版本控制与Git基础

日期：2025/06

目录

01 版本控制概述

版本控制的定义

- **文件备份功能**：对不断变化的文件进行备份，防止丢失或损坏，保障数据安全。
- **历史记录存储与访问**：存储并提供带注释的历史记录，方便追溯文件演变过程。
- **变更合并管理**：协调多人对同一文件的修改，管理不同变更集的合并。

版本控制的重要性

- **确保结果可重复性**：通过版本控制恢复代码状态，重新修改提交实验结果。
- **科研工作的价值体现**：记录软件和数据的开发历史，类似实验室笔记本，提高科研严谨性。

案例

吴博士在模拟核物理实验时使用Git进行版本控制，当数月后收到期刊审稿意见要求将论文中带电粒子电流的符号显示方式从负电流改为正电流时，尽管期间代码已发生多次改动，但她通过Git的版本历史记录快速定位到论文提交时的代码版本，执行命令将代码回滚到当时状态，仅花费少量时间修改图表符号并重新运行绘图脚本提交修订版，之后又顺利将代码库更新到最新状态，整个过程高效可逆，避免了因无法追溯历史版本而可能导致的数周整理文件的麻烦，充分体现了版本控制在科研中保障代码可回溯性和实验可重现性的关键作用。

版本控制的工具类型

集中式版本控制系统

集中式系统指定单一的中央存储库位置，如Concurrent Versions System(cvs)、Subversion (svn)、Perforce (p4) 。所有操作都依赖中央服务器，适合小型团队。

分布式版本控制系统

分布式系统将所有位置视为平等，如Decentralized CVS (dcvs)、mercurial (hg)、bazaar (bzr)、Git (git) 。更适合协作项目，管理和合并多开发者的更改能力强且用户友好。

重点工具Git

Git由Linux创建者Linus Torvalds编写，是分布式版本控制系统。因其流行、灵活和协作性强，后续将以它为例讲解版本控制概念。

02 Git基础操作

Git的安装与配置

Git的获取与安装

检查是否安装在终端输入 `which git` ，若返回路径（如 `/usr/bin/git` ），表示系统已安装。

未安装时的操作

macOS: 通过 `brew install git` 或Xcode命令行工具安装；

Linux: 使用 `sudo apt install git` ；

Windows: 从git-scm.com 下载官方安装包。

用户信息配置

全局用户信息

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

用于标记每次提交的作者身份。

默认编辑器配置

```
git config --global core.editor "code --wait"
```

指定VS Code或Vim等编辑器用于编辑提交信息。

验证配置结果

查看所有配置项： `git config --list` 输出中若显示用户信息，说明配置成功。

创建仓库与文件管理

创建本地仓库

初始化仓库

`mkdir parity_code` `cd parity_code` `git init`

成功后生成隐藏目录.git，其中保存所有版本信息。

查看隐藏目录 `ls -A`

输出中若包含.git说明初始化成功。

添加与提交文件

添加文件至暂存区 `git add readme.md`

将文件加入监控列表（Stage Area）。

提交文件 `git commit -m`

首次提交：添加项目说明文件,生成一个永久的版本快照（commit）。

提交信息的重要性

特点：清晰、简洁、具体。例如：

`feat`：添加数据预处理模块

`fix`：修复图像路径错误

`doc`：更新实验记录说明

有助于后续追踪和协作。

状态、差异与撤销操作

查看仓库状态与历史

查看当前状态 `git status`
显示当前分支、已暂存文件、未跟踪文件。
查看提交历史 `git log`
包含提交ID、作者、时间与信息，可使用--oneline简化显示。

查看文件差异

未暂存更改 `git diff`
对比工作区与暂存区差异。
已暂存但未提交的更改 `git diff --cached`
差异统计摘要git diff --stat

撤销与回退操作

取消暂存 `git reset HEAD readme.md`
取消 `git add` 操作。
撤销修改 `git checkout -- readme.md`
放弃工作区未提交的修改。
版本回退`git reset --hard <commit_id>
回到指定版本（操作会丢失当前更改）。

