深 圳 大 学 实 验 报 告

课程名称 软件工程

项目名称 Git 分支及标签管理

学　　院 计算机与软件学院

专　　业 计算机科学与技术

指导教师 李俊杰

报 告 人 吴嘉楷 学号 2022150168

实验时间 2024 年 11 月 21 日

提交时间 2024 年 11 月 29 日

教务处制

一、实验目的与要求

（1）了解分支和标签概念
（2）掌握如何创建与合并分支
（3）掌握如何解决分支
（4）掌握多人协作进行软件开发的方法
（5）掌握标签的创建和相关操作

二、实验内容与方法

请同学们认真学习 Git 在线课程教程。
网址如下：
https://www.liaoxuefeng.com/wiki/896043488029600

（1）在本地创建一个“深圳大学本科生参与竞赛交流平台”项目，将所有的项目源程序文件提交到本地库中。
（2）为某个文件创建分支，在分支中提交和对比文件
（3）将修改的内容合并到主分支
（4）与项目组同学合作，各自将本地代码提到同一个远程版本库
（5）与项目组同学共同编写同一个代码文件，体验冲突解决。
（6）为项目增加本地和远程标签，并查看、删除标签

要求提交图文并茂的实验报告，认真实践项目的所有过程。

三、实验步骤与过程

1. 创建一个关于“深圳大学本科生参与竞赛交流平台”项目的 git 本地仓库

由于我们的项目是基于 github 上某个开源的社区交流平台进行开发的，于是，我们不再需要在执行“git init”命令去初始化 git 仓库，而直接使用“git clone”命令，将代码克隆到本地新建的目录文件夹即可。

（1）在本地创建项目的一个新目录文件夹，命名为“CompetitionPlatform”

图 1.1 创建项目文件夹

（2）进入该文件夹路径的 cmd 命令行，执行“git clone”命令，将开源代码克隆到本地

```
C:\Users\86178\Desktop\CompetitionPlatform>git clone https://github.com/kaikaiGit/Competition-Platform.git
Cloning into 'Competition-Platform'...
remote: Enumerating objects: 37440, done.
remote: Counting objects: 100% (87/87), done.
remote: Compressing objects: 100% (68/68), done.
remote: Total 37440 (delta 37), reused 44 (delta 14), pack-reused 37353 (from 1)
Receiving objects: 100% (37440/37440), 223.77 MiB | 2.33 MiB/s, done.

Resolving deltas: 100% (8813/8813), done.
Updating files: 100% (4317/4317), done.
```

