

数字商品生产协作工具 Git原理及实训



主讲人：庄表伟



时间：2024.11

目录 - CONTENTS

01.

版本管理简介

02.

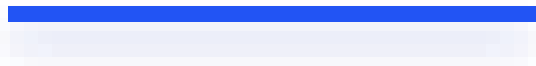
从产品经理的角度，
来看版本管理

03.

如何学习Git？

04.

开始Git实战





01

版本管理简介

物理世界里的忒修斯之船

- 忒修斯是希腊神话中的一位英雄，他的船在一场重要的航行后被保存下来，成为纪念他的象征。为了使船长期存放而不腐朽，工匠们不断将破损的木板替换为新的木板，直到整艘船的所有部件都被替换掉。
- 问题：当所有原始材料被替换后，这艘船还是原来的“忒修斯之船”吗？
 - **进一步的思考**：如果把替换下来的旧木板重新组装成一艘船，那哪艘才是真正的“忒修斯之船”？
- 这个悖论由古希腊哲学家普鲁塔克提出，后来被许多哲学家和科学家用来探讨**物体同一性、变化与身份**的关系。
- 哲学思考
 - 在物理世界里，忒修斯之船的问题引发了对**同一性**的动态理解：是取决于**物质的组成**，还是**结构的保持**，又或是**功能的延续**？

物理学无法给出哲学意义上的确定答案，但从系统连续性、功能性和信息保存的角度，可以为这一古老悖论提供新的解读和视角。



数字世界里的变化如何记录？

- 在数字的世界里，我们不必陷入这么困难的哲学思考
 - 变化肯定已经发生了，我们需要思考的是：如何记录这些变化
- 最小的变化：我们就变更最小的版本号
 - 忒修斯 1.0 → 忒修斯 1.0.0.1
- 中等的变化：我们就变更一个比较关键的版本号
 - 忒修斯 1.1 → 忒修斯 1.2
- 更大的变化：我们需要变更主版本号
 - 忒修斯 1.2 → 忒修斯 2.0
- 如果是更大的变化，我们就不能只改版本号了
 - 忒修斯 2.0 → 方舟 1.0

数字/数据的存储方式

- 真正复杂的问题是：数字世界里的“那些东西”，我们是如何存储的？
 - 内存/外存/分布式存储
 - 文件/文件系统/数据库/分布式数据库
- 计算机如何读写这些数据？人类如何读写这些数据？
 - 格式问题
- 如何记录这些变化？如何展示这些变化？如何追查这些变化？
 - 规范问题



版本的概念



发言稿.docx

发言稿20240909.docx

发言稿20240909-2.docx

发言稿20240910最终版.docx

发言稿20240911最终版-2.docx

发言稿20240911最终版-final.docx

01

文件的版本：一个文件，发生了哪些变化，在什么时间发生的？我们往往需要保留一个文件的多个版本。

02

文件库的版本：一次变化，涉及到多个文件，如何统一记录？

03

版本需要记录一些什么？

- 谁干的？
- 什么时间？
- 干了啥？
- 为啥？

02

从产品经理的角度，
来看版本管理



一个版本管理工具的诞生

- 设计一个隐藏目录，放那些过去的版本
- Ctrl+S的时，自动触发一个版本保存？
 - 这样会因为无意识的多次Ctrl+S，保存了太多版本
- 还是用手动方式来提交吧
 - `vc commit`
- 这个工具，在只有一个文件的时候，应该可以工作良好
- 但是，如果在这个文件夹里，有多个需要管理版本的文件怎么办？
 - 因为某一个原因，我修改了N个文件，但是这N个文件，应该算作一个版本，用一个提交命令
 - `vc commit file1 file2 file3 file4`
 - 好累啊！



