上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

软件工程
SOFTWARE ENGINEERING

Git

沈备军

饮 水 思 源 · 爱 国 荣 校

代码版本管理场景1：代码演化历史跟踪

- 软件开发过程中新增或修改代码，例如
  - 可能花了2周时间，写了6000行代码来实现一个客户或产品经理所要求的新特性
  - 花了2天时间，改动了30个文件来重构代码以提高软件的可维护性
  - 花了3个小时，改动了4个方法来修复了一个测试人员报告的缺陷

- 如果没有代码版本管理
  - 不记得为何、在何处以及如何修改的代码
  - 导致自己以及他人难以理解代码的变动过程、开展代码评审以及及时发现修改过程中引入的缺陷

代码版本管理场景2：历史版本回退

- 开发人员希望回到此前的一个稳定的代码版本上，例如
  - 新特性上线后不符合客户或市场要求，需要去除这一新特性
  - 由于增加新特性、代码重构或其他原因而导致重要的功能不可用

- 如果没有代码版本管理
  - 很难准确而快速地回退到指定的代码版本，从而增加了代码维护的难度

代码版本管理场景3：多人开发协作冲突

- 多人协作开发可能出现冲突，例如
  - 花了很长时间修改了一个代码文件，然后发现这个文件已经被其他开发人员修改或者删除了
  - 修改完代码提交后过了一段时间发现被其他开发人员覆盖了

- 如果没有代码版本管理
  - 协作冲突造成修改内容丢失
  - 协作冲突难以被察觉和及时处理
  - 最终导致开发上的混乱

代码版本管理场景4：代码质量检查

- 开发人员可能会将有质量问题（功能缺陷或可维护性问题）的代码合入到版本库中
  - 造成潜在的缺陷隐患
  - 影响其他开发人员的代码理解和修改
- 需要根据规范要求检查需要合入的代码
  - 通过代码自检、同行评审以及门禁检查等手段
  - 通过检查的代码才允许提交或合入主干分支

- 如果没有代码版本管理
  - 难以形成代码质量管控的“关口”
  - 难以有效地管控开发人员所提交的代码质量

版本控制系统

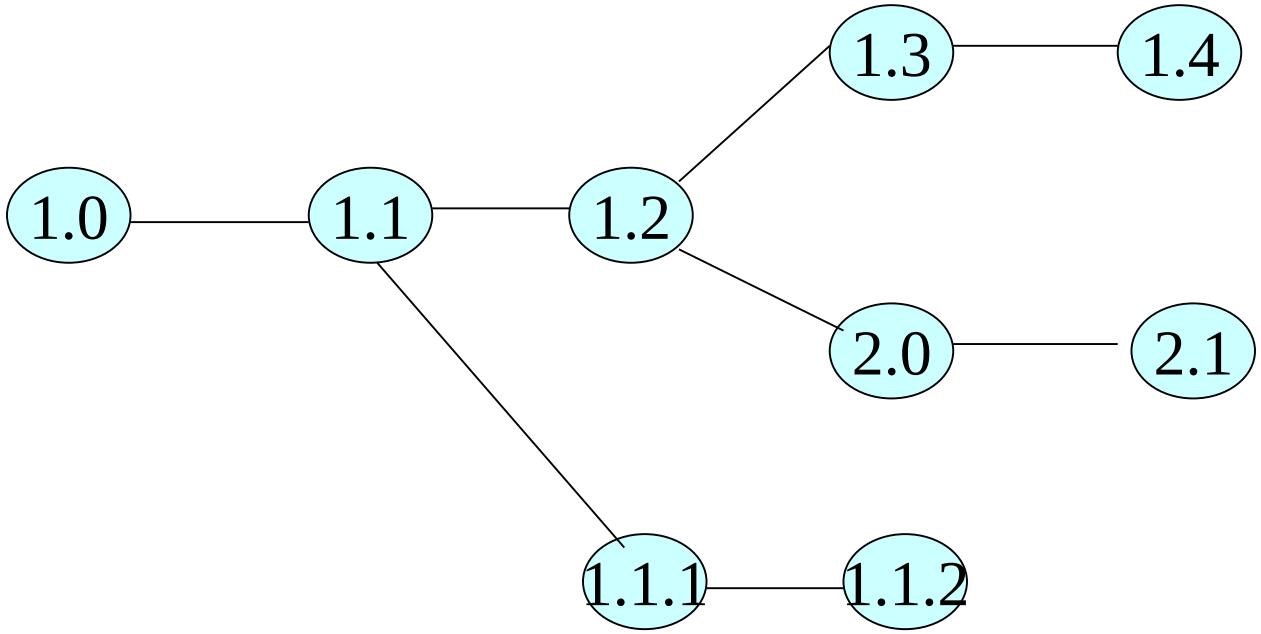
- 实现代码版本管理目标的一种有效途径
  - 存储、追踪代码修改的完整历史记录（即版本库）
  - 提供多种机制帮助开发人员进行协同开发
  - 实现代码合入前的质量检查等门禁检查功能

软件配置项

- 软件版本管理的基本实体是软件配置项 (Software Configuration Item, SCI)。
- 一个软件配置项是在软件生存周期所产生或使用的一个工作产品或一组相关的工作产品，包括代码、文档、模型、数据等。