图 1.2 克隆开源代码到本地

（3）查看克隆结果

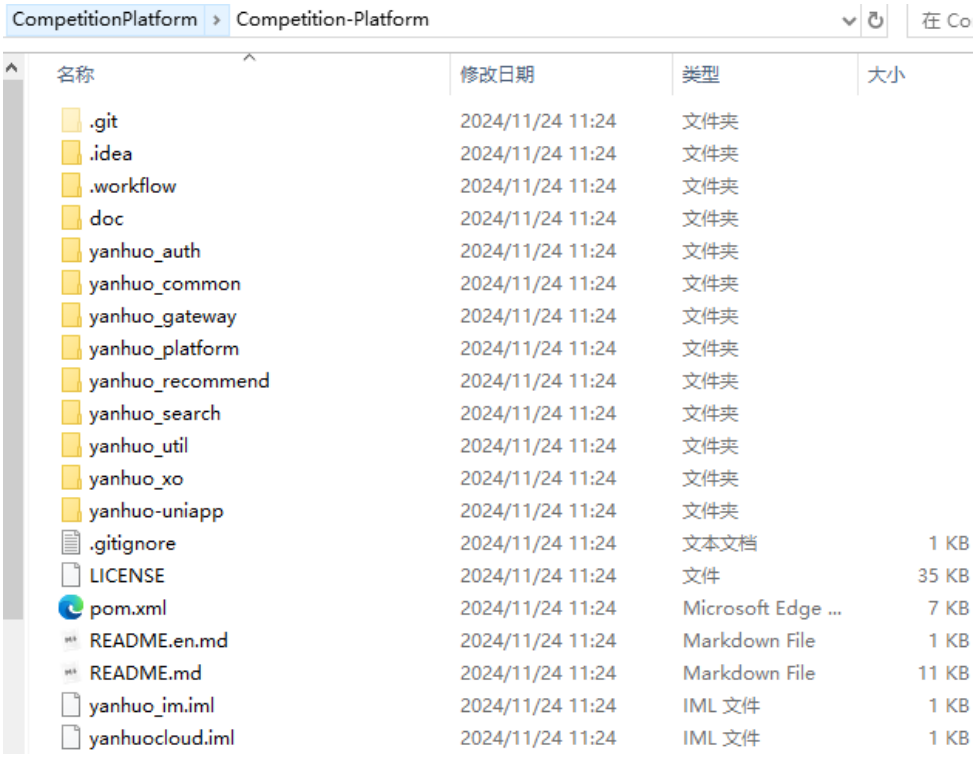

图 1.3 执行克隆命令后的文件夹内容

由上图可见，克隆完成后，原本的空目录“CompetitionPlatform”中多出了一个包含了源代码文件的新文件夹“Competition-Platform”，新文件夹的名字对应远程仓库的仓库名，文件夹的代码文件对应远程仓库的开源代码。

2. 在本地创建新分支，分支名为“dev-web”

```
86178@kaikai-laptop MINGW64 ~/Desktop/CompetitionPlatform/Competition-Platform (dev-web)
$ git checkout -b dev-web
Switched to a new branch 'dev-web'
```

图 2 创建“dev-web”分支

3. 在 dev-web 分支上修改代码并提交，然后对比文件差异

（1）修改代码

```
<view class="container">
  <!-- 标题 -->
  <view class="title">大学生竞赛交流平台</view>
  <view class="info">
    <view class="center">
      <!-- 用户名输入框 -->
      <input placeholder="请输入手机号" class="info-input" placeholder-style="letter-spacing:2rpx" v-model="userInfo.username"/>
      <!-- 密码输入框 -->
      <input placeholder="请输入密码" class="info-input" placeholder-style="letter-spacing:2rpx" type="password" v-model="userInfo.password" />
    </view>

    <view class="login">
      <!-- 登录按钮 -->
      <button @click="login">登录</button>
    </view>
    <view class="info-bottom">
      <!-- 验证码登录按钮 -->
      <view @click="loginByCode">验证码登录</view>
      <view class="info_right">
        <!-- 忘记密码和注册按钮 -->
        <view @click="findPassword">忘记密码?</view>
        <view style="margin: 0 10rpx;">|</view>
        <view @click="regist">立即注册</view>
      </view>
    </view>
  </view>
```

图 3.1 调整登录页面内容

首先，修改页面标题为“大学生竞赛交流平台”。然后，为输入框的默认占位文字添加基础样式。此外，在“忘记密码”、“立即注册”两个选项之间加入一个分割线。

最后，调整一下文字的大小和颜色，部分 css 代码改动如下图所示：

```css
.title {
    text-align: center;
    padding-top: 40px;
    font-size: 60rpx;
}
```

```css
.other-login {
    display: flex;
    justify-content: space-between;
    padding: 0 20rpx;
    font-size: 24rpx;
}
```

```css
.info-bottom {
    margin-top: 20px;
    display: flex;
    justify-content: space-between;
    color: #000064;
    font-size: 24rpx;
}
```

图 3.1.2 调整登录页面 css 样式

（2）调整后的页面效果

图 3.2 登录页面效果

（3）使用 git 提交代码改动记录

首先，使用“git add .”命令，将文件的更改添加到 暂存区。它只是将你修改的文件或新增的文件标记为待提交状态，但并没有真正提交到 Git 仓库的历史记录中。

然后，输入“git commit -m "修复登录页面"”命令，将暂存区的内容提交到 Git 仓库的历史记录中，同时注释本次改动的内容为："修复登录页面"。

```
86178@kaikai-laptop MINGW64 ~/Desktop/CompetitionPlatform/Competition-Platform (dev-web)
$ git add .

86178@kaikai-laptop MINGW64 ~/Desktop/CompetitionPlatform/Competition-Platform (dev-web)
$ git commit -m "修复登录页面"
[dev-web c4a0a295] 修复登录页面
 2 files changed, 10 insertions(+), 13 deletions(-)
```

图 3.3 提交代码改动内容到 git 仓库

（4）查看 git 的历史记录

```
86178@kaikai-laptop MINGW64 ~/Desktop/CompetitionPlatform/Competition-Platform (dev-web)
$ git log --oneline
c4a0a295 (HEAD -> dev-web) 修复登录页面
05f32e3c (main) 准备创建新分支
```

