Business Economics

Record

Author: Player_He

Publisher: Angel

August 26, 2025

目　录

0.1 远程分支创建与提交全流程指南

一句话总结：通过本地分支关联远程仓库实现分支创建与提交，核心流程为：本地创建分支 → 推送至远程 → 开发提交 → 定期同步。

0.1.1 完整操作流程

1. **创建本地分支**
   ```
   $ git checkout -b feature/login   // 创建并切换到新分支
   ```
2. **关联远程仓库**
   ```
   $ git push -u origin feature/login
   ```
   → 推送分支到远程并建立追踪关系（首次推送）
3. **开发并提交代码**
   ```
   $ touch login.js
   $ git add login.js
   $ git commit -m " 添加登录页面"
   $ git push   // 后续提交只需 git push
   ```
4. **同步远程更新**
   ```
   $ git pull origin feature/login   // 获取他人提交
   ```

0.1.2 关键步骤详解

1. 创建本地分支

- **基于当前分支创建**：
  `git branch < 新分支名 > + git checkout < 新分支名 >`
- **基于特定提交创建**：
  `git checkout -b fix/issue-23 8a3f5e9`
- **克隆远程分支**：
  `git checkout -b dev origin/dev`

2. 推送至远程仓库

命令	作用
`git push -u origin < 分支名 >`	首次推送并建立追踪
`git push origin < 分支名 >`	后续推送更新
`git push --set-upstream origin < 分支名 >`	为现有分支设置上游
`git push -f`	强制覆盖（慎用）

3. 验证远程分支

- **查看远程分支**:
  `git branch -r` // 查看远程分支列表
- **检查关联状态**:
  `git branch -vv`
  → 显示 `feature/login [origin/feature/login]`

0.1.3 分支状态管理

状态	检测命令	处理方案
本地领先	`git status` 提示"ahead"	立即推送
远程领先	`git status` 提示"behind"	先执行 `git pull`
冲突状态	`git pull` 提示冲突	解决冲突后提交
分支游离	`HEAD detached` 警告	创建临时分支保存

0.1.4 多场景应用示例

场景 1：新功能开发

```
# 创建登录功能分支
git checkout -b feature/login
git push -u origin feature/login

# 开发并提交
echo "function login(){}" > login.js
git add login.js
git commit -m "实现登录函数"
```

```
git push

# 创建PR/MR合并请求（GitHub/GitLab）
```

场景 2：紧急热修复

```
# 基于生产分支创建修复分支
git checkout production
git pull
git checkout -b hotfix/auth-bug

# 修复并推送
vim auth.js
git commit -am "修复权限验证漏洞"
git push -u origin hotfix/auth-bug

# 立即部署到生产环境
```

场景 3：团队协作开发

```
# 获取队友新建的远程分支
git fetch origin
git checkout -b feature/payment origin/feature/payment

# 协作开发
git add payment.js
git commit -m "添加支付处理逻辑"
git push

# 同步队友提交
git pull
```

0.1.5 最佳实践

- **命名规范**：使用 `feature/`、`fix/` 等前缀
- **及时推送**：每日工作结束前推送代码
- **分支清理**：合并后删除远程分支：

```
git push origin --delete feature/login
```

- **权限控制**：保护主分支，强制 Code Review
- **钩子应用**：配置 pre-push 钩子运行测试

0.1.6 故障排除

1. **推送失败（无权限）**

```
$ git remote -v   // 检查远程地址
$ git remote set-url origin https://<token>@github.com/user/repo.git
```

2. **追踪关系丢失**

```
$ git branch -u origin/feature/login
```

3. **误删远程分支**

```
$ git reflog   // 查找分支最后提交
$ git push origin 8a3f5e9:feature/login   // 恢复分支
```

0.2 远程分支删除操作指南

一句话总结：使用 `git push` 或 `git branch` 命令可安全删除远程分支，配合 `--delete` 参数精确控制清理范围，保持仓库整洁。

0.2.1 删除方法详解

1. **标准删除命令**

```
$ git push origin --delete feature/login
```

→ 删除远程的 feature/login 分支

2. **等效简写命令**

```
$ git push origin :feature/login
```

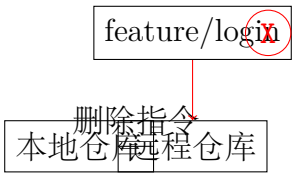
→ 冒号语法等效于 –delete

3. **批量删除分支**

```
$ git push origin --delete feat/old hotfix/test
```

→ 同时删除多个分支

0.2.2 操作流程图示



0.2.3 删除前验证步骤

1. **检查分支状态**

```
$ git branch -a   // 确认分支存在
  main
* feature/login
  remotes/origin/feature/login
```