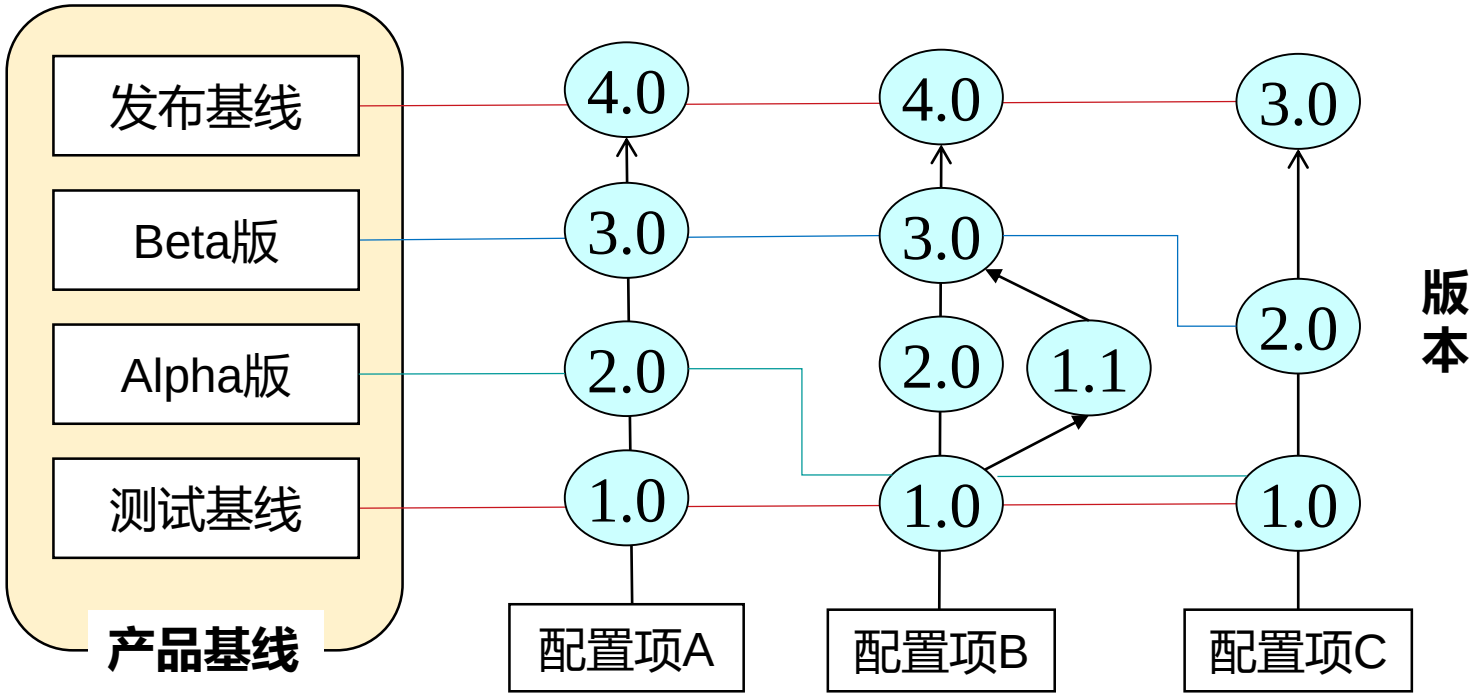
版本 version

- 配置项在软件开发过程中会不断变更，形成多个版本(Version)。
- 每个配置项的版本演化历史可以形象地表示为图形化的版本树(Version Tree)。



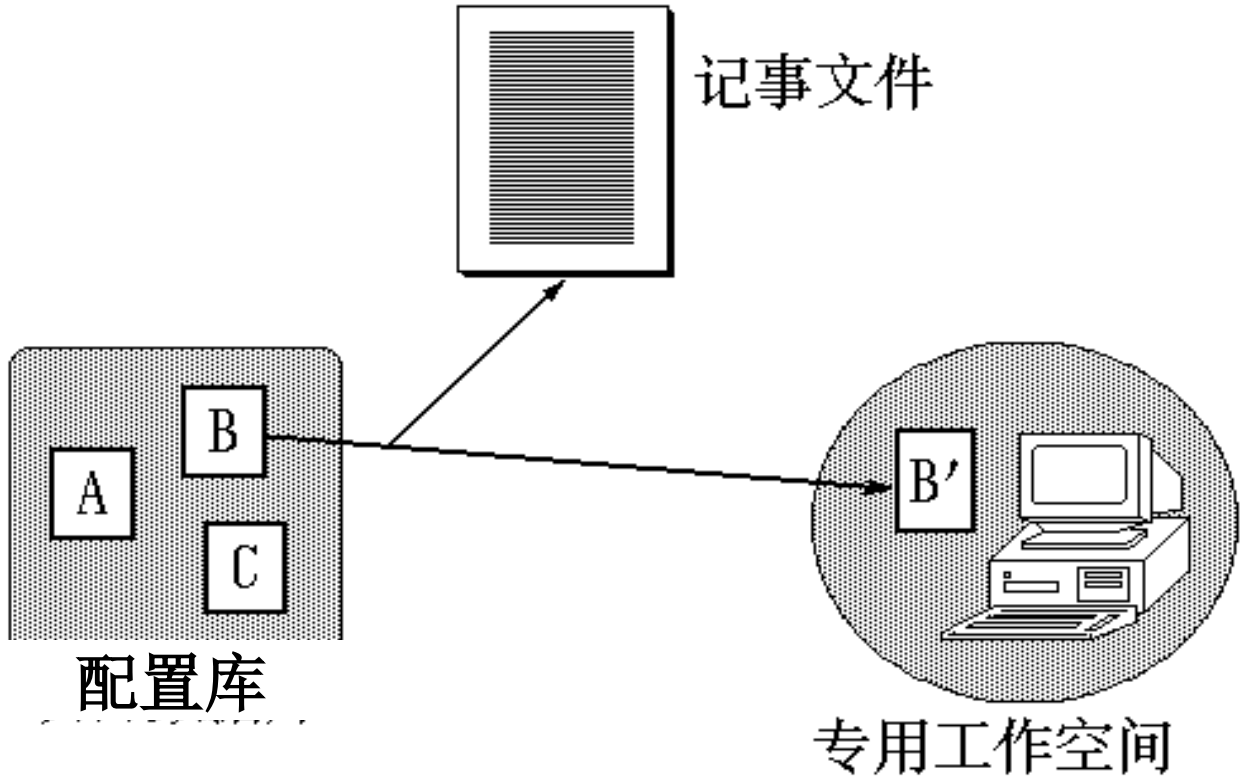
基线 baseline

- 基线是软件产品中各配置项版本在特定时期的一个“快照”，它提供一个正式标准，随后的工作基于此标准，并且只有经过授权后才能变更这个标准。
- 一个软件产品可有多条基线。

