Git — 简明现代魔法

主讲人：方孝君

CONTENTS

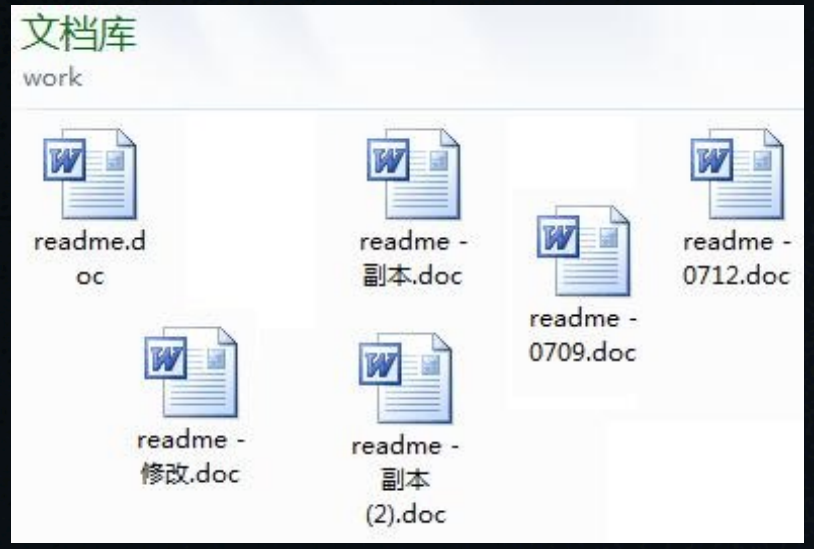
X-lab
开放实验室

Git简介

X-lab
开放实验室

01 Git简介—版本控制系统

Git 是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。



记录每次改动

进行协同工作

版本	文件名	用户	说明	日期
1	service.doc	张三	删除了软件服务条款5	7/12 10:38
2	service.doc	张三	增加了License人数限制	7/12 18:09
3	service.doc	李四	财务部门调整了合同金额	7/13 9:51
4	service.doc	张三	延长了免费升级周期	7/14 15:17

X-lab 开放实验室

01 Git的诞生

同生活中的许多伟大事物一样，Git 诞生于一个极富纷争大举创新的年代。

Linux 内核开源项目有着为数众多的参与者。 绝大多数的 Linux 内核维护工作都花在了提交补丁和保存归档的繁琐事务上（1991－2002年间）。 到 2002 年，整个项目组开始启用一个专有的分布式版本控制系统 BitKeeper 来管理和维护代码。