图 3.4 查看历史提交记录

（5）对比主分支与开发分支差异（这里即改动前后的差异）

使用“git diff branch1 branch2”命令对比两个分支之间的差异，部分差异如下：

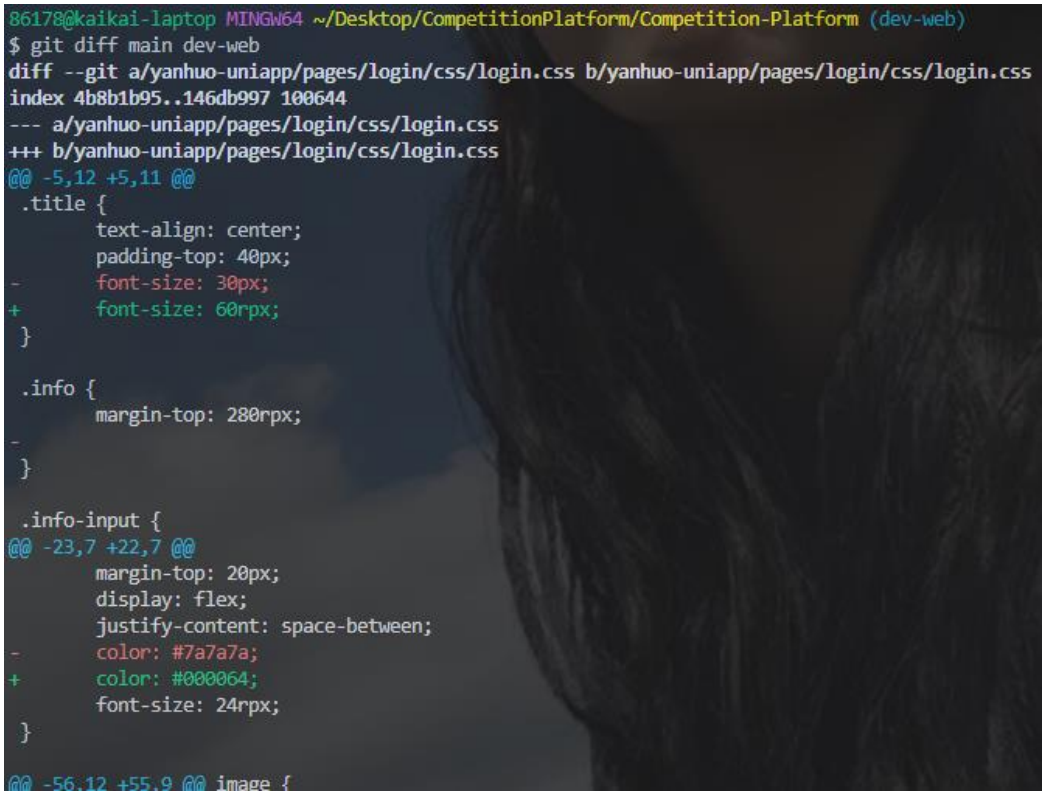

图 3.5 对比分支差异

使用“git graph”插件后，我们也可以从图形化界面中看到差异，如下图所示：

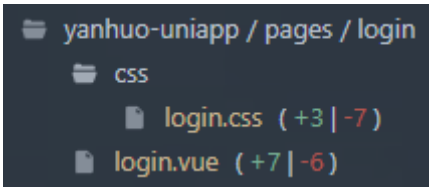

图 3.5.2 当前版本与上一个版本的改动差异

4. 将本地分支“dev-web”合并到本地的主分支

首先，使用“git checkout main”命令，切换到 main 分支上。然后，使用“git merge dev-web”命令，将“dev-web”分支合并进来，这样就完成了分支的合并。合并过程如下图所示：