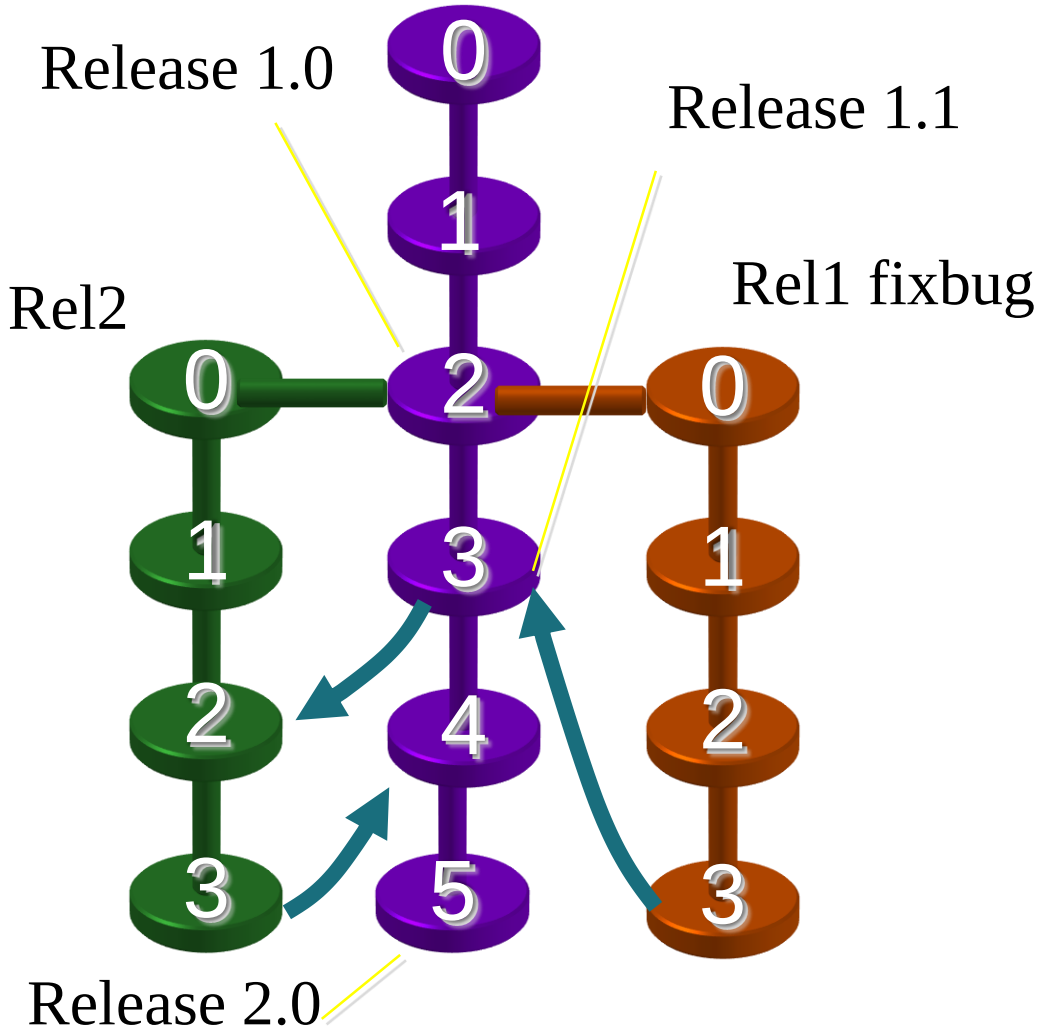


版本库 Repository

- 存储配置管理信息及软件软件配置项的版本信息。



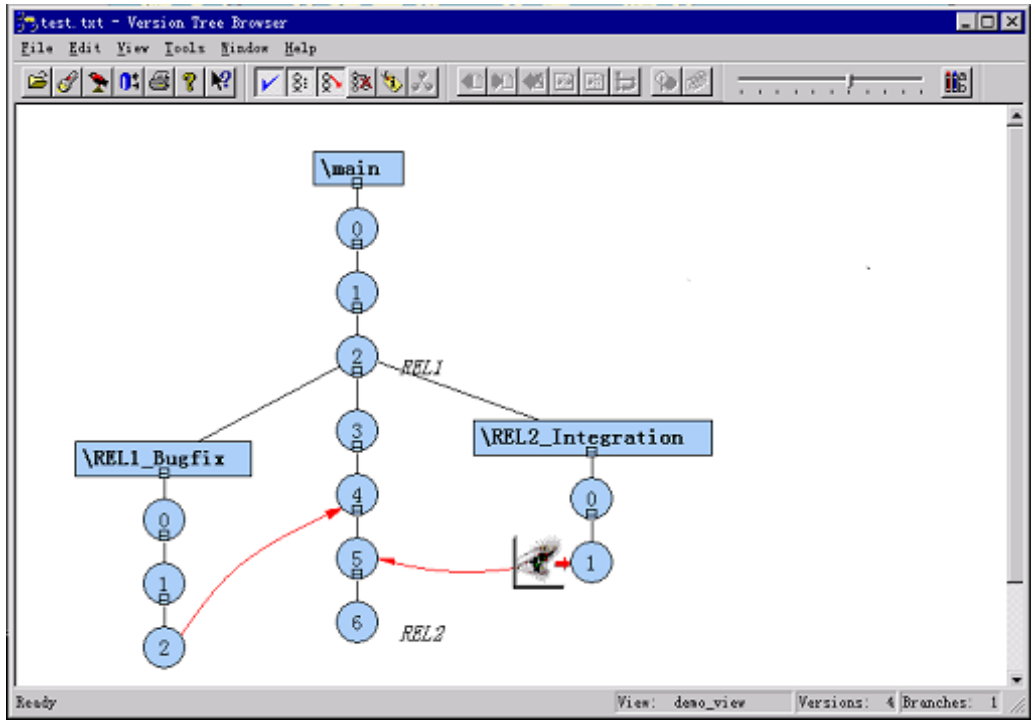
分支管理和并行开发



- 常用分支(branch)类型
  - 主分支（master）：对应版本发布
  - 开发分支（develop）：对应开发环境中的代码
  - 特性分支（feature）：对应新功能
  - 发布分支（release）：为版本发布做好准备
  - 补丁分支（hotfix）：为修复缺陷