2. **确认合并状态**

```
$ git log origin/main..origin/feature/login
```

→ 检查是否有未合并的提交

3. **权限验证**

// 确保有远程仓库删除权限

0.2.4 删除后清理操作

命令	作用
`git fetch -p`	清除本地缓存的远程分支记录
`git branch -D feature/login`	删除本地关联分支
`git remote prune origin`	等效于 `git fetch -p`

0.2.5 特殊场景处理

1. **恢复误删分支**

```
$ git reflog   // 查找分支最后提交
$ git push origin 8a3f5e9:refs/heads/feature/login
```

2. **删除受保护分支**

→ 需在 GitLab/GitHub 设置中取消分支保护

3. **无权限删除**

→ 联系仓库管理员执行删除

0.2.6 最佳实践

- **命名规范**：使用前缀如 `feat/`、`fix/` 便于识别
- **自动化清理**：配置 CI/CD 自动删除合并后分支
- **定期维护**：每月执行批量清理
  `git branch -r | grep 'origin/feat/' | sed 's/origin//' | xargs git push origin --delete`
- **权限控制**：限制成员删除主分支权限
- **备份机制**：重要分支删除前创建标签
  `git tag archive/feature/login-v1 origin/feature/login`

0.2.7 GUI 工具操作

- **GitHub**：仓库页面 → Branches → 分支右侧垃圾桶图标
- **GitLab**：仓库 → Branches → 分支右侧删除按钮
- **VS Code**：分支视图 → 远程分支右键"Delete Branch"
- **GitKraken**：远程分支右键"Delete origin/feature/login"

0.2.8 删除状态验证

```
# 删除前
$ git ls-remote origin
8a3f5e9...refs/heads/feature/login

# 删除后
$ git ls-remote origin
[feature/login 分支消失]
```

0.3 误删分支恢复操作指南

一句话总结：通过 Git 的引用日志和分支追踪机制，可高效恢复误删分支，核心步骤为：定位历史提交 → 重建分支指针 → 同步远程仓库。

0.3.1 恢复流程详解

1. **查找分支最后提交**
   `$ git reflog` // 显示所有操作历史

```
8a3f5e9 (HEAD -> main) HEAD@{0}: checkout: moving from feature/login to main
d2b4c6c HEAD@{1}: commit: 用户登录优化
8a3f5e9 HEAD@{2}: checkout: moving from main to feature/login
```

2. **重建本地分支**

```
$ git branch feature/login d2b4c6c   // 基于提交哈希重建
```

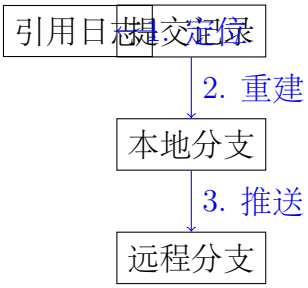
3. **验证分支内容**

```
$ git checkout feature/login
$ git log --oneline   // 确认提交历史完整
```

4. **恢复远程分支**

```
$ git push -u origin feature/login
```

0.3.2 恢复原理图示



0.3.3 关键恢复技术

1. 引用日志查询

- `git reflog`: 显示所有 HEAD 变更记录
- `git reflog show --all`: 显示所有引用变更
- 时间过滤: `git reflog --since="2 days ago"`

2. 分支重建方法

场景	命令
已知提交哈希	`git branch < 分支名 > <commit_hash>`
最后操作记录	`git branch < 分支名 > HEAD@{1}`
远程分支恢复	`git push origin <commit_hash>:refs/heads/< 分支名 >`

3. 恢复验证

- 提交历史：`git log -5 --oneline`
- 文件验证：`git show HEAD:filename`
- 分支关联：`git branch -vv`

0.3.4 特殊场景处理

1. **引用日志被清除**

```
$ git fsck --full   // 检查悬空对象
$ git show d2b4c6c   // 手动验证提交
```

2. **仅删除远程分支**

```
$ git checkout -b feature/login
$ git push -u origin feature/login
```