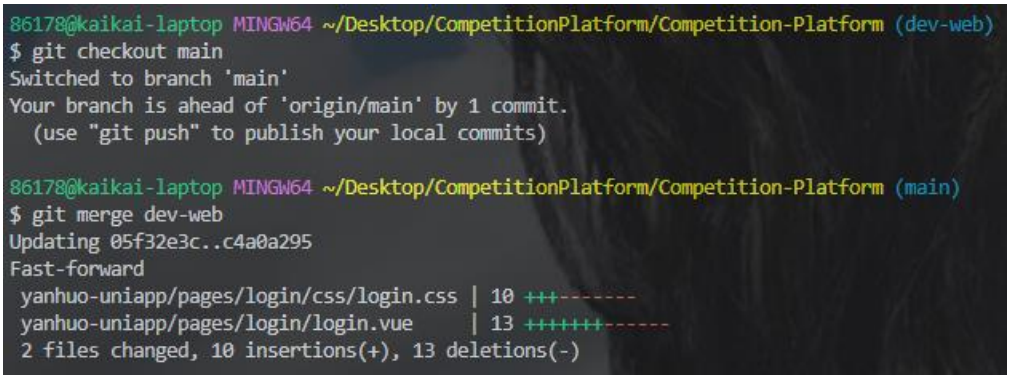

图 4 合并分支

5. 为项目增加本地标签

命令操作：

使用“git tag tag_name commit_hash”命令给指定的提交打标签。如下图所示：

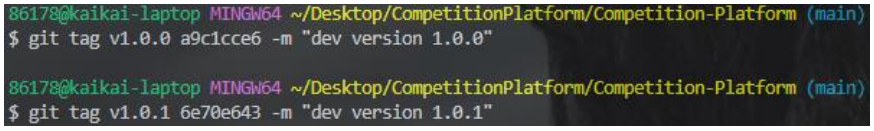

图 5.1 为两次指定的提交打标签

打标签结果：

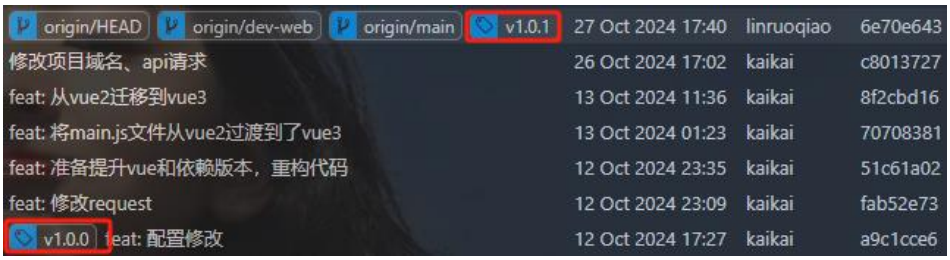

图 5.2 结果截图

6. 将本地标签推送至远程仓库（git push --tags）

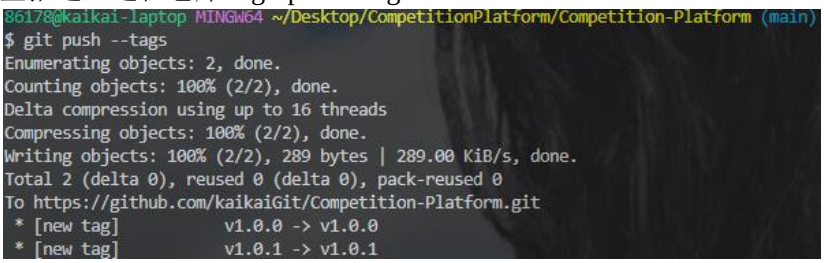

图 6 为远程仓库增加项目标签

7. 查看删除标签

查看所有标签：git tag

查找某些标签：git tag -l "v1.*"

删除本地标签：git tag -d tag_name

删除远程标签：git push origin --delete tag_name

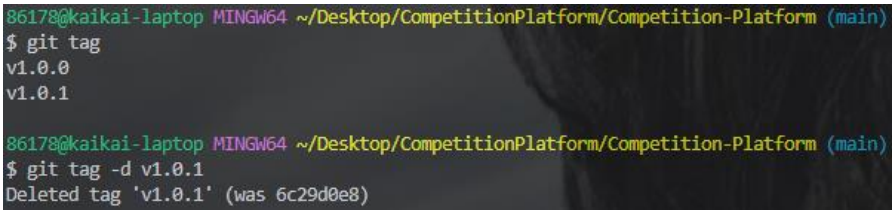

图 7 查找、删除本地标签

8. 体验冲突的产生和解决

Git 冲突的原因：

冲突通常是因为在合并分支或应用补丁时，Git 无法自动解决版本间的差异。这种情况常发生在多个开发者同时修改同一文件的同一区域时。例如，一个分支修改了一段代码为 "Hello World!"，另一个分支将其修改为 "Hello Git!"。当 Git 尝试合并时，不确定哪种改动是正确的，于是产生冲突。此外，冲突也可能因为文件被重命名、移动或删除而未被同步，

