身份信息配置**
   首次使用 Git 必须设置全局身份（与 GitHub 一致）：
   ```
   git config --global user.name "YourName"
   git config --global user.email "email@example.com"
   ```
2. **分支命名规范**
   现代 Git 默认使用 `main` 分支（历史用 `master`）：
   ```
   git branch -m master main  # 旧分支重命名
   ```
3. **文件忽略规则**
   创建`.gitignore` 过滤非追踪文件：

```
# 示例内容
node_modules/
.env
*.log
```

0.1.3 高级场景处理

1. **推送冲突解决**
   当远程有本地缺失的提交（如初始化 README）：
   ```
   git pull origin main --rebase  # 变基合并
   git push origin main
   ```
2. **远程关联错误修正**
   ”origin 已存在” 时的处理方案：
   ```
   git remote set-url origin 新 URL  # 更新仓库地址
   git remote rm origin  # 删除后重新添加
   ```
3. **空仓库推送策略**
   本地无提交内容时需先创建初始提交：
   ```
   echo "# Project" > README.md
   git add . && git commit -m "init"
   ```

0.1.4 操作验证命令

查看仓库状态 `git status`
检查远程关联 `git remote -v`
验证分支关联 `git branch -vv`
查看提交历史 `git log --oneline`

最佳实践：推送前执行 `git status` 确认无未跟踪文件，使用 `git push` 前优先 `git pull --rebase` 避免冲突，敏感信息务必写入`.gitignore`。

0.2 后续修改提交远程仓库标准流程

核心策略：基于主分支（main/master）直接提交或创建特性分支并行开发，通过四步标准化操作实现高效更新，匹配不同协作场景需求。

0.2.1 分支决策模型

1. **直接主分支提交**
   适用场景：个人项目/紧急修复/微小变更
   流程：工作目录修改 → 暂存 → 提交 → 推送
   优势：路径最短，适合独立开发者
2. **创建特性分支提交**
   适用场景：团队协作/功能开发/长期修改
   流程：创建分支 → 开发 → 本地测试 → 推送分支 → PR 合并
   优势：隔离风险，支持并行开发

0.2.2 主分支直接提交流程

1. **修改工作文件**
   在项目目录编辑代码文档：
   `vim index.html  # 或 IDE 可视化编辑`
2. **查看变更状态**
   确认修改范围及内容：
   `git status  # 显示红标未暂存文件`
   `git diff  # 查看具体代码变动`
3. **暂存并提交变更**
   选择需提交的修改：
   `git add .  # 添加所有修改`
   `git add src/  # 添加特定目录`
   `git commit -m " 修复登录页面样式异常"`
4. **推送到远程仓库**
   同步至云端（首次后无需-u 参数）：
   `git push origin main  # 关联后简化为 git push`

0.2.3 特性分支开发流程

1. **创建开发分支**
   基于主分支新建工作分支：
   `git checkout -b feat/user-profile`
   命名规范：`feat/`功能 `fix/`修复 `docs/`文档
2. **分支内迭代开发**
   在独立分支完成修改：

`git add . && git commit -m " 增加头像上传功能"`

※ 可多次提交形成开发历史

3. **推送到远程分支**

   上传分支至 GitHub：

   `git push -u origin feat/user-profile`

4. **合并到主分支**

   通过 PR/MR 完成集成：

   - GitHub：创建 Pull Request
   - GitLab：创建 Merge Request
   - 命令行合并：`git checkout main && git merge feat/user-profile`

0.2.4 关键注意事项

冲突预防	推送前执行 `git pull --rebase` 避免版本冲突
原子提交	单次提交仅完成单一功能（避免混合修改）
信息规范	提交消息格式：`< 类型 >(< 范围 >): < 描述 >`
分支清理	合并后删除本地分支：`git branch -d feat/user-profile`
敏感防护	修改内容若含密钥需重置历史：`git filter-repo --force`

操作效率工具链：

- `git stash`：临时保存未完成修改
- `git restore`：撤销工作区修改
- `.gitignore`：配置忽略文件模板
- GitHub Desktop：可视化分支管理