一个版本管理工具，如何管理多个文件？

- 一次提交多个文件，是一个麻烦的事情
 - Git add/Git Commit解决了这问题
- 如果这些文件不是平铺的，而是存在目录结构的，又怎么办？
 - SVN的做法：每个目录下，有一个.svn
 - Git的做法，在根目录下，有一个.git
 - 哪一种更好？
- SVN的优点：一次checkout一个目录，可以分目录管理权限
- Git的优点：数据结构更加简单，效率更高，为后续优化提供可能

多人合作时的版本管理怎么做？

- Visual Source Safe的逻辑是什么？
 - Checkin/Checkout
- SVN做了哪些改进？
 - 版本号：全局自增长ID号
 - 原子提交：要么全部成功，要么全部失败
 - 与网络传输的不稳定有关
 - 分支：两个同时进行的工作，可以独自进行
 - 分支合并，成为一个大问题
 - 存储与网络访问优化（略）



团队协作的场景下，版本管理的关键概念

- 版本的含义
 - 版本号：为一个阶段工作成果命名
 - 隐含质量属性：单双数、小数点、alpha、beta、final
 - 方便收集bug
- 围绕版本的工作
 - 区分两种代码提交（feature、bugfix）
 - 划分两种工作阶段（特性开发期、发布前准备期）
 - 第三种工作阶段：针对某个长期维护版本，不断打补丁，直至宣布EOS
- 分支的含义
 - 为并行工作，提供可能
 - 不同的版本，定义不同的分支
 - 为一组（一类）持续的工作，命名与定义
 - 批量汇集工作成果（Merge）



03

如何学习Git?

Git的诞生

- Linux之父Linus是坚定的CVS反对者，他也同样地反对SVN。2002年Linus顶着开源社区精英们的口诛笔伐，选择了一个商业版本控制系统BitKeeper作为Linux内核的代码管理工具。和CVS/SVN不同，BitKeeper是属于分布式版本控制系统。
- 2005年发生的一件事最终导致了Git的诞生。在2005年初Andrew Tridgell，即大名鼎鼎的Samba的作者，试图尝试对BitKeeper反向工程，以开发一个能与BitKeeper交互的开源工具。这激怒了BitKeeper软件的所有者BitMover公司，要求收回对Linux社区免费使用BitKeeper的授权。迫不得已，Linus选择了自己开发一个分布式版本控制工具以替代BitKeeper。以下是Git诞生大事记：
 - 2005年4月3日，开始开发Git。
 - 2005年4月6日，项目发布。
 - 2005年4月7日，Git就可以作为自身的版本控制工具了。
 - 2005年4月18日，发生第一个多分支合并。
 - 2005年4月29日，Git的性能就已经达到了Linus的预期。
 - 2005年6月16日，Linux核心2.6.12发布，那时Git已经在维护Linux核心的源代码了。



版本管理工具的演进历程

V • T • E

Version control software

[hide]

Years, where available, indicate the date of first stable release. Systems with names *in italics* are no longer maintained or have planned end-of-life dates.

Local only	Free/open-source	RCS (1982) • SCCS (1972)
	Proprietary	PVCS (1985) • <i>QVCS</i> (1991)
Client–server	Free/open-source	CVS (1986, 1990 in C) • CVSNT (1998) • QVCS Enterprise (1998) • Subversion (2000)
	Proprietary	AccuRev SCM (2002) • ClearCase (1992) • <i>CMVC</i> (1994) • Dimensions CM (1980s) • <i>DSEE</i> (1984) • Endeavor (1980s) • Integrity (2001) • Panvalet (1970s) • Perforce Helix (1995) • SCLM (1980s?) • Software Change Manager (1970s) • StarTeam (1995) • Surround SCM (2002) • Synergy (1990) • Team Concert (2008) • Team Foundation Server (2005) • Visual Studio Team Services (2014) • Vault (2003) • <i>Visual SourceSafe</i> (1994)
Distributed	Free/open-source	<i>ArX</i> (2003) • BitKeeper (2000) • <i>Codeville</i> (2005) • Darcs (2002) • <i>DCVS</i> (2002) • Fossil (2007) • Git (2005) • <i>GNU arch</i> (2001) • GNU Bazaar (2005) • Mercurial (2005) • Monotone (2003) • Veracity (2010)
	Proprietary	<i>TeamWare</i> (1990s?) • Code Co-op (1997) • Plastic SCM (2006) • Team Foundation Server (via Git) (2013) • Visual Studio Team Services (via Git) (2014)
Concepts	Baseline • Branch • Changeset • Commit • Data comparison • Delta compression • Fork (Gated commit) • Interleaved deltas • Merge • Monorepo • Repository • Tag • Trunk	