- 分支合并(merge)
  - 发生合并冲突的文件中标记出不同分支中的内容
  - 开发人员之间对合并冲突进行协商

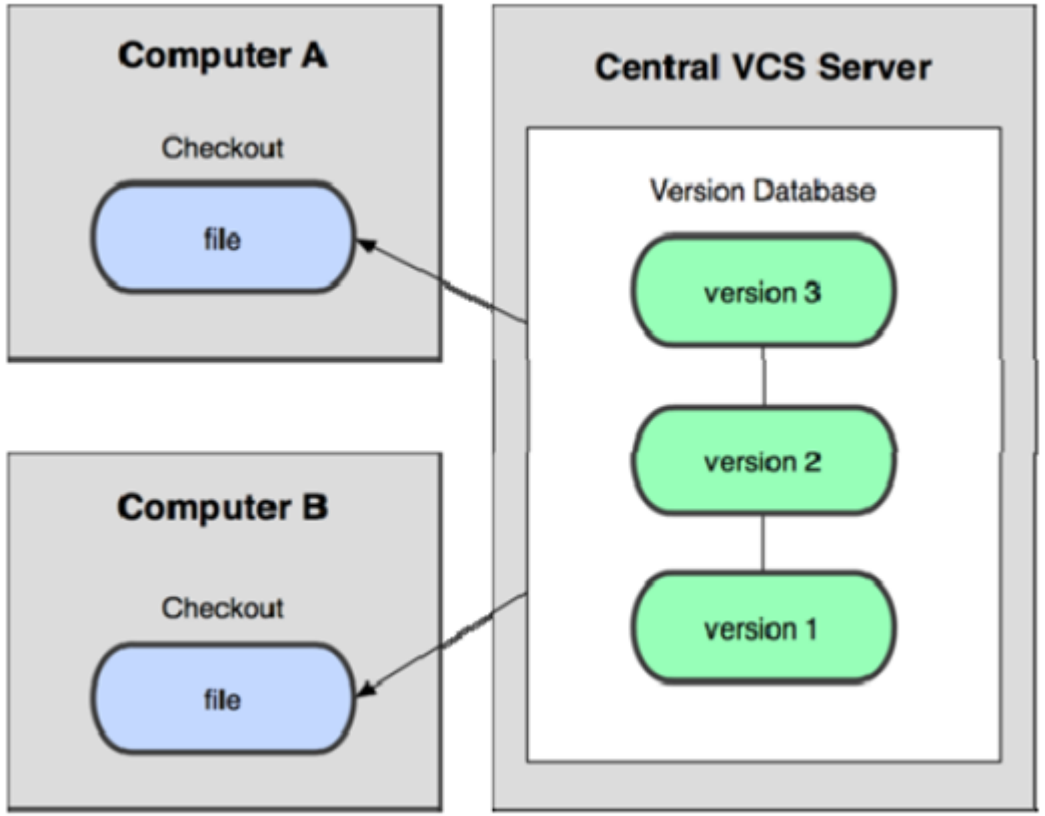
常用的版本控制系统

- 集中式
  - Merant PVCS
  - IBM Rational Clearcase
  - Microsoft Visual Source Safe
  - CVS (Freeware)
  - SVN (Freeware)
- 分布式
  - Git (Freeware)
  - GitHub 和 Gitee (云平台)



