CryptoOracle 项目 Git 版本控制与实战指南

本文档提供了从基础配置到高级问题解决的完整 Git 指南，旨在帮助团队成员安全、高效地管理代码。

⚠ 核心安全警告 (Security First)

在执行任何 Git 操作前，请务必确认以下文件**已被包含在 `.gitignore` 中**，防止 API Key 和敏感配置泄露：

- ❌ `.env` (包含环境变量和密钥 - **绝对禁止提交**)
- ❌ `config.json` (包含实际交易配置 - **绝对禁止提交**)
- ❌ `log/*.log` (包含运行日志)
- ❌ `data/pnl_history.csv` (包含资金流水)
- ❌ `__pycache__/` (Python 缓存文件)
- ❌ `test/__pycache__/` (测试缓存文件)

注：项目已预置 `.gitignore` 文件，请勿随意删除或修改其中的敏感文件规则。

---

1. 环境初始化 (Initialization)

1.1 全局配置 (首次运行)

告诉 Git 您是谁，这将显示在提交历史中。

```
# 配置用户名
git config --global user.name "您的名字 (拼音或英文)"

# 配置邮箱
git config --global user.email "your_email@example.com"

# (可选) 让 Git 输出带颜色，更易读
git config --global color.ui true
```

1.2 初始化新仓库

如果您是项目发起人，在项目根目录下执行：

```
git init
```

1.3 关联远程仓库

将本地代码关联到 GitHub/GitLab/Gitee。

```
# 添加远程仓库别名为 origin
git remote add origin https://github.com/your_username/your_repo.git

# 查看当前关联的远程仓库
git remote -v
```

---

2. 日常开发标准流程 (Standard Workflow)

养成良好的习惯：**“拉取 -> 修改 -> 暂存 -> 提交 -> 推送”**。

步骤 1: 拉取最新代码 (Pull)

最佳习惯：每天开工前（即修改任何代码之前），先执行此步骤同步远程进度。

```
git pull origin main
```

- 💡 **为什么会冲突?**
  - 如果远程仓库有更新，而您本地也有修改（未提交），Git 为了保护您的修改，可能会**拒绝拉取**。
  - 如果远程和您本地都修改了同一个文件的同一行，拉取时就会触发**冲突 (Conflict)**。
- 🛡️ **如何避免?**
  - 请确保在 `git pull` 之前，工作区是干净的（使用 `git status` 检查）。
  - 如果本地有未写完的代码，请先使用 `git stash` 暂存（详见下文“场景 4”）。
  - 如果遇到冲突，请参考下文“场景 6”进行手动解决。

步骤 2: 查看状态 (Status)

随时检查当前文件状态。

```
git status
```

- 🔴 **红色**: 修改了但未暂存 (Unstaged)。
- 🟢 **绿色**: 已暂存，准备提交 (Staged)。

步骤 3: 暂存更改 (Add)

将工作区的修改添加到暂存区。

```
# 添加指定文件
git add src/okx_deepseek.py

# 添加所有修改文件 (不包括被 .gitignore 忽略的文件)
git add .
```

步骤 4: 提交更改 (Commit)

将暂存区的内容生成一个版本快照。

```
# -m 后面是提交说明，请务必清晰描述
git commit -m "feat: 增加动态止盈功能，优化风控逻辑"
```

💡 **提交规范推荐**:

- `feat`: 新功能 (Feature)
- `fix`: 修补 Bug
- `docs`: 仅修改文档
- `style`: 格式调整 (不影响逻辑)
- `refactor`: 代码重构 (无新功能或 Bug 修复)

步骤 5: 推送至远程 (Push)

```
git push origin main
```

---

3. 分支管理策略 (Branching)

不要直接在 `main` 分支上开发新功能，应该使用特性分支。

3.1 创建并切换分支

```
# 创建名为 feature/v2.4 的分支并切换过去
git checkout -b feature/v2.4

# (新版 Git 命令)
git switch -c feature/v2.4
```

3.2 在分支上工作

在分支上进行正常的 add, commit 操作。

3.3 合并分支 (Merge)

开发完成后，将代码合并回主分支。

```
# 1. 先切回主分支
git checkout main

# 2. 拉取主分支最新代码 (防止别人更新了)
git pull origin main

# 3. 合并分支
git merge feature/v2.4

# 4. 推送合并后的主分支
git push origin main
```

3.4 删除分支

```
# 删除本地已合并的分支
git branch -d feature/v2.4
```

---

4. 常见问题与救火指南 (Troubleshooting)

