

附录三、Git以及LLM

本地运行大语言模型

1. 安装Ollama
2. 下载大语言模型：

SHELL

```
ollama pull gemma:2b qwen2:1.5b qwen2-math nomic-embed-text
```

3. 测试大语言模型：

SHELL

```
ollama run qwen2:1.5b  
# 天空为什么是蓝色的？
```

大语言模型配合Obsidian

1. 安装插件 **Copilot**，并设置大语言模型 **qwen2:1.5b** 以及嵌入模型 **nomic-embed-text**。注意，在 **general settings>Add Custom Model>Model Name (qwen2:1.5b)>Provider(ollama)>Add Model>Set Default to this new added model**，不要忘记了**保存并重载**。
2. 理论上，从开始菜单运行 **Ollama** 即可正常使用。如果不能，则可进行如下debug：
 - 使用管理员运行 **cmd**，然后运行 **ollama serve**
 - 测试链接：**curl http://localhost:11434/v1/models**，正常会列出所有安装的模式。

User: 为什么天空是蓝色的，夕阳的时候又是红色的？

Copilot: 天空之所以呈现蓝色，是因为太阳光中的蓝光被大气中的微小颗粒散射。这种现象被称为瑞利散射，它主要发生在地球大气层中，尤其是在对流层。

当阳光穿过大气层时，其中的波长较短的蓝光比波长较长的红光更容易被散射。这是因为蓝光的波长更短，所以它们在空气中传播得更快，从而更容易与空气分子碰撞并散射到各个方向上。而红光的波长较长，因此它们在大气中传播得较慢，不容易被散射。

夕阳时天空呈现红色的原因是因为太阳光线穿过大气层中的臭氧层。当阳光照射到地球表面时，一部分紫外线会被大气中的氧气和氮气吸收，但其中的一部分紫外线仍然能够穿透大气到达地面。这些未被吸收的紫外线与大气中的水分子、二氧化碳等物质相互作用，产生化学反应，释放出能量，使得天空呈现出红色或橙色。

此外，夕阳时太阳光线的角度较低，照射到地球表面的部分光线需要穿过更多的大气层，这也会导致阳光中波长较长的蓝光散射得更多，而红光则更容易穿透大气到达地面。因此，在夕阳时分，天空呈现红色和橙色的现象更为普遍。

Git版本管理

安装Git

1. 下载Git: [Git - Downloads \(git-scm.com\)](https://git-scm.com/downloads).
2. 安装, 并添加路径到环境变量

使用版本控制系统的优点

1. **记录更改历史**: 版本控制可以详细记录每次对笔记的修改, 包括添加、删除和编辑内容。这有助于跟踪思路的发展, 并可以在需要时查看或恢复到先前的版本。
2. **协作**: 如果你与他人合作编写或修改笔记, 版本控制允许你和你的合作者并行工作, 避免冲突, 并能够合并不同人的贡献。
3. **备份和恢复**: 所有的更改都被记录和保存, 这样即使你的计算机出现故障, 你也可以从版本控制系统中恢复丢失的内容。
4. **版本对比**: 版本控制系统可以让你方便地对比不同版本的笔记, 查看具体的更改内容, 并理解这些更改对整个笔记的影响。
5. **实验和分支**: 你可以在一个独立的分支上进行实验, 尝试新的思路或格式, 而不影响主笔记。完成实验后, 你可以将其合并到主分支或丢弃实验结果。
6. **透明和审计**: 所有更改都记录在案, 提供了透明的记录和审计轨迹。这对于学术工作尤其重要, 因为它可以帮助验证和解释如何得出特定的结论。
7. **跨设备访问**: 通过使用云服务 (如 GitHub、GitLab), 你可以在不同的设备上访问和同步你的笔记, 确保你随时可以获取最新版本。

这些优点使得版本控制成为管理数学笔记的一个强大工具, 尤其在处理复杂的数学内容和长时间的研究过程中尤为重要。

Git版本控制基础流程

以下是如何在数学笔记记录中使用 Git 进行版本控制的说明, 基于你提供的内容:

开始版本控制

首先, 设置你的 Git 配置, 以确保你在版本控制中的所有提交都标记上你的名字和电子邮件地址。

SHELL

```
cd /path/to/your/project
git init .
cd .git
git config user.email "your-email@example.com"
git config user.name "your-name"
```

提交初始版本

初始化 Git 仓库并提交你的第一个版本。这一步标志着你的笔记进入版本控制的起点。

SHELL

```
cd ..  
git add .  
git commit -m "Initial commit"
```

为阶段开发版本打标签

标签帮助你标记重要的版本，如阶段性的完成版本。

SHELL

```
git tag -a <tag-name> -m "<tag-comment>"  
# 例如  
git tag -a v1.0 -m "Initial working version"  
# 可以推送到远程  
# 推送指定标签  
git push origin <tag-name>  
# 推送所有标签  
git push --tags  
# 列出所有标签  
git tag  
# 删除本地标签  
git tag -d <tag-name>  
# 删除远程标签  
git push origin --delete <tag-name>
```