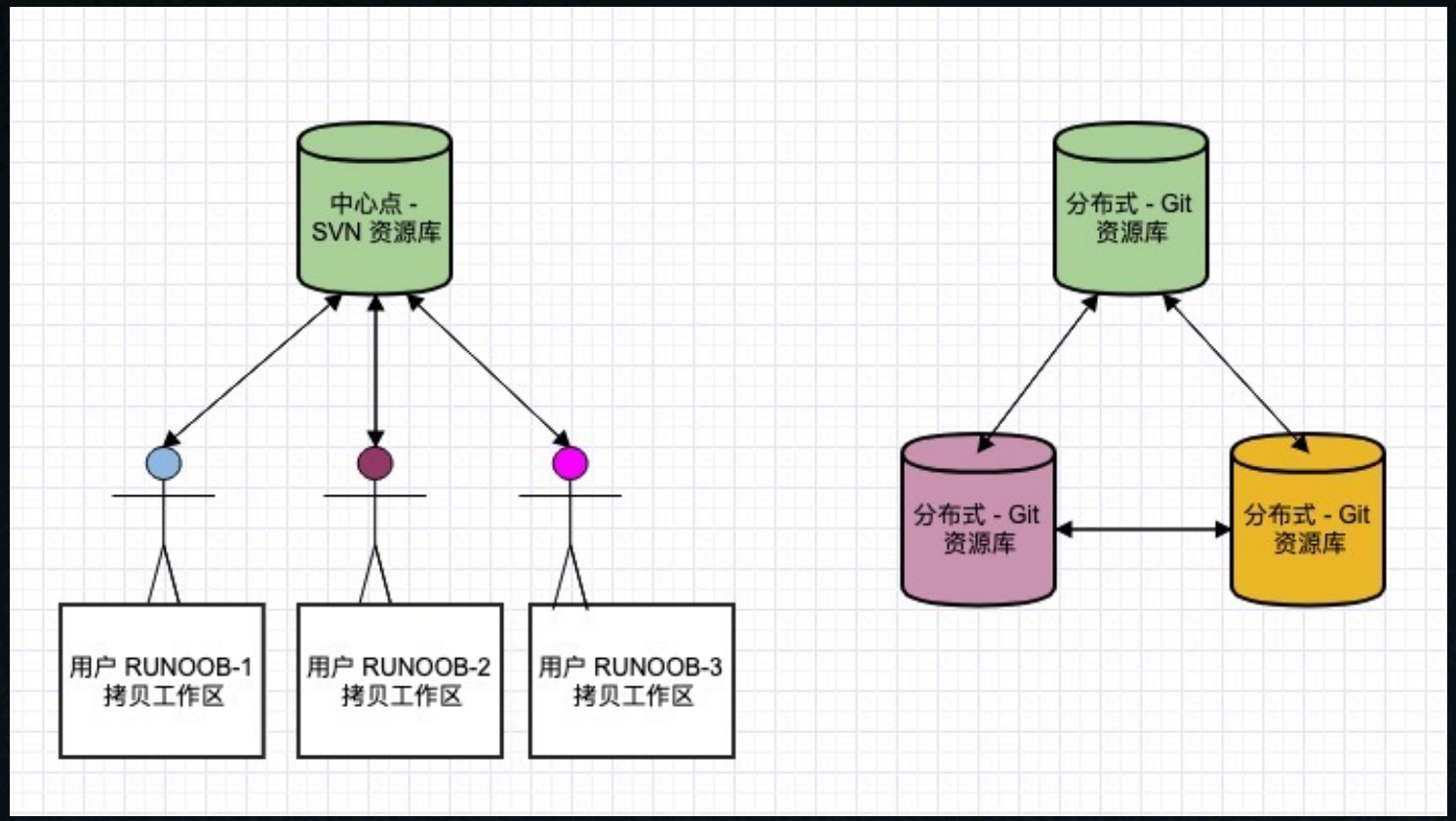
到了 2005 年，开发 BitKeeper 的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了 Linux 内核社区免费使用 BitKeeper 的权力。 这就迫使 Linux 开源社区（特别是 Linux 的缔造者 Linus Torvalds）基于使用 BitKeeper 时的经验教训，开发出自己的版本系统。 他们对新的系统制订了若干目标：

速度
简单的设计
对非线性开发模式的强力支持（允许成千上万个并行开发的分支）
完全分布式
有能力高效管理类似 Linux 内核一样的超大规模项目（速度和数据量）

自诞生于 2005 年以来，Git 日臻成熟完善，在高度易用的同时，仍然保留着初期设定的目标。 它的速度飞快，极其适合管理大项目，有着令人难以置信的非线性分支管理系统。