分支的概念和作用

什么是分支（branch）

分支是仓库中的一个“平行宇宙”——它允许在不影响主版本的情况下进行修改和实验。
每个分支都是对项目历史的一条独立线索，指向不同的提交（commit）。

分支的意义

在科学计算或科研项目中，分支可用于：
--开发新算法或模型；
--测试不同参数设定；
--维护稳定版与实验版共存；
主分支（通常为 main 或 master）保持稳定，其他分支负责创新或试验。

查看当前分支
git branch
星号（*）标记当前所在分支，例如：

- master
  experimental

分支的创建和管理

分支的创建与切换

创建新分支 `git branch experimental`
此命令从当前分支创建一个名为 experimental 的分支。
切换分支 `git checkout experimental`
切换后，所有操作（编辑、提交等）都会发生在该分支上。
创建并切换一步完成 `git checkout -b experimental`
相当于先创建分支再切换。
验证分支切换成功 `git status`
终端中会显示： `On branch experimental`

删除与管理分支

删除分支（已合并） `git branch -d experimental`
若分支内容已合并回主分支，可安全删除。
强制删除分支（未合并） `git branch -D experimental`
警告：此操作会永久丢失未合并的工作。

分支的合并

合并语法：先切换到目标分支，再执行合并命令：

```
git checkout master
git merge experimental
```

此操作会把 experimental 的提交整合进 master，形成新的合并提交。

快进合并（Fast-forward） 若目标分支无额外提交，Git会自动“快进”至新版本，不产生新节点。

合并冲突（Merge Conflict） 当两个分支修改了同一文件的同一行时，Git无法自动决定保留哪一方。

例如：

Dr. Nu 在 master 的首行改为 "Bonjour" ,Fran 在 experimental 的首行改为 "Howdy"

合并时，Git会在文件中插入冲突标记：

```
<<<<<<< HEAD
Bonjour
=======
Howdy
>>>>>>> experimental
```

用户需手动修改文件、保留正确内容，再执行： `git add 文件名` `git commit`

表示冲突已解决。

查看分支历史与合并图

可视化分支结构 `git log --graph --oneline --all`

输出图示结构化的分支与合并轨迹，便于理解项目演化。

03 远程版本控制

远程仓库与托管平台

远程仓库概念: Git 允许将本地仓库与服务器上的远程仓库同步，实现多人协作。
远程仓库相当于“中心节点”，存储共享版本记录与分支信息。

常见托管服务

--Launchpad：由 Canonical（Ubuntu 开发商）提供，偏向开源社区。
--Bitbucket：支持私有仓库与团队协作，适合小组项目。
--Google Code / SourceForge：较早的开源项目托管平台。
--GitHub：目前最主流的 Git 托管服务。

GitHub 的功能优势

--代码托管与版本控制集成；
--支持 README 展示与 Markdown 渲染；
--提供 Wiki、问题跟踪（Issues）、版本发布（Releases）；
--可视化网络图（Network Graph）与代码高亮；
--支持协作者权限控制、邮件通知与 Pull Request 工作流；
--适合科研、开源及教学项目协同开发。

创建 GitHub 仓库

注册与登录

前往 GitHub 官网，注册账号并登录。

创建远程仓库空间

依次点击：

New repository → 输入仓库名（如 parity_code） → 选择公开/私有 → Create repository

获得仓库URL

创建完成后，系统提供两种远程地址：

```
HTTPS: https://github.com/NouveauNu/parity_code.git
SSH: git@github.com:NouveauNu/parity_code.git
```

后续将本地项目与该地址关联。

关联本地与远程仓库

git remote命令 用于管理远程仓库地址及别名：

```
git remote add <别名> <仓库URL>
git remote -v # 查看所有远程仓库
git remote rename <旧名> <新名>
git remote remove <别名>
```

常见约定：主远程仓库命名为 origin。

示例：以 Dr. Nu 的项目为例

```
git remote add origin https://github.com/NouveauNu/parity_code.git
```