集中化的版本控制系统

- 适合于局域网小团队开发
  - 版本库集中存放在中央服务器之中
  - 开发前先从中央服务器取得最新版本
  - 开发完再把自己的工作推送给中央服务器
- 典型系统：**CVS、SVN、ClearCase**



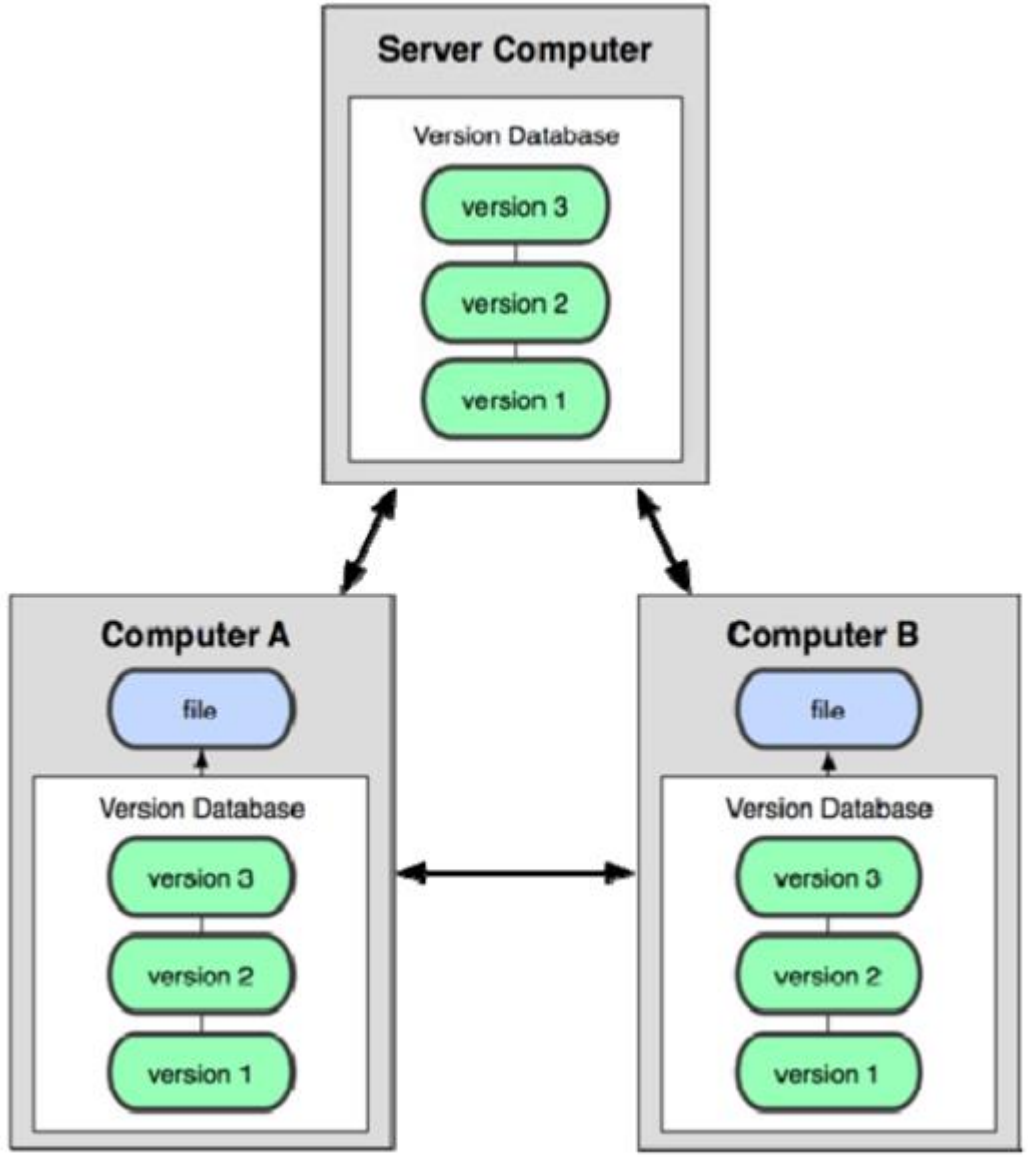
分布式版本控制系统

- 适合于任何规模的团队开发

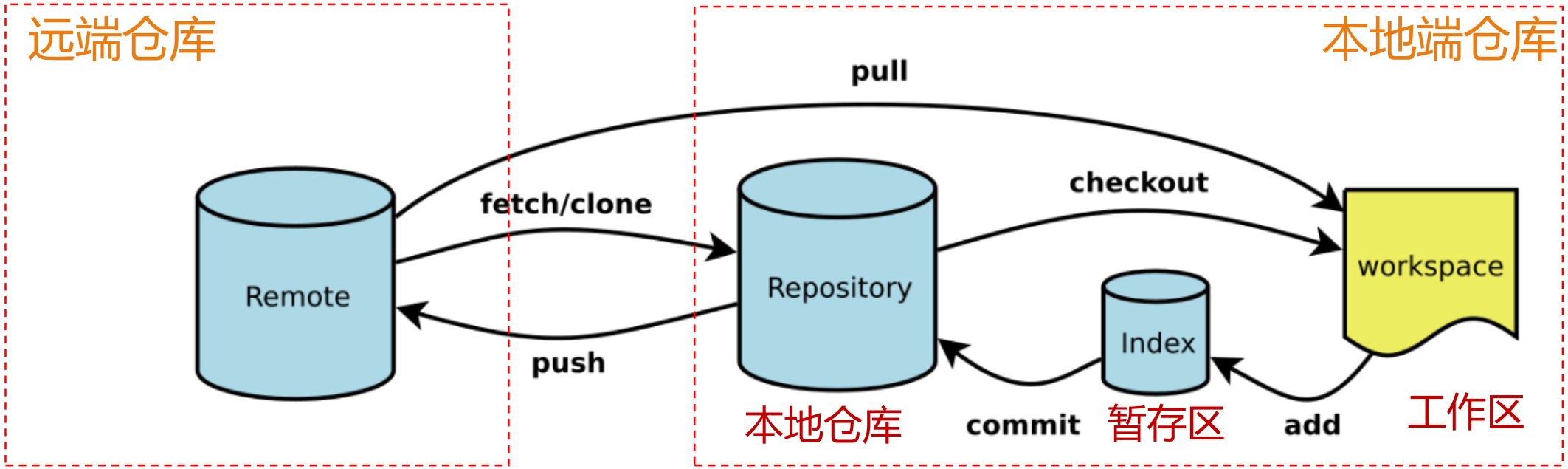
  支持Internet或局域网等各种网络

  - 有一个中央仓库
  - 开发前在本机上拷贝一个完整软件仓库
  - 开发完把自己工作提交到本地仓库中
  - 需要同步给协作者时再递交到中央仓库
  - 版本库分步存储于各协作者电脑中