X-lab
开放实验室

01 Git 与 SVN 区别点



1、Git 是分布式的，SVN 是集中式

2、Git 把内容按元数据方式存储，而 SVN 是按文件

3、Git 分支和 SVN 的分支不同

4、Git 没有一个全局的版本号，而 SVN 有

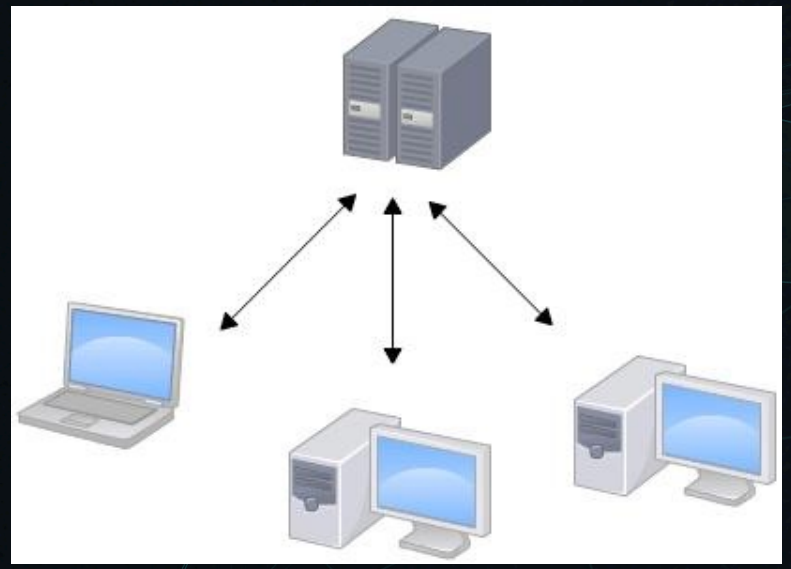
5、Git 的内容完整性要优于 SVN

X-lab
开放实验室

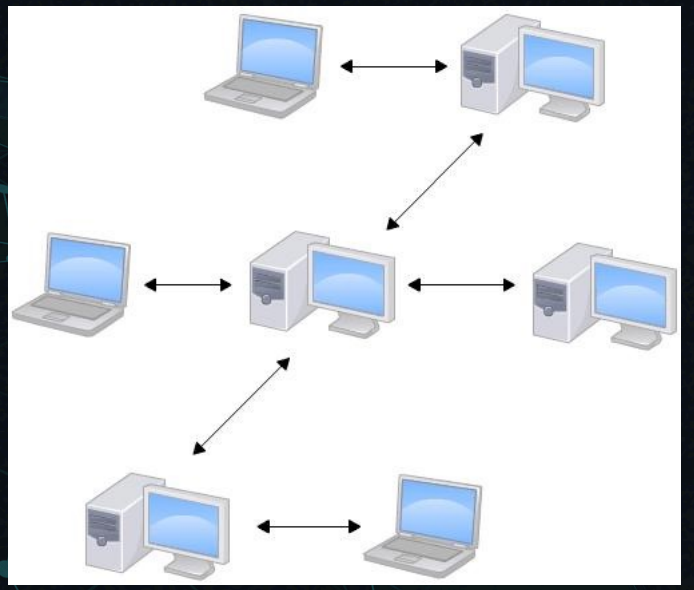
01 集中式vs分布式

CVS作为最早的开源而且免费的集中式版本控制系统，直到现在还有不少人在用。由于CVS自身设计的问题，会造成提交文件不完整，版本库莫名其妙损坏的情况。
同样是开源而且免费的SVN修正了CVS的一些稳定性问题，是目前用得最多的集中式版本库控制系统。