或者两方对同一个文件添加了不同的内容，而这些修改在逻辑上相互冲突。

冲突的产生：

因为在 login.vue 和 login.css 文件中，我和队友同时修改了这两个文件的相同位置，于是 git 不知道听谁的，产生了合并冲突。

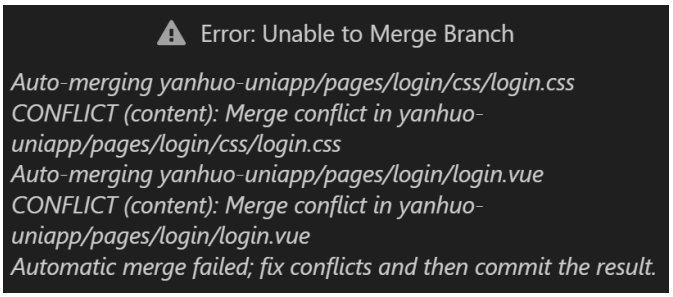

图 8 git 冲突产生的报错信息（git graphy 插件）

解决冲突：

以解决 login.vue 文件中的冲突为例，我们可以选择接受传入，或者接受当前分支的内容，手动选择我们要保留的分支的代码，从而解决冲突：

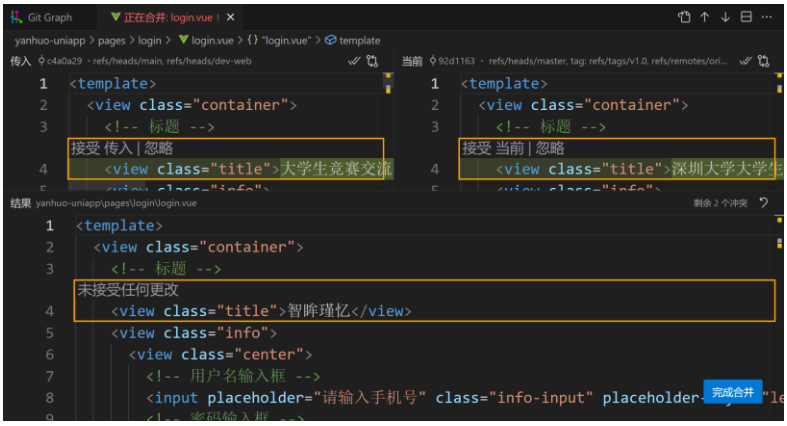

图 9 在 vscode 中解决 login.vue 的合并冲突

解决冲突后提交合并成功的代码：

图 10 提交合并后的结果

9. 将最新代码推送到远程仓库中

在 pull 完成后，使用“git push”命令，将本地最新代码推送到 github 远程仓库：

```
PS C:\Users\24312\Desktop\软件工程\实验五\code\Competition-Platform> git push -u origin master
Enumerating objects: 50, done.
Counting objects: 100% (42/42), done.
Delta compression using up to 16 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (20/20), 1.97 KiB | 672.00 KiB/s, done.
```

图 11 推送最新代码

四、实验结论或体会

实验总结：

在完成“深圳大学本科生参与竞赛交流平台”项目的 Git 分支及标签管理实验后，我深刻体会到了版本控制工具在软件开发中的重要性。通过本次实验，我不仅掌握了 Git 的基本操作，还深入理解了分支和标签的概念，以及它们在多人协作开发中的关键作用。Git 不仅能够有效地管理代码版本，还提供了分支和标签等高级功能，极大地提升了团队协作开发的效率。

实验过程中，我首先通过克隆开源代码到本地，创建了项目的新目录文件夹，并在其中初始化了 Git 仓库。随后，我在本地创建了名为“dev-web”的新分支，并在该分支上进行了代码的修改和提交。通过调整登录页面的内容和样式，我体验了分支工作的流程，并学习了如何使用 git add 和 git commit 命令将更改记录到 Git 仓库中。

此外，我还学习了如何使用 git diff 命令对比分支间的差异，并利用图形化工具 git graph 更直观地查看版本变化，我还掌握了如何为项目的重要里程碑打上标签，并通过 git push --tags 将这些标签推送到远程仓库。

实验难点：

1. 分支合并冲突：在合并分支时，我遇到了代码冲突的问题。这是由于我和队友同时对同一文件的相同区域进行了修改。

解决方案：

我采用了手动合并的方法，通过比较不同分支的代码，仔细比对两个版本的代码，决定哪些更改应该被保留。这个过程虽然耗时，但让我对代码合并有了更深入的理解。

这也让我明白了，在进行代码修改之前，我应该和队友进行充分的沟通，避免同时对同一文件进行修改，从而减少冲突的产生。

2. 多人协作中的代码同步：在多人协作开发中，如何保持代码的同步是一个挑战。

解决方案：

我们采用了定期 pull 和 push 的策略，确保每个人的工作都是基于最新的代码状态。此外，我们还使用了分支策略，将开发工作隔离在不同的分支中，直到功能完成并通过测试后才合并到主分支。

（实验报告的篇幅控制在 6-8 页）

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。