版本管理的关键概念

- 仓库 (Repository) :
 - 存储项目所有版本的数据库，包含所有文件的历史记录。仓库可以是本地的，也可以是远程的，以便团队协作。
- 提交 (Commit) :
 - 将本地更改保存到仓库中，形成一个新的版本。提交通常需要添加注释，描述更改的内容。
- 版本 (Version) /修订 (Revision) :
 - 对文件或项目的每一次更改都会创建一个新的版本，带有唯一的标识符，便于追踪和回溯。
- 分支 (Branch) :
 - 创建一个项目的并行版本，允许在不影响主线的情况下开发新功能或实验。
- 主干 (Trunk) /主线 (Mainline) :
 - 项目的主要开发线路，通常包含稳定的代码。
- 合并 (Merge) :
 - 将一个分支的更改整合到另一个分支中，常用于将开发分支的功能合并回主线。
- 标签 (Tag) :
 - 给特定的版本打上标记，通常用于标识发布版本或重要的里程碑。

版本管理的关键概念

- 集中式版本控制系统（Centralized VCS）：
 - 使用中央服务器存储所有版本历史，客户端需要与服务器通信才能获取或提交更改。例子包括Subversion（SVN）、CVS。
- 分布式版本控制系统（Distributed VCS）：
 - 每个客户端都有完整的仓库副本，包括所有历史记录，支持离线操作。例子包括Git、Mercurial。
- 工作副本（Working Copy）：
 - 开发者在本地机器上正在编辑的项目文件集。
- 冲突（Conflict）：
 - 当不同的更改影响到同一部分代码时，合并过程中会产生冲突，需要手动解决。
- 操作模型：
 - 锁-修改-解锁（Lock-Modify-Unlock）：确保一次只有一个人能修改文件，避免冲突。
 - 拷贝-修改-合并（Copy-Modify-Merge）：允许多人同时修改，通过合并解决冲突，提高协作效率。

系统学习Git的路径

- 常用Git命令手册
 - <https://www.cnblogs.com/angel88/p/8194014.html>
- 《Pro Git》最好的免费教程
 - <https://www.git-scm.com/book/en/v2>
 - <https://www.git-scm.com/book/zh/v2>
 - 深入阅读第10章：Git内部原理
- 参考手册（Reference）常读常新
 - <https://www.git-scm.com/docs>
 - 每一个命令都值得细读
 - 通过查看不同版本之间，Reference的差异，更可以理解Git的改进何在
- 《Git权威指南》蒋鑫著
 - <https://www.worldhello.net/gotgit/>
- 阅读Git的源代码，先从早期版本开始学起
 - <https://github.com/git/git>