CVS及SVN都是集中式的版本控制系统，而Git是分布式版本控制系统，集中式和分布式版本控制系统有什么区别呢？



集中式



分布式

X-lab 开放实验室

Git目录结构

X-lab
开放实验室

01 Git目录结构

Git给自己的定义是一套内存寻址文件系统，当你在一个目录下执行git init命令时，会生成一个.git目录，它的目录结构如右图：

其中branches目录已经不再使用
description文件仅供GitWeb程序使用
config文件保存了项目的配置。

需要我们重点关注的是**HEAD**和**index**文件以及**objects**和**refs**目录。
其中index中保存了暂存区的一些信息

```
.git/
├── branches
├── config
├── description
├── HEAD
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── prepare-commit-msg.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

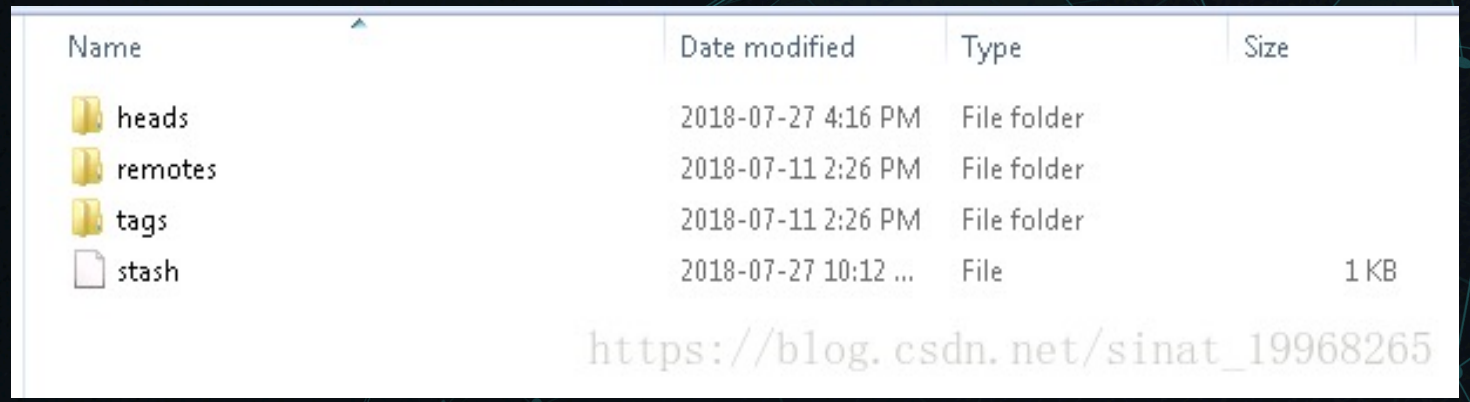
02 Git目录结构

Objects

存储对象的目录,本地仓库，git中对象分为三种：commit对象，tree对象(多叉树)，blob对象；