- 典型系统： **Git**

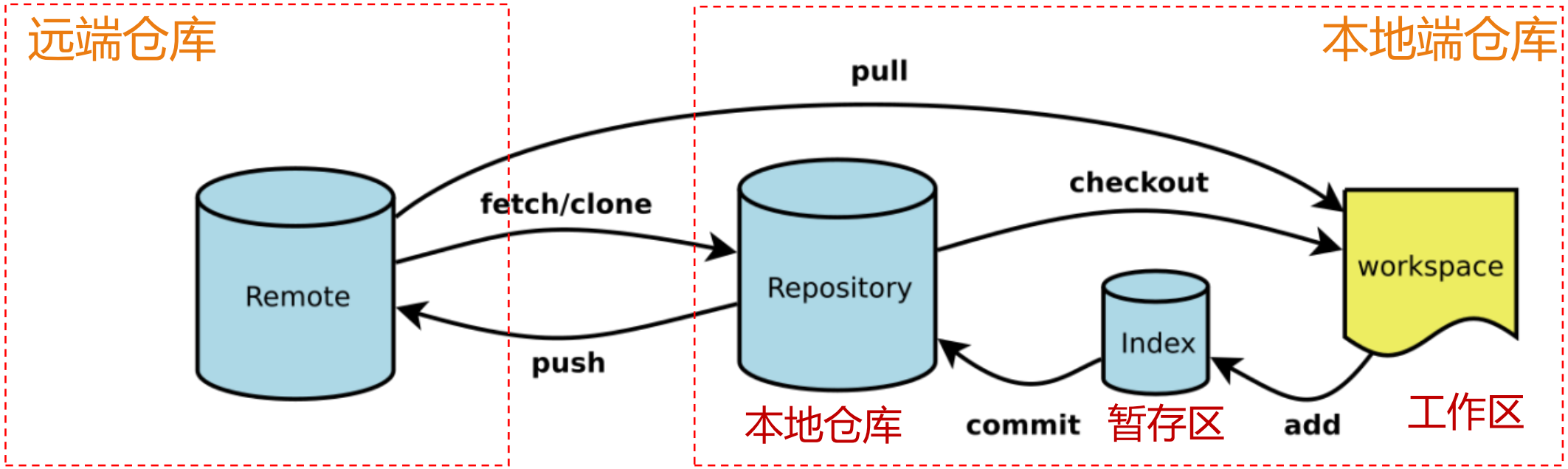


Git 的仓库组成



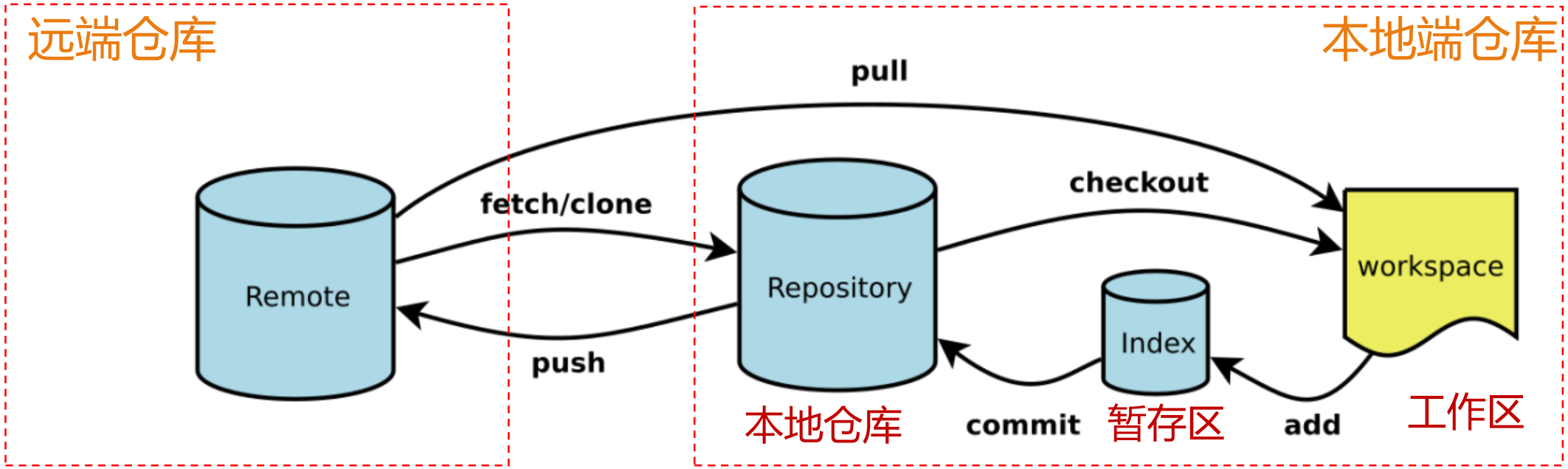
- Workspace： 工作区，是你平时存放项目代码的地方
- Index / Stage： 暂存区，用于临时存放你的改动，事实上它只是一个文件，保存即将提交到文件列表信息
- Repository： 本地仓库，存放着你提交的所有版本的数据
- Remote： 远程仓库，托管代码的服务器

Git 的常用命令



- fetch/clone命令：将中央服务器上该项目的远程仓库拷贝到本地机器上
- pull命令：将中央服务器上远程仓库的所有最新提交全部拉取到本地仓库并进行合并
- push命令：将本地仓库中的本地提交推送到中央服务器的远程仓库中

Git 的常用命令 （续）



- commit命令：将暂存区中的文件提交到本地仓库
- checkout命令：将(改乱的)文件重置为暂存区或者本地仓库中的文件内容
- add命令：将指定的文件（更改）保存到暂存区

Git中的提交（Commit）

- 原子性：提交粒度上要确保每次提交只做一件事情，不要把多件事混在一次提交中
- 在commit命令中指定该提交的描述消息
  - 类型：新功能、缺陷修复、重构、测试、文档、格式化等
  - 主题：提交的简要描述
  - 主体：提交的详细描述
  - 链接：与其他软件制品（如开发任务、缺陷）的链接关系

fix: couple of unit tests for IE9

Older IEs serialize html uppercased, but IE9 does not...
Would be better to expect case insensitive, unfortunately jasmine does not allow to user regexps for throw expectations.