3. **分支名称遗忘**

```
$ git log --all --grep=" 登录功能"   // 按提交信息搜索
```

0.3.5 预防措施

- **分支保护**：
  - GitHub/GitLab 设置保护分支
  - `git config --global branch.autosetupmerge always`
- **定期备份**：
  - 创建存档标签：`git tag archive/feature/login-v1 feature/login`
  - 推送备份仓库：`git push backup-repo --all`
- **操作规范**：
  - 使用 `git branch -d` 而非 `-D`
  - 删除前执行：`git merge --no-ff feature/login`
- **工具支持**：
  - IDE 分支管理（VSCode/GitKraken）
  - 日志可视化：`git log --graph --all --oneline`

0.3.6 恢复工作流示例

```
# 场景：误删feature/login分支
$ git branch -D feature/login

# 步骤1：查询操作记录
```

```
$ git reflog
d2b4c6c HEAD@{3}: commit: 用户登录优化
8a3f5e9 HEAD@{4}: checkout: moving from main to feature/login

# 步骤2: 重建分支
$ git branch feature/login d2b4c6c

# 步骤3: 验证内容
$ git checkout feature/login
$ ls login.js  # 确认关键文件存在

# 步骤4: 恢复远程
$ git push -u origin feature/login

# 步骤5: 确认恢复
$ git branch -a
  feature/login
  remotes/origin/feature/login
```

0.3.7 图形化工具操作

- **VS Code**:
  - GitLens 插件 → REFERENCE HISTORY
  - 右键提交记录 → "Create Branch..."
- **GitKraken**:
  - 左侧菜单 → REFLOG
  - 右键提交 → "Create branch here"
- **Sourcetree**:
  - 视图 → "Reflog"
  - 右键记录 → " 创建分支"

0.4 远程仓库文件删除操作指南

一句话总结：通过 `git rm` 命令移除本地文件并提交，再推送到远程仓库可永久删除文件/文件夹，需注意历史记录仍可通过 Git 追溯。

0.4.1 文件删除流程

1. **删除单个文件**

```
$ git rm config.yml   // 移除文件并暂存删除操作
$ git commit -m " 移除冗余配置文件"
$ git push origin main
```

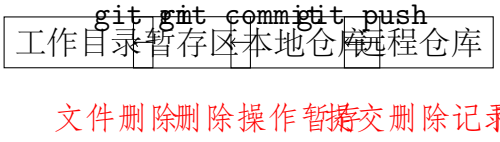
2. **删除整个文件夹**

```
$ git rm -r logs/   // 递归删除目录
$ git commit -m " 清理日志目录"
$ git push
```

3. **仅从 Git 删除（保留本地）**

```
$ git rm --cached sensitive.key   // 移除追踪但保留本地文件
```

0.4.2 操作原理图示



0.4.3 关键参数解析

命令	作用
`git rm <file>`	删除文件并停止追踪
`git rm -r <dir>`	递归删除目录
`git rm --cached`	仅停止追踪（保留本地文件）
`git rm -f`	强制删除已修改文件
`git rm -n`	模拟删除（显示将删除的文件）

0.4.4 特殊场景处理

1. **恢复误删文件**

```
$ git checkout HEAD~1 -- config.yml   // 恢复上一版本文件
```

2. **彻底清除历史文件**

```
$ git filter-branch --tree-filter 'rm -f password.key'
```

→ 重写历史永久删除（需强制推送）

3. **删除远程已存在文件**

```
$ git rm --cached -r .idea
$ echo ".idea" >> .gitignore
$ git add .gitignore
$ git commit -m " 停止追踪 IDE 配置"
$ git push
```

0.4.5 删除验证步骤

1. 本地确认文件删除:
   `ls | grep filename`
2. 检查远程状态:
   `git ls-remote --heads origin`
3. 克隆新副本验证:
   ```
   git clone https://repo-url tmp-dir
   cd tmp-dir && ls
   ```

0.4.6 最佳实践

- **操作前备份**: 重要文件先本地备份
- **小步操作**: 分批删除避免大型提交
- **更新.gitignore**: 防止文件再次被追踪
- **权限确认**: 确保有远程仓库写入权限
- **通知团队**: 删除后通知其他成员更新

0.4.7 批量删除示例

```
# 删除所有.tmp文件
git rm **/*.tmp

# 删除空目录
find . -type d -empty -exec git rm -r {} \;

# 提交并推送
git commit -m "清理临时文件"
git push origin main
```

0.4.8 GUI 工具操作

- **VS Code**：资源管理器右键文件 → ”Delete” → 提交推送
- **GitHub Desktop**：文件列表右键 → ”Discard Changes” → 提交推送
- **Sourcetree**：文件状态视图选中文件 → ”Remove” → 提交推送