Git 全面指南：常用命令与易错点

目录

1. 仓库操作
2. 提交管理
3. 分支与合并
4. 远程协作
5. 撤销与恢复
6. 查看与对比
7. 标签管理
8. 储藏与清理
9. 高级操作
10. 配置与帮助
11. 最佳实践与易错总结

1. 仓库操作

基本命令

```
# 初始化新仓库
git init

# 克隆远程仓库
git clone <URL>                # 克隆默认分支
git clone -b <分支名> <URL>     # 克隆指定分支

# 查看远程仓库信息
git remote -v                  # 显示所有远程仓库
git remote show origin         # 查看远程分支详细信息
```

易错点

- **克隆时协议错误**：
  - 使用SSH需提前配置公钥，否则会提示权限被拒绝（`Permission denied`）。
  - 使用HTTPS需注意网络代理或凭据缓存问题。
- **重复初始化**：在已有`.git`目录的文件夹内再次运行`git init`会覆盖配置，可能导致历史记录混乱。

2. 提交管理

基本命令

```
# 添加文件到暂存区
git add <file>          # 添加单个文件
git add .               # 添加所有修改（包括删除）
git add -u              # 添加已跟踪文件的修改（不包括新增文件）

# 提交更改
git commit -m "message"          # 提交暂存区内容
git commit -am "message"         # 添加所有已跟踪文件的修改并提交（跳过`git add`）

# 修改最后一次提交
git commit --amend               # 修改提交信息或追加文件
git commit --amend --no-edit     # 追加文件且不修改提交信息
```

易错点

- `git add .` **的陷阱**:
  - 会添加所有未忽略的文件，可能意外提交临时文件（如`.log`）。建议配合`.gitignore`使用。
- `git commit -am` **的局限性**:
  - 仅对已跟踪（tracked）文件生效，新增文件仍需手动`git add`。
- `--amend`**的风险**:
  - 修改已推送的提交会导致本地与远程历史不一致，需用`git push --force`覆盖（慎用！）。

---

3. 分支与合并

基本命令

```
# 分支操作
git branch                      # 查看本地分支
git branch -a                   # 查看所有分支（含远程）
git switch <branch>             # 切换到已有分支
git switch -c <new-branch>      # 创建并切换分支
git branch -d <branch>          # 删除分支（安全模式）
git branch -D <branch>          # 强制删除未合并的分支

# 合并与变基
git merge <branch>              # 合并指定分支到当前分支
git rebase <branch>             # 将当前分支变基到目标分支
git rebase --abort              # 终止变基并恢复到之前状态
git rebase --continue           # 解决冲突后继续变基
```

易错点

- **合并冲突未解决**:
  - 冲突后必须手动修改文件→`git add`→`git commit`才能完成合并。
- **误用**`rebase`**导致历史混乱**:
  - 不要对已推送到远程的分支执行`rebase`，否则需强制推送（破坏他人代码历史）。
- **删除未合并分支**:

- `git branch -D`会强制删除分支，可能导致代码丢失。建议先合并或创建备份标签。

---

4. 远程协作

基本命令

```
# 关联远程仓库
git remote add <name> <URL>     # 添加远程仓库
git remote remove <name>        # 删除远程仓库

# 推送与拉取
git push <remote> <branch>      # 推送分支到远程
git push -u <remote> <branch>   # 设置上游分支（后续可简写`git push`）
git push --force                # 强制覆盖远程分支（慎用！）
git fetch <remote>              # 拉取远程更新但不合并
git pull                        # 相当于`git fetch` + `git merge`
git pull --rebase               # 用变基代替合并（保持线性历史）
```

易错点

- **强制推送（`--force`）的风险**：
  - 覆盖远程分支可能导致团队其他人的代码丢失。推荐使用`--force-with-lease`（检查是否有人先推送了代码）。
- **`git pull`的合并冲突**：
  - 若本地有未提交的修改，直接`git pull`可能失败。建议先`git stash`保存修改。
- **忽略远程分支更新**：
  - 长期不执行`git fetch`可能导致本地分支与远程严重偏离，合并时出现大量冲突。

---

5. 撤销与恢复

基本命令

```
# 撤销工作区修改
git restore <file>              # 丢弃未暂存的修改
git restore --staged <file>     # 将文件移出暂存区（保留修改）

# 回退提交
git reset --soft HEAD~1         # 回退提交但保留修改（可重新提交）
git reset --hard HEAD~1         # 彻底丢弃最近一次提交的修改
git revert <commit-hash>        # 创建一个新提交来撤销指定提交

# 恢复误删分支
git reflog                      # 查看操作历史（含已删除的提交）
git checkout -b <branch> <hash> # 根据reflog中的哈希值恢复分支
```

易错点

- `git reset --hard` **的破坏性**：
  - 会永久丢弃未提交的修改，无法恢复！建议先`git stash`保存工作进度。
- **误用**`revert`**与**`reset`：
  - `reset`修改历史，适用于未推送的提交；`revert`生成新提交，更适合已推送的代码。
- **依赖**`reflog`**的局限性**：
  - `reflog`默认保留30天，过期后无法恢复被删除的分支或提交。

（由于篇幅限制，以下为部分内容预览，完整版需展开）

6. 查看与对比

```
git status                  # 查看工作区状态
git log --oneline --graph   # 简洁历史与分支图
git diff                    # 对比工作区与暂存区
git diff --staged           # 对比暂存区与最新提交
```

7. 标签管理

```
git tag -a v1.0 -m "Release"    # 创建附注标签
git push origin --tags          # 推送所有标签
```

8. 储藏与清理

```
git stash               # 临时保存工作区
git stash pop           # 恢复最近一次储藏
git clean -df           # 删除未跟踪的文件/目录（慎用！）
```

---

11. 最佳实践与易错总结

最佳实践

1. **小步提交**：每个提交只解决一个问题，便于回退和代码审查。
2. **频繁拉取远程变更**：避免本地分支与远程偏离过大。
3. **善用分支**：新功能开发使用独立分支，避免直接修改`main`分支。
4. **规范提交信息**：使用Conventional Commits等约定式提交格式。

高频易错场景

场景	错误操作	正确操作
提交后发现漏文件	直接新增提交	`git commit --amend`
误删未合并分支	`git branch -D`	从`reflog`中恢复

场景	错误操作	正确操作
强制推送导致团队代码丢失	`git push --force`	`git push --force-with-lease`
忽略`.gitignore`	提交临时文件	提前配置`.gitignore`规则

提示：建议将本文保存为`Git_CheatSheet.md`，方便随时查阅！