Closes #392

Git 多人协同工作流1：Github Flow

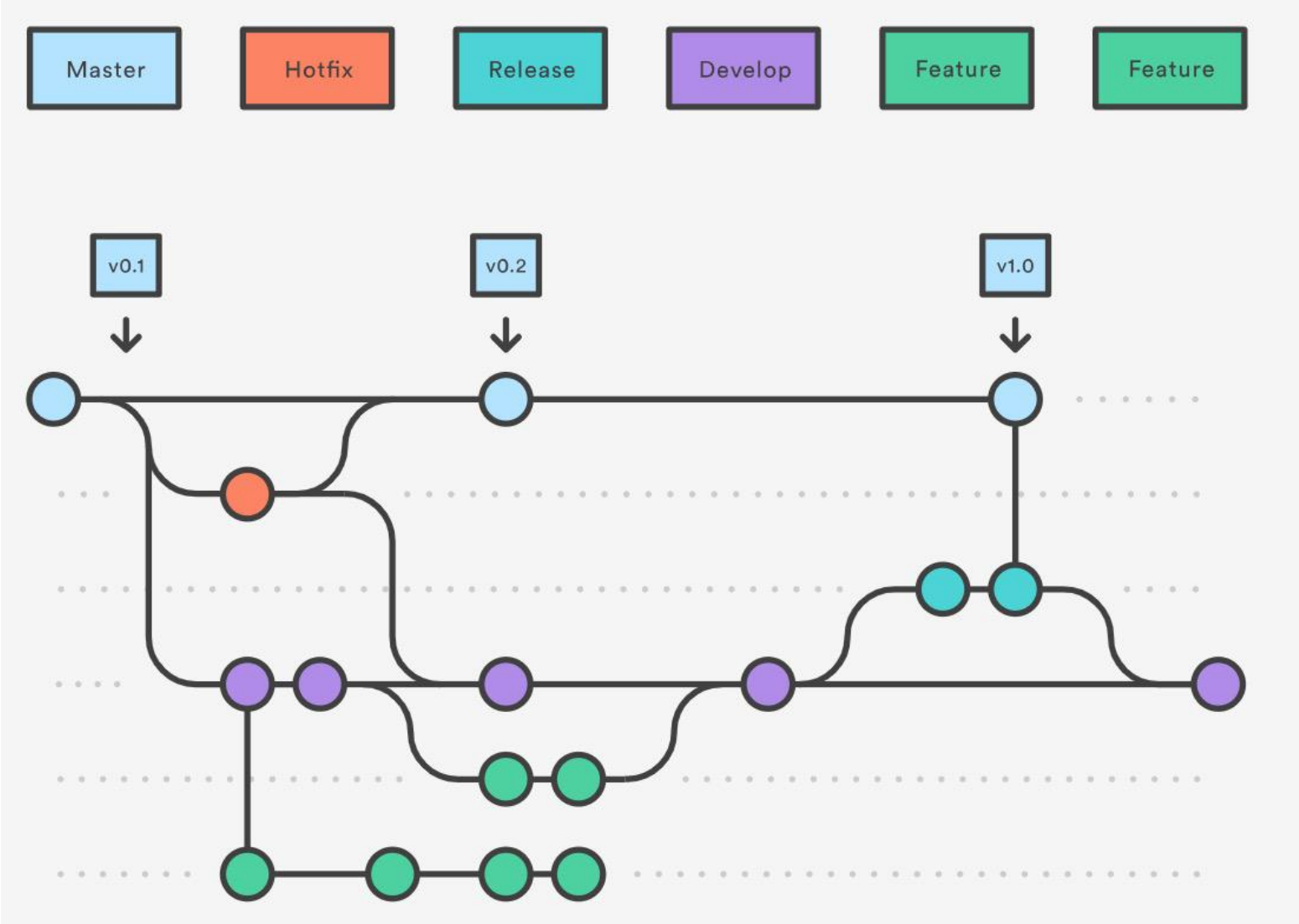
GitHub flow Docs

- 主分支：Master分支
- 短分支：开发新功能或修复缺陷时，创建一个新分支，完成后通过Pull Request (PR) 合并至Master，删除分支

Git 多人协同工作流2：Gitflow Workflow

Gitflow Workflow Tutorial

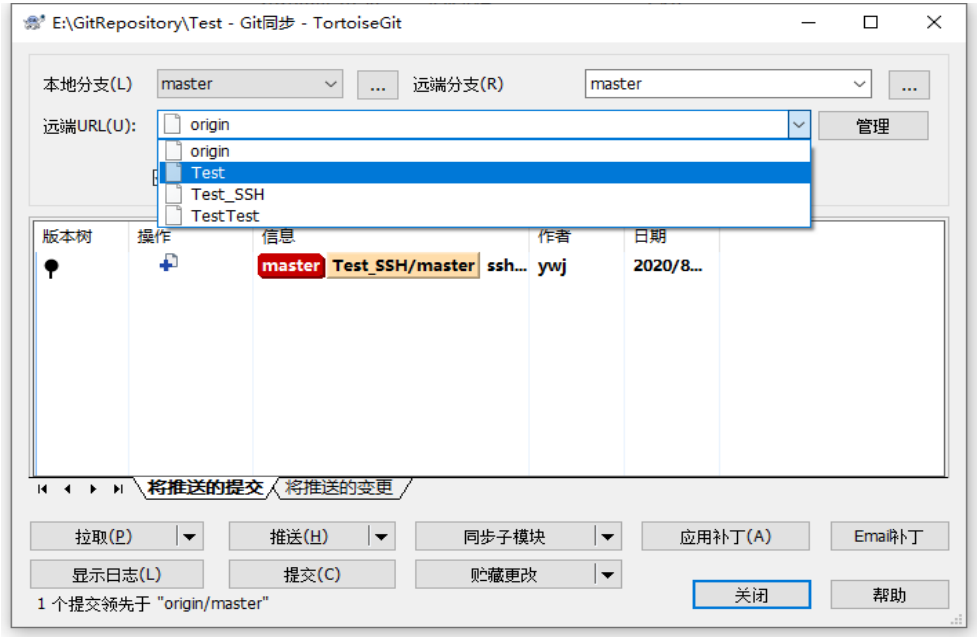
- 两个主分支：
  - Master分支保存官方的发布历史
  - Develop分支用于集成各种功能开发的分支
- 在创建新的功能开发分支时，父分支应该选择Develop



Gitflow Workflow Tutorial

Git客户端软件 TortoiseGit

- 一种开源的Git版本控制系统客户端软件
  - 它采用Windows图形化界面，使得使用Git变得更加简便和易于操作
- 安装访问TortoiseGit官网



Git 学习资料

- GitHub Guides: https://guides.github.com/ （必用）
  - Video: https://www.bilibili.com/video/BV1i44y1e7hv/?spm_id_from=333.337.search-card.all.click&vd_source=c2fea6caa8dabef148d703c539e441c8
- GitHub Flow：https://docs.github.com/cn/get-started/quickstart/github-flow （简单，建议现在使用）
  - Video: https://www.youtube.com/watch?v=juLIxo42A_s
- Gitflow Workflow: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow （高级，建议以后使用）
  - Video : https://www.youtube.com/watch?v=8UguQzmswC4
- Conventional Commits: https://www.conventionalcommits.org/en/v1.0.0/
- github/gitignore: https://github.com/github/gitignore