解决 Git 冲突

为了设置 Git 使其能正确合并，我们先开始做一些简单的配置。

# 通用设置建议

当遇到冲突时，可以在命令行或者其他可视化工具中输入"git mergetool"来初始化合并活动。在".gitconfig"中用"merge.tool"变量来设置 Git 中自己喜欢的冲突解决软件，比如用 KDiff3 的可能会这样填写".gitconfig" 的 merge 部分：等价于shell命令，git config --global merge.tool kdiff3

|  |
| --- |
| [merge]  tool="kdiff3" |

在冲突标记中显示（分支）共同的祖先：用下面的设置来改进冲突标记使其也显示（分支）共同祖先（感谢罗宾·斯托克和休·吉登斯）：

|  |
| --- |
| git config --global merge.conflictstyle diff3 |

这个设置命令新添加一部分标记||||||| 从而给冲突加了注释，这样可以看到冲突行在有问题的两个分支的共同祖先处是什么状态。

合并时使用"耐心"算法：如果文件内容很长（比如一个 XML文件）、冲突很多或者两个版本很不一致时，试着用下面的命令再次合并：

|  |
| --- |
| git merge --strategy-option=patience |

“耐心”算法的结果应该可以更好地协调一些函数中或者标记中没有配对的括号，具体算法细节可以参考 Stack Overflow 上的一个回答。

当你需要单个文件的历史信息时：

|  |
| --- |
| git log --merge --decorate --source -p path/to/file/you/care/about |

# 手动解决冲突

从手头的合并开始，看下有多少文件冲突

|  |
| --- |
| git merge the/other/branch  git status |

对每个冲突文件：在编辑器中打开文件（比如 IntelliJ 或 Vim ），看看每个被冲突标记（“>>>>”和“<<<<”）围绕的区块。看看（被冲突标记的区块）是否有意义，每个作者的意图是什么，如果能弄清楚就解决掉。如果冲突标记无法理解，通常是这些文件改动很大，运行下面的命令：

|  |
| --- |
| git diff HEAD...the/other/branch -- path/to/conflicting/file  git diff the/other/branch...HEAD -- path/to/conflicting/file  这样做是为了看哪边改动较小 |

通常下面的命令，能帮助理解另一边改动了什么。

git log -p HEAD..the/other/branch -- path/to/conflicting/file

git log -p the/other/branch..HEAD -- path/to/conflicting/file

回溯文件到改动最大的一边：

git checkout the/other/branch -- path/to/conflicting/file

手动检查并且再重新应用从另一边对文件的更改:

git add path/to/conflicting/file

当这些更改都修复之后要构建整个项目，确保至少可以编译通过，如果测试可以很快运行起来，也要运行一下这些测试：git commit

# 合并工具kdiff3的安装和使用

安装Kdiff3 软件。（最好使用默认路径）

添加kdiff3到Git mergetool里。 git config --global merge.tool kdiff3

添加kdiff3路径到 git global config里。git config --global mergetool.kdiff3.path "C:\Program Files\KDiff3\kdiff3.exe"

以后merge发生冲突时：git mergetool 来做图形化merge。

git config --global mergetool.keepBackup false