如果你想从 v1.0 添加新功能

1. 切换到标签版本并从该版本创建新分支，以在新分支上开发新功能。

SHELL

```
git checkout v1.0  
# 注意此时为无头 stage，无法追踪版本更改  
# 因此我们需要以 v1.0 为头新建一个分支  
git checkout -b <feature-v1.1>
```

2. 在新分支上进行开发，并将更改提交到该分支。

```
git add .  
git commit -m "Add new feature for v1.1"
```

3. 将新功能合并到主分支，并推送到远程仓库。

```
git checkout main  
git merge <feature-v1.1>  
# 可以选择保留新分支到远程  
git push origin <feature-v1.1>  
# 提交合并到远程  
git push origin main  
# 如果开发完毕，你可以删除这个分支  
# 本地删除  
git branch -d <feature-v1.1>  
# 远程删除  
git push origin --delete <feature-v1.1>
```

分支切换

1. 在切换分支之前，最好先提交当前修改。

```
git add .  
git commit -m "Describe your changes"
```

2. 如果不想提交当前修改，可以丢弃修改或使用暂存功能。

```
# 丢弃修改  
git checkout -- .  
# 或者暂存修改  
git stash  
# 切换到目标分支  
git checkout bugfix  
# 完成工作后恢复暂存  
git stash apply
```

3. 列出和管理暂存的内容。

```
git stash list
git stash drop stash@{n}
git stash clear
```

提交到远程仓库 (GitHub/GitLab)

1. 在 GitHub 或 GitLab 上创建一个新仓库。
2. 添加远程仓库地址。

```
git remote add origin <your-repo-URL>
```

3. 如果远程仓库已有文件，如 `README.md` 和 `LICENSE`，需要先同步下来并解决冲突。

```
git pull origin main --allow-unrelated-histories
```

4. 解决 `.gitignore` 文件中的冲突。

```
# 修改 .gitignore 文件
git add .gitignore
git commit -m "Resolve merge conflict in .gitignore"
git push -u origin main
```

从以前提交的版本中恢复文件

1. 查找文件的历史修改记录。

```
git log -- requirements.txt
```

2. 比较版本之间的差异，并恢复到以前的状态。

```
git diff ^HEAD~1 README.md  
git restore --source=HEAD~1 -- requirements.txt
```

3. 提交恢复后的文件，并推送到远程仓库。

```
git commit -am "Restore requirements.txt to its state from one commit before HEAD"  
git push origin main
```

这些操作将帮助你高效地管理和维护数学笔记，使其在版本控制下更加有序和安全。

附录

- Git 官方文档: [Git Documentation](#)
- GitHub 帮助文档: [GitHub Help](#)

实战训练：提交本Obsidian笔记仓库到Github

1. 添加 `.gitignore` 到仓库根目录

```
# Obsidian 相关
*.md~
*.md#*
.obsidian/
.vscode/
.DS_Store

# 保留 attachments 中的所有文件
# 忽略 LaTeX 输出目录中的其他文件, 保留 .tex, .html 和 .pdf
MathSoftware/out/*
!MathSoftware/out/*.tex
!MathSoftware/out/*.html
!MathSoftware/out/*.pdf

# LaTeX 临时文件 (已由 out/* 覆盖, 不重复)
*.aux
*.fdb_latexmk
*.fls
*.log
*.synctex(busy)
*.synctex.gz
*.synctex.gz.sum.synctex

# Python 相关
python-dev/venv/
*.pyc
__pycache__/*
*.pyo
*.pyd

# Windows 临时文件
*.bak
*.tmp
Thumbs.db
```

2. 初始化仓库

使用streamlit开发Python程序

1. 创建一个开发目录

```
cd /d e:\MathNotes\MathSoftwear  
mkdir python-dev
```

2. 创建虚拟开发环境

```
pyenv local 3.11.9-win32  
python --version  
python -m venv venv  
.\venv\Scripts\activate  
pip config set global.index-url https://pypi.mirrors.ustc.edu.cn/simple/  
pip install streamlit
```

3. 你需要提前安装Visual Studio: [Visual Studio: 面向软件开发人员和 Teams 的 IDE 和代码编辑器](#)，下载Community2022社区版，对个人开发者免费
4. 安装时选择python以及C++，其他默认即可。
5. 如果还是安装报错，可能是需要使用64位的python：

```
pyenv install 3.11.9  
pyenv local 3.11.9  
python -m venv venv  
.\venv\Scripts\activate  
pip config set global.index-url https://pypi.mirrors.ustc.edu.cn/simple/  
pip install streamlit  
# 运行示例  
streamlit hello
```

6. 学习streamlit: [30天入门Streamlit](#)