04

开始Git实战



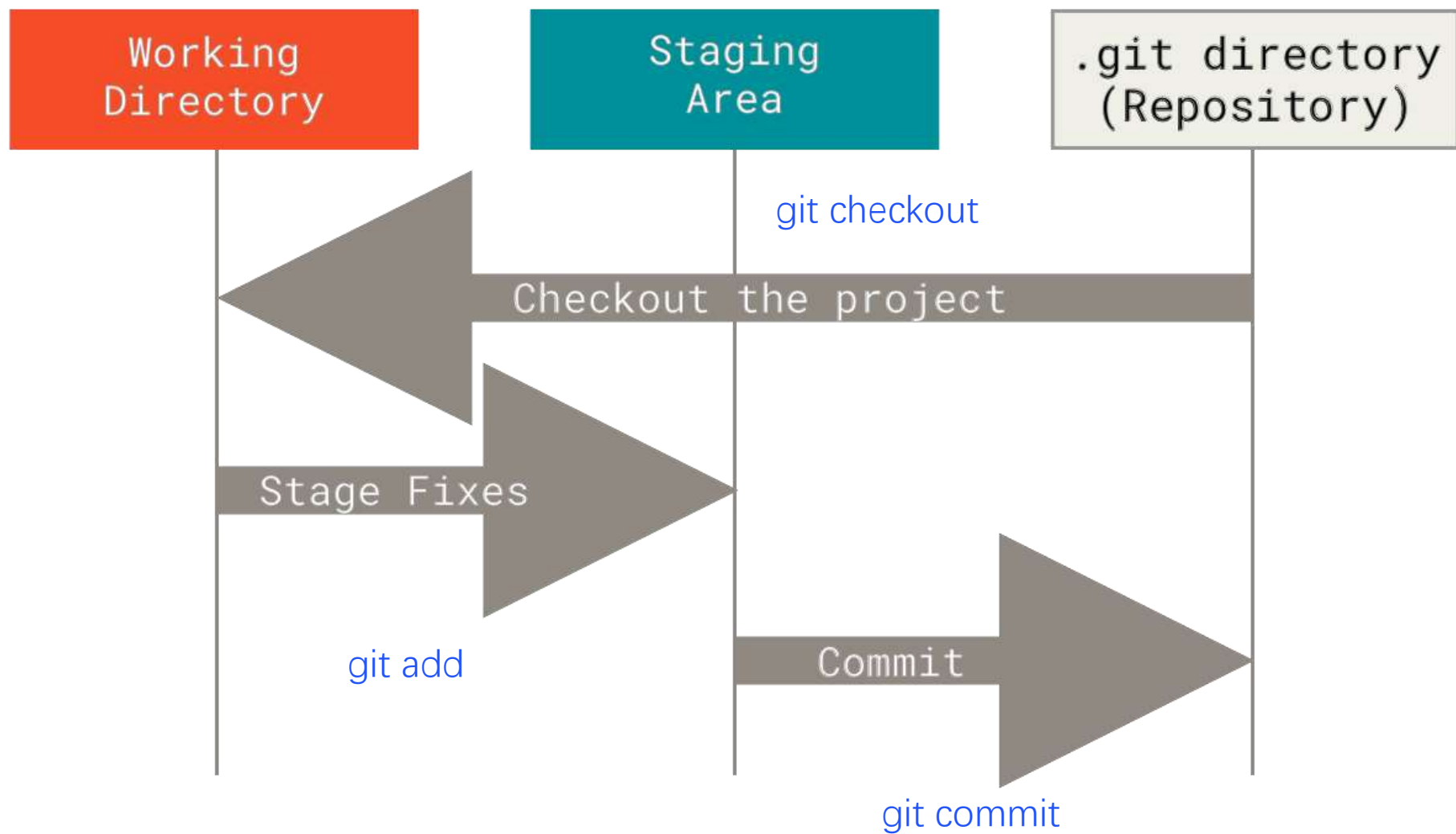
Git - 安装 Git




配置Git客户端

Git - 初次运行 Git 前的配置

Git仓库的三种状态





你的第一个Git仓库

Git - 获取 Git 仓库



记录你的每一次更新

Git - 记录每次更新到仓库



查看仓库的历史

Git - 查看提交历史



撤销操作

Git - 撤销操作



Git - 打标签



与分支相关的操作

Git - 分支简介

与Git服务器打交道

Git - 协议

Git - GitLab

<https://github.com/go-gitea/gitea>



后续的学习安排

下一次的课程：学习GitHub & Gitee

谢谢大家



主讲人：庄表伟



时间：2024.11