🚑 场景 1: 我改乱了文件，想放弃本地修改

尚未 `git add`，想把文件恢复到上次提交的状态。

```
# 恢复指定文件
git checkout -- src/okx_deepseek.py

# (新版 Git 命令)
git restore src/okx_deepseek.py
```

警告：此操作不可逆！

场景 2: 我已经 `git add` 了，想撤回暂存

不想提交了，但保留文件修改内容。

```
git reset HEAD src/okx_deepseek.py

# (新版 Git 命令)
git restore --staged src/okx_deepseek.py
```

场景 3: 提交信息写错了，或者少提了一个文件

不需要创建新的 commit，直接修正上一次提交。

```
# 1. (可选) 如果漏了文件，先 git add 补上
git add missed_file.py

# 2. 修改最后一次提交
git commit --amend -m "fix: 修正后的提交说明"
```

场景 4: 正在开发功能 A，突然要修 Bug B (暂存现场)

不想提交半成品的代码，但需要切换分支。

```
# 1. 将当前工作现场“储藏”起来
git stash save "正在开发功能A，临时挂起"

# 2. 此时工作区变干净了，可以去修 Bug...
# ... 修完 Bug 提交后 ...

# 3. 恢复之前的现场
git stash pop
```

场景 5: 不小心把 `config.json` 提交到仓库了！

千万不要直接 Push！

```
# 1. 从暂存区移除文件 (但保留本地文件)
git rm --cached src/config.json

# 2. 提交这个移除操作
git commit -m "fix: stop tracking sensitive config file"

# 3. 确保 .gitignore 里已经加了 config.json
```

如果已经 *Push* 到了远程，历史记录里会包含密钥，建议立即**重置交易所 *API Key***。

场景 6: 发生代码冲突 (Merge Conflict)

当 `git pull` 或 `git merge` 提示 Conflict 时：

1. **打开冲突文件**：你会看到类似 `<<<<<<< HEAD` 的标记。
2. **手动修改**：保留需要的代码，删除 `<<<<<<<`, `=======`, `>>>>>>>` 标记。
3. **重新提交**：

```
git add .
git commit -m "fix: 解决代码合并冲突"
```

---

5. 进阶风险防范 (Advanced Risks)

除了上述常见问题，以下操作可能导致严重后果，请务必警惕：

风险 1: 暴力强推 (`git push -f`)

- **后果**: 会直接覆盖远程仓库的历史记录。如果其他人也在此分支上工作，他们的代码将会**永久丢失**。
- **策略**:
    - **绝对禁止**在 `main` 或公共分支上使用 `-f` 或 `--force`。
    - 仅在自己私有的 `feature` 分支上（且确定只有你一个人用时）才可谨慎使用。

风险 2: 换行符地狱 (CRLF vs LF)

- **现象**: Windows 使用 `CRLF` 换行，Linux/Mac 使用 `LF`。跨系统协作时，Git 可能会认为“整个文件每一行都变了”，导致 Diff 混乱。
- **策略**: Windows 用户请务必执行以下配置（自动转换）：

```
git config --global core.autocrlf true
```

(Mac/Linux 用户设为 input)

风险 3: 大文件阻塞

- **现象**: 不小心提交了 `100MB+` 的大文件（如模型权重、数据库备份）。GitHub 会拒绝推送，且一旦 Commit 进本地历史，很难清理。
- **策略**:
    - 养成习惯：**先写 `.gitignore` 再写代码。**
    - 如果必须管理大文件，请安装使用 `git-lfs`。

风险 4: 游离指针 (Detached HEAD)

- **现象**: 当你执行 `git checkout <commit_id>` 查看历史时，处于“游离状态”。此时修改代码并提交，这些提交**不属于任何分支。**一旦切走分支，这些修改就会**丢失。**
- **策略**:
    - 如果你需要在历史版本上修改代码，**必须**创建一个新分支：

```
git checkout -b fix/old-version-patch
```

6. 常用命令速查表 (Cheatsheet)

场景	命令
查看	
查看状态	`git status`
查看历史提交	`git log --oneline --graph`
查看具体改动	`git diff <filename>`
撤销	
丢弃工作区修改	`git restore <filename>`
丢弃暂存区修改	`git restore --staged <filename>`
回退到上个版本	`git reset --hard HEAD^` (慎用)
分支	
查看分支	`git branch -a`
切换分支	`git switch <branch_name>`
远程	
强制拉取覆盖本地	`git fetch --all && git reset --hard origin/main`