Refs

存储指向branch的最近一次commit对象的指针，也就是commit对象的sha-1值（就是hash值，sha-1是一种散列算法），refs的目录下包括以下目录(git init后并没有remotes和stash,需要有从remote地址中pull code等交互性操作才会出现remotes目录,stash文件则是做了stash操作才会出现)：



heads是存储本地仓库每一个分支最近一次commit对象的sha-1值；

remotes则是存储最近一次push到远程仓库的commit对象的sha-1值；

Tags目录下的文件存储的是标签对应的commit对象的sha-1值；

stash存储藏匿处(就是GitExtentsions中的stash操作)的对象sha-1值;

X-lab 开放实验室

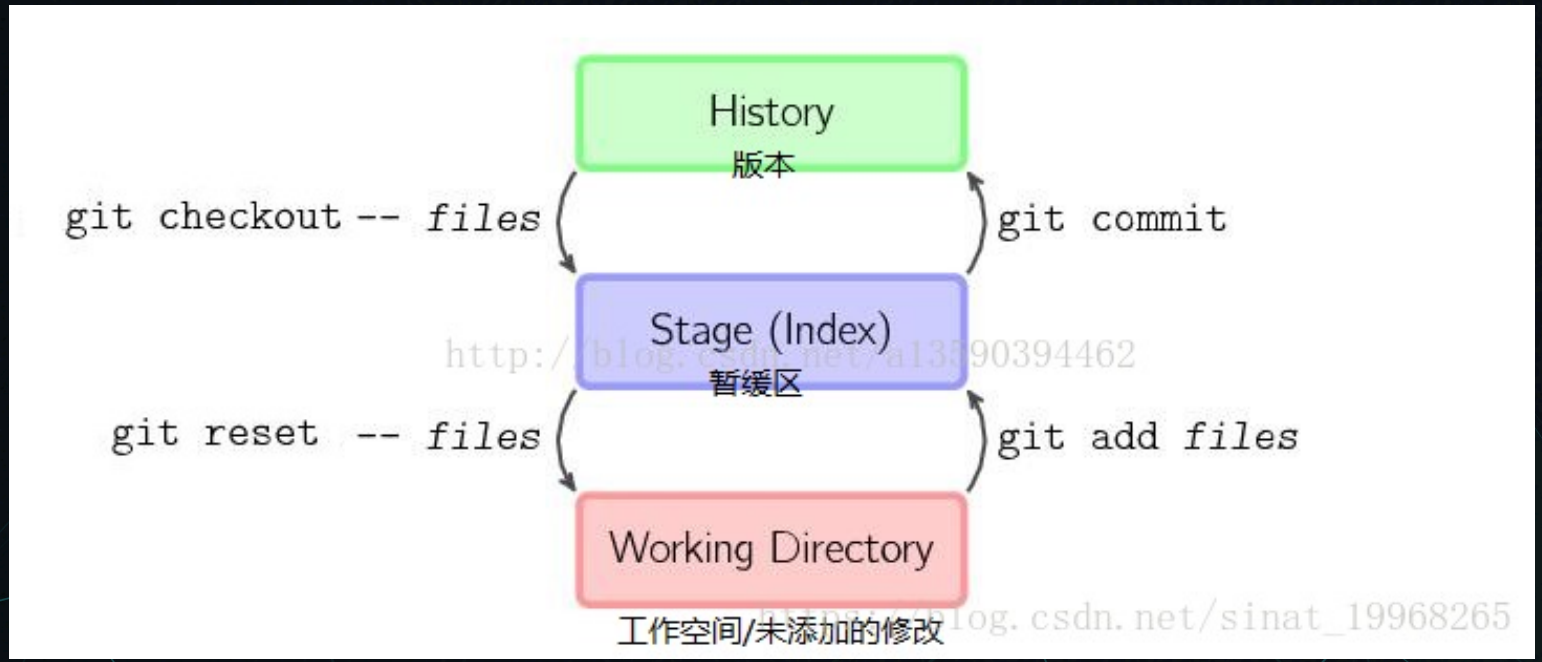
02 Git目录结构

HEAD文件

该文件表示当前本地签出的分支，例如存储的值：ref: refs/heads/master;

Index文件

存储缓冲区(GitExtensions中的stage)的内容，内容包括它指向的文件的时间戳、文件名、sha1值等
(git三大区域:工作区，缓冲区，历史记录区，如下图)

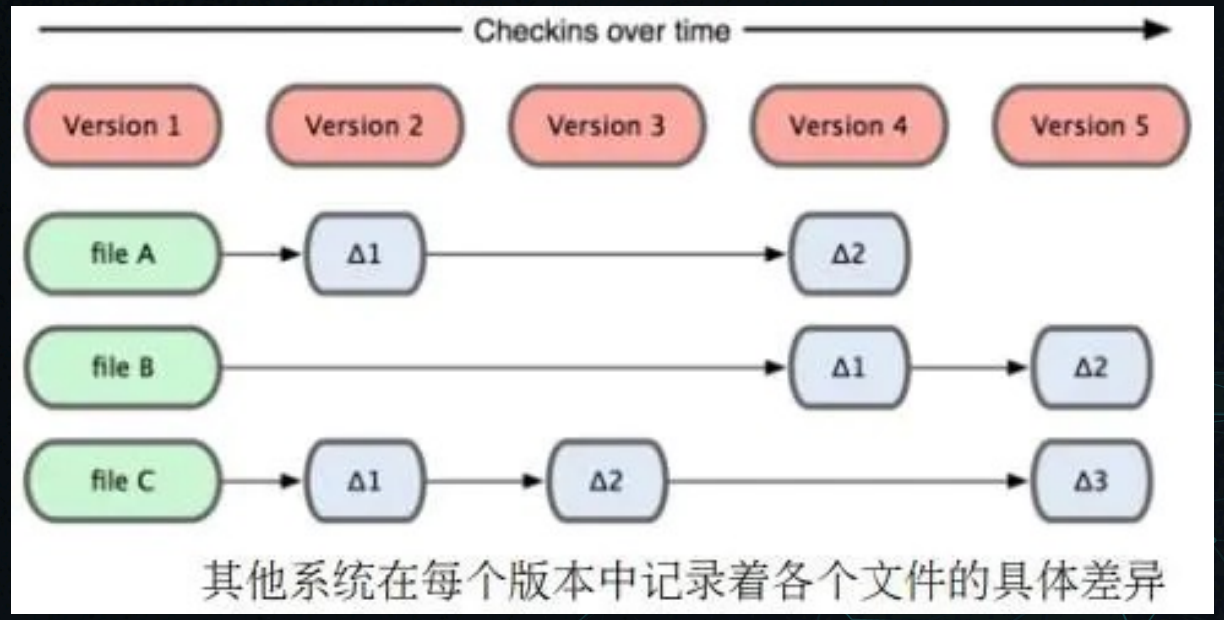


X-lab 开放实验室

