git tutorial

举例：

- 主仓：
  - 主分支
  - 分支 1
  - 分支 2
- 子仓：

基本操作

创建新的存储仓

echo "# git_tutorial" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/huzibo-yyds/git_tutorial.git
git push -u origin main

从命令行推送现有存储仓

git remote add origin https://github.com/huzibo-yyds/git_tutorial.git
git branch -M main
git push -u origin main

一般开发流程

原则：早拉取、原子化修改

- git pull
- 修改代码
- git pull
  - 无冲突
- git add、commit、push

代码冲突如何处理：

push 前没有 pull

关键词 *rejected*

```
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/huzibo-yyds/git_tutorial_submodule.git'
```

- git pull
  - CONFLICT (content): Merge *conflict* in README.md
- 手动解决冲突
- 重新提交

提交与撤销 commit、reset

可以执行多次 commit，只要不进行 push，都只存在本地
📍 原则，频繁 commit，谨慎push

修改提交文字
git commit --amend -m "新的更规范的提交说明"

reset

- 撤销 commit ，但保留更改
  - git reset --soft HEAD~1
  - HEDA~1 表示撤销最后 1 次，HEDA~n，表示撤销最后 n 次
- 撤销 commit，且不保留修改 慎用
  - git reset --hard HEAD~1

push 后提交

1. 安全撤销，不删除历史记录 显示为 “Revert xxx”

```shell
# 1. 找到你想要撤销的那次提交的 ID
git log --oneline

# 2. 撤销该提交（Git 会自动创建一个新的提交来抵消它）
git revert <commit-id>

# 3. 将这个“抵消”提交推送上去
git push origin <分支名>
```

2. 彻底撤销，抹除提交记录（仅限个人分支

```shell
# 1. 本地回退到指定版本（--hard 会删除代码修改，--soft 会保留修改）
git reset --hard HEAD~1

# 2. 强制推送到远程（覆盖服务器记录）
git push -f origin <分支名>
```

分支相关

不影响稳定代码（主分支）的情况下，自由地开发新功能或修复 Bug

常用命令

- `git branch`: 查看本地所有分支。
- `git checkout -b <name>`: 创建并切换到新分支（现在推荐使用 `git switch -c <name>`）。
- `git branch -d <name>`: 删除分支（通常在合并后执行）。
- `git status`: 随时查看当前处于哪个分支，以及是否有未提交的修改。

创建自己分支并第一次提交

```shell
git checkout dev    # 切换到开发主分支
git pull origin dev # 拉取远程最新代码

# 创建并同时切换到新分支
git checkout -b feature/my-task
# 编写完代码后查看修改
git status
git add .  # 添加所有改动
git commit -m "feat: xx"
# 以后在该分支只需输入 git push，无需再写后面一长串
git push -u origin feature/my-task  # 🟥 只有加 -u 本地和远程才会联系，以后提交不用显示指定远程分支
```

自己分支合并到主分支

在本地分支 commit 修改
切回主分支
git checkout main
拉取主分支最新代码
git pull origin main
执行合并
git merge feat-branch
推送到远程服务器
git push origin main

本地分支拉取主分支最新修改

1. 确保分支干净
2. 方法1 ——直接在分支拉取，同时会合进当前分支
    1. git pull origin main
3. 方法2 ——通过本地主分支中转
    1. 不推荐
       git checkout main
       git pull origin main
       git checkout feat-login
       git merge main

在‘个人’分支开发提交前

git stash # 1. 隐藏 或者使用
git fetch origin dev # 2. 下载最新的 dev 代码（不切换分支）
git merge origin/dev # 3. 将下载的最新代码合并到当前分支
git stash pop # 4. 恢复
git stash drop # 4. 删除 （可选

子模块相关

添加子模块

`git submodule add <url> <子模块路径>`
git submodule add git@github.com:huzibo-yyds/git_tutorial_submodule.git ./lib

- 此处使用 ./lib 则项目根目录会到 lib中
    - 最好使用 ./lib/子模块名
      会增加.gitmodules文件，包含 path 与 url

克隆带有子模块的仓库

- 方法 1
  - git clone --recursive <主仓库地址>
- 方法 2，已经 clone 了主仓库，未关注 submodel
  - git submodule update --init --recursive
  - 为什么要初始化子模块
    - 默认情况只是一个子模块指针，子模块中内容是不会下载下来的

同步更新子模块

- 方法 1 （ 进入子模块根目录更新）
  进入子模块目录 cd libs/shared
  git pull origin main cd .
  ./..
  主仓库会发现指针变了，需要提交这个变动
  git add libs/shared
  git commit -m "更新子模块到最新版本"
- 方法 2（一键更新子模块💡
  - git submodule update --remote --merge

修改和推送子模块的更改

❗ 需要先更新子仓库，再更新主仓库
方法 1
进入子模块：cd libs/shared
提交修改：git add . → git commit -m "fix something"
推送子模块：git push origin main（必做！ 否则别人拉不到你的代码）
返回主仓库提交指针：cd ../.. → git add libs/shared → git commit -m "更新子模块指针"

方法 2（一键全推
git push --recurse-submodules=on-demand

设置子模块自动更新

📍 **保持主项目中子模块版本一致**
`submodule.recurse`
（本质时，在执行 git pull 时，默认带上参数 --recurse-submodules

- 全局配置
  - git config --global submodule.recurse true
- 仅为当前项目配置
  - git config submodule.recurse true
- 验证配置
  - git config --list | grep submodule.recurse
- 效果
  - git pull，自动同步子模块（当主模块中子模块指针更新时自动更新
  - git checkout，自动将子模块切换到，分支中子模块所指向的版本
  - git merge，自动根据子模块指针，合并子模块

🟥 误解为自动更新子模块代码到最新，下面是正确理解

- 保持子模块与父项目，所记录版本一致
- 只有当主项目拉取到一个新提交，且新提交中包含子模块指针更新时，才会更新子模块

遇到问题与解决

.diff 文件是什么？需要提交么？

`lib.diff` **并不是一个真实存在于你硬盘上的文件**，它是 VS Code 的 Git 插件为了向你展示“子模块发生了什么变化”而临时生成的一个**虚拟视图**。

不要提交
正确做法

- 更新提交子模块中，在主模块中提交更新
- 如果子模块修改是误操作，则撤销子模块更新
  - cd lib
  - git restore .

当前代码有问题，需要切换会历史版本build 一版可执行文件

```shell
# 暂存当前修改
git stash  # 暂时藏起来
# 找到历史版本
git log --oneline -n 20
# 基于目标 ID 创建一个名为 build-fix 的新分支并切换过去
git checkout -b build-fix a1b2c3d
( 直接git checkout <id>

# 同步子模块（可选
git submodule update --init --recursive

# 执行构建
xxx

# 切换回开发分支
git checkout dev    # 回到开发分支
git branch -D build-fix # 删除临时构建分支
git stash pop       # 恢复你之前写了一半的代码
```

git config 说明

TODO

小技巧

简化终端显示

比如

- 修改前：huzi@huMacBook-Air git_tutorial %
- 修改后：git_tutorial %

```shell
# 通过 PS1 环境变量自定义终端提示符
echo 'export PS1="%1~ %# "' >> ~/.zshrc
source ~/.zshrc
```