# 1. 配置相关

git config --global user.name "Your Name" # 配置用户名

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

git config --global core.editor "vim" # 设置默认编辑器

git config --list # 查看当前的配置

# 2. 初始化与克隆仓库

git init # 初始化一个新的 Git 仓库

git clone <url> # 克隆远程仓库到本地

# 3. 状态查看

git status # 查看当前文件的状态

git log # 查看提交日志

git log --oneline # 简洁版日志输出

git diff # 比较工作目录和暂存区的差异

git diff --staged # 比较暂存区和最后一次提交的差异

# 4. 分支管理

git branch # 列出所有分支

git branch <branch-name> # 创建新分支

git checkout <branch-name> # 切换到某个分支

git checkout -b <branch-name> # 创建并切换到新分支

git merge <branch-name> # 合并指定分支到当前分支

git branch -d <branch-name> # 删除本地分支

git branch -D <branch-name> # 强制删除本地分支

# 5. 提交更改

git add <file> # 将文件添加到暂存区

git add . # 将所有更改添加到暂存区

git commit -m "commit message" # 提交暂存区到本地仓库

git commit --amend # 修改最近一次提交

# 6. 远程仓库操作

git remote -v # 查看当前远程仓库地址

git remote add origin <url> # 添加远程仓库

git fetch # 从远程仓库获取更新

git pull # 拉取远程仓库的最新更新并合并

git push # 推送本地分支到远程仓库

git push origin <branch-name> # 推送指定分支到远程仓库

# 7. 撤销操作

git checkout -- <file> # 撤销对文件的修改（未暂存）

git reset <file> # 取消暂存区的文件

git reset --hard # 回滚到最后一次提交，并删除所有未提交的更改

git revert <commit-hash> # 撤销指定的提交

# 8. 标签管理

git tag # 列出所有标签

git tag <tag-name> # 创建一个新标签

git tag -a <tag-name> -m "message" # 创建带注释的标签

git push origin <tag-name> # 推送标签到远程仓库

# 9. 比较与检查

git show <commit-hash> # 显示某次提交的详细信息

git blame <file> # 显示每行代码的修改记录

git log -p # 查看每次提交的差异

# 10. 其他实用命令

git stash # 暂存当前未提交的工作区更改

git stash pop # 恢复暂存的更改

git clean -f # 强制删除未跟踪的文件

git reflog # 查看所有分支的移动记录

在 PyCharm 中使用 Git 命令非常方便，因为它内置了 Git 集成。你可以通过菜单操作、快捷键或命令行实现大多数 Git 操作。以下是几个常用 Git 命令在 PyCharm 中的操作方式举例：

### 1. 配置相关（`git config`）

PyCharm 通常会自动识别你的 Git 配置，例如用户名和邮箱。你可以通过以下步骤查看或修改配置：

- \*\*菜单路径\*\*：`File > Settings > Version Control > Git`

- 在这里，你可以设置 Git 的路径，并点击 `Test` 来验证。

### 2. 初始化与克隆仓库

- \*\*初始化仓库（`git init`）\*\*：

- 右键点击项目目录，然后选择 `Git > Enable Version Control Integration`，选择 `Git`，PyCharm 会自动为项目初始化 Git 仓库。

- \*\*克隆远程仓库（`git clone`）\*\*：

- 在 PyCharm 启动界面，选择 `Get from Version Control`，输入仓库的 URL 和本地保存路径，点击 `Clone` 即可。

### 3. 状态查看（`git status`、`git log`、`git diff`）

- \*\*查看状态（`git status`）\*\*：

- 右键点击项目目录，选择 `Git > Show Git Log`，可以看到文件的状态变化。

- 未追踪的文件、修改的文件会直接在项目文件树中显示不同颜色标识。

- \*\*查看提交历史（`git log`）\*\*：

- 选择 `Git > Show Git History`，可以看到项目的提交历史。

- \*\*查看差异（`git diff`）\*\*：

- 右键点击某个文件或某个文件夹，选择 `Git > Compare with...`，选择历史版本进行对比。

- 或者直接在代码编辑器的左侧点击 `View Difference` 图标。

### 4. 分支管理（`git branch`、`git checkout`、`git merge`）

- \*\*创建新分支（`git branch <branch-name>`）\*\*：

- 选择 `Git > Branches`，点击 `New Branch`，输入分支名称即可创建新分支。

- \*\*切换分支（`git checkout <branch-name>`）\*\*：

- 同样在 `Git > Branches` 中，点击你想切换的分支。

- \*\*合并分支（`git merge <branch-name>`）\*\*：

- 选择 `Git > Branches`，在当前分支下点击 `Merge into Current`，选择你要合并的分支。

### 5. 提交更改（`git add`、`git commit`）

- \*\*添加文件到暂存区（`git add`）\*\*：

- 右键点击文件或文件夹，选择 `Git > Add`，文件会被添加到暂存区。

- \*\*提交更改（`git commit`）\*\*：

- 选择 `Git > Commit...` 或使用快捷键 `Ctrl+K`，在弹出的窗口中输入提交信息并选择要提交的文件，点击 `Commit`。

### 6. 远程仓库操作（`git pull`、`git push`、`git fetch`）

- \*\*拉取远程更新（`git pull`）\*\*：

- 选择 `Git > Pull...` 或点击工具栏的 `Pull` 按钮，拉取远程仓库的最新代码。

- \*\*推送到远程仓库（`git push`）\*\*：

- 选择 `Git > Push...` 或点击工具栏的 `Push` 按钮，将本地更改推送到远程仓库。

- \*\*获取远程更新（`git fetch`）\*\*：

- 选择 `Git > Fetch`，这只会下载远程更新，而不会进行合并。

### 7. 撤销操作（`git reset`、`git revert`）

- \*\*撤销文件修改（`git checkout -- <file>`）\*\*：

- 右键点击文件，选择 `Git > Revert...`，可以撤销该文件的修改。

- \*\*重置暂存区（`git reset`）\*\*：

- 选择 `Git > Reset Head...`，可以重置暂存区的文件。

- \*\*回滚到特定版本（`git revert`）\*\*：

- 在 `Git > Show Git History` 中，右键点击某个历史提交，选择 `Revert Commit`。

### 8. 标签管理（`git tag`）

- \*\*创建标签（`git tag <tag-name>`）\*\*：

- 选择 `Git > Branches`，点击 `Tags`，选择 `New Tag`，输入标签名称即可创建。

- \*\*推送标签到远程仓库（`git push origin <tag-name>`）\*\*：

- 选择 `Git > Push...`，在弹出的窗口中选择标签并推送到远程仓库。

### 9. 比较与检查（`git show`、`git blame`）

- \*\*查看提交详情（`git show`）\*\*：

- 在 `Git > Show Git Log` 中，点击某个提交，右侧会显示该提交的详细信息。

- \*\*查看文件修改记录（`git blame`）\*\*：

- 右键点击某个文件，选择 `Git > Annotate`，可以看到每行代码的提交历史及提交者。

### 10. 其他实用命令（`git stash`、`git reflog`）

- \*\*暂存未提交的更改（`git stash`）\*\*：

- 选择 `Git > Stash Changes...`，输入备注即可

在 Git 仓库的 `main` 目录下创建一个新的文件夹 `win`。

mkdir win

移动

mv file1.py file2.py file3.py win/

git add win/

git commit -m "Move three Python files to the win folder"

git push origin main

rm .\Self\_study\_python\main.py