此命令在本地注册一个名为 origin 的远程仓库。之后，Git 会将推送（push）与拉取（pull）操作默认指向该 URL。

查看远程信息 `git remote -v`

输出示例：

```
origin https://github.com/NouveauNu/parity_code.git (fetch)
origin https://github.com/NouveauNu/parity_code.git (push)
```

表明当前本地仓库已成功与 GitHub 仓库建立连接。

后续操作预览

推送本地提交至远程仓库 `git push -u origin master`

首次推送时使用 -u 参数建立追踪关系。

从远程拉取更新 `git pull origin master`

使本地仓库与远程保持同步。

推送与拉取（Push & Pull）

推送本地提交到远程仓库

推送命令 `git push origin master`
将当前分支（如 master）的最新提交上传至远程仓库的对应分支。
首次推送 `git push -u origin master`
`-u` 选项建立本地与远程分支的“追踪关系”，之后可直接使用 git push。
常见场景
--上传新的实验结果；
--分享可复现分析脚本；
--团队成员同步更新。

拉取远程更新到本地

拉取命令 `git pull origin master`
相当于两步操作：

```
git fetch origin master # 从远程获取更新
git merge origin/master # 自动合并至本地
```

同步的重要性
--保持代码、数据和分析脚本一致；
--避免因多人修改造成的历史分歧；
--减少冲突与数据丢失风险。

克隆远程仓库（Clone）

克隆操作

命令示例 `git clone https://github.com/NouveauNu/parity_code.git`
该命令会：
--在本地创建一个名为 parity_code 的文件夹；
--自动初始化 .git 目录；
--建立与远程仓库的追踪关系；
--默认签出主分支（master 或 main）。

克隆后的典型操作流程

查看分支 `git branch`
创建新分支 `git branch dev`
切换分支 `git checkout dev`
修改与提交 `git add 文件名` `git commit -m` "添加分析模块"
推送更新 `git push origin dev`
推送后，团队成员即可在 GitHub 上查看新分支并进行评审或合并。

克隆的意义

--便于协作开发；
--在科研中确保实验环境一致；
--任何人都能在本地独立复现他人结果。

冲突与解决（Conflicts）

冲突产生的原因

当多人修改同一文件的同一部分内容时（例如实验说明或绘图参数），Git 无法自动判断保留哪一个修改。

例如：Dr. Nu 在 master 中将 readme.rst 首行改为 “Bonjour”；Fran 在 experimental 分支中改为 “Howdy”；

两人同时提交并尝试合并时会触发冲突。

冲突标记示例

合并后文件中会出现以下内容：

```
<<<<<<< HEAD
Bonjour
=======
Howdy
>>>>>>> experimental
```

HEAD：表示当前分支的内容；experimental：表示合并分支的内容。

冲突解决步骤

打开冲突文件并编辑,删除冲突标记，只保留正确版本；

标记已解决 `git add readme.rst`

重新提交 `git commit`

若自动生成的提交信息包含“Merge branch”，表示合并已完成。

解决建议

--合并前先执行 git pull 保持最新；

--在分支中进行独立实验；

--定期与主分支同步，减少冲突概率。

04 版本控制总结与拓展

版本控制总结

版本控制核心要点

版本控制可备份文件、存储历史记录、管理变更合并，是科研和软件开发中确保结果可重复性的关键。

本地版本控制操作

本地操作包括创建仓库 `git init`、暂存文件 `git add`、提交快照 `git commit`、查看状态 `git status` 和历史 `git log` 等。

远程版本控制操作

远程操作有关联仓库 `git remote`、推送 `git push`、拉取 `git pull`、克隆 `git clone` 等，实现本地与远程仓库同步。

版本控制重要性

在科研和软件开发中，版本控制能有效管理代码和数据，便于回溯和协作，提高工作效率和质量。

拓展资源推荐

- **Pro Git Book**：免费开源电子书，全面介绍Git功能与场景。
- **Software Carpentry的Git Lessons**：系统课程，通过案例练习掌握操作。
- **Software Carpentry的Git Reference**：简洁参考手册，快速查找命令。

THE END

谢谢！