git简介与安装

1. Git 是什么?

Git 是一款由 Linus Torvalds（Linux 操作系统创始人）于 2005 年开发的**分布式版本控制系统（Distributed Version Control System, DVCS）**，专门用于管理代码（或文本类文件）的修改历史，支持多人协同开发。

- **核心属性**：分布式 —— 与传统的“集中式版本控制”（如 SVN）不同，Git 不需要依赖中央服务器，每个开发者的本地设备都会保存完整的代码仓库（包含所有历史版本和分支），即使离线也能进行版本管理。
- **本质定位**：不是“代码存储工具”，而是“代码修改的追踪与管理工具”，核心是记录文件从创建到每次修改的“快照”，并支持对这些快照的灵活操作（如回滚、对比、合并）。

2. Git 的作用

Git 的核心作用是解决“代码管理中的痛点”，具体可分为以下 4 类：

1. **追踪版本历史**：记录每一次代码修改（谁改的、什么时候改的、改了什么内容），可随时查看

任意版本的代码，或回滚到历史版本（比如代码引入 bug 时，快速恢复到之前的正常版本）。

2. **支持多人协同**：多个开发者可同时修改同一项目的不同文件（或同一文件的不同部分），Git 能自动合并大部分修改，若出现冲突（多人修改同一行代码），也能提供清晰的冲突标记，帮助开发者手动解决。
3. **隔离开发任务**：通过“分支”功能，可在不影响主代码（如 `main` 分支）的前提下，创建独立的开发分支（如 `feature/login` 开发登录功能、`fix/bug123` 修复 bug），开发完成后再将分支合并到主代码，避免开发过程中污染稳定版本。
4. **备份与安全**：由于是分布式架构，每个开发者的本地仓库都是完整备份，即使中央仓库（如 GitHub、GitLab）故障，也能从任意本地仓库恢复完整代码，降低代码丢失风险。

3. Git 的优势

对比集中式版本控制（如 SVN）或无版本控制（手动备份文件），Git 有明显优势，具体如下表：

优势维度	具体说明
分布式架构	本地保存完整仓库，离线可操作；无“单点故障”，备份更安全。
速度快	大部分操作（如提交、分支切换、查看历史）基于本地完成，无需网络请求，响应迅速。
分支轻量	Git 分支本质是“指针”（指向某个版本快照），创建、切换、合并分支的成本极低（毫秒级）。
冲突易解决	提供可视化的冲突标记（如 `<<<<<<< HEAD` `=======` `>>>>>>> branch-name`），支持逐行对比并手动选择保留内容。
开源免费	完全开源，可自由修改和扩展；无商业授权费用，个人和企业均可免费使用。

生态丰富	可与主流代码托管平台（GitHub、GitLab、Gitee）无缝集成，支持 CI/CD（持续集成 / 持续部署）流程。

4. Git 一般使用流程（基于“本地仓库 + 远程仓库”协同）

日常开发中，Git 通常配合远程仓库（如 GitHub）使用，核心流程可分为“初始化 / 克隆仓库”“本地开发与提交”“同步远程仓库”三步，具体步骤如下：

步骤 1：初始化或克隆仓库（建立本地与远程的连接）

- **场景 1：新项目（本地先创建，再关联远程）**
  a. 本地创建项目文件夹，进入文件夹后打开终端，执行 `git init`：初始化本地 Git 仓库（生成隐藏的 `.git` 目录，存储版本信息）。
  b. 关联远程仓库：执行 `git remote add origin [远程仓库地址]`（如 `git remote add origin https://github.com/xxx/xxx.git`），将本地仓库与远程仓库绑定。
- **场景 2：已有远程仓库（直接克隆到本地）**
  a. 执行 `git clone [远程仓库地址]`：将远程仓库的完整代码（含所有历史版本）下载到本地，自动创建与远程仓库的关联。

步骤 2：本地开发与提交（记录代码修改）

1. **编写代码**：在本地项目中编写或修改代码（如新增 `index.js`、修改 `style.css`）。
2. **查看修改状态**：执行 `git status`：查看当前文件的修改状态（哪些文件已修改、未被 Git 追踪）。
3. **暂存修改**：执行 `git add [文件名]`（如 `git add index.js`）或 `git add .`（暂存所有修改文件）：将修改的文件“暂存”到“暂存区”（临时存储待提交的修改）。
4. **提交到本地仓库**：执行 `git commit -m "提交说明"`（如 `git commit -m "feat: 新增登录页面布局"`）：将暂存区的修改“提交”到本地仓库，生成一个新的版本快照，`-m` 后的说明需清晰描述本次修改内容（便于后续查看历史）。

步骤 3：同步远程仓库（与团队协同）

- **场景 1：本地修改推送到远程（如完成功能后提交代码）**

a. 先拉取远程最新代码（避免冲突）：执行 `git pull origin [分支名]`（如 `git pull origin main`），将远程仓库的最新代码同步到本地，若有冲突需先解决。

b. 推送本地代码到远程：执行 `git push origin [分支名]`（如 `git push origin main`），将本地仓库的提交推送到远程仓库，完成同步。

- **场景 2：获取远程最新代码（如团队其他成员提交了新代码）**

a. 执行 `git pull origin [分支名]`：直接将远程指定分支的最新代码拉取到本地，并自动合并到当前本地分支（若合并失败，需手动解决冲突后再提交）。

补充：分支相关常用流程（隔离开发任务）

1. **创建并切换到新分支**：执行 `git checkout -b [分支名]`（如 `git checkout -b feature/payment`），创建“支付功能”分支并切换到该分支。
2. **在新分支开发**：重复“步骤 2”的提交流程（`add` → `commit`）。
3. **合并分支到主分支**：开发完成后，切换回主分支（`git checkout main`），执行 `git merge [新分支名]`（如 `git merge feature/payment`），将新分支的代码合并到主分支。
4. **推送合并后的主分支到远程**：执行 `git push origin main`，同步到远程。

5.windows下面安装git：

下载git 到git官网上下载，自行选择合适系统的 https://git-scm.com/ 如果是win版本的，直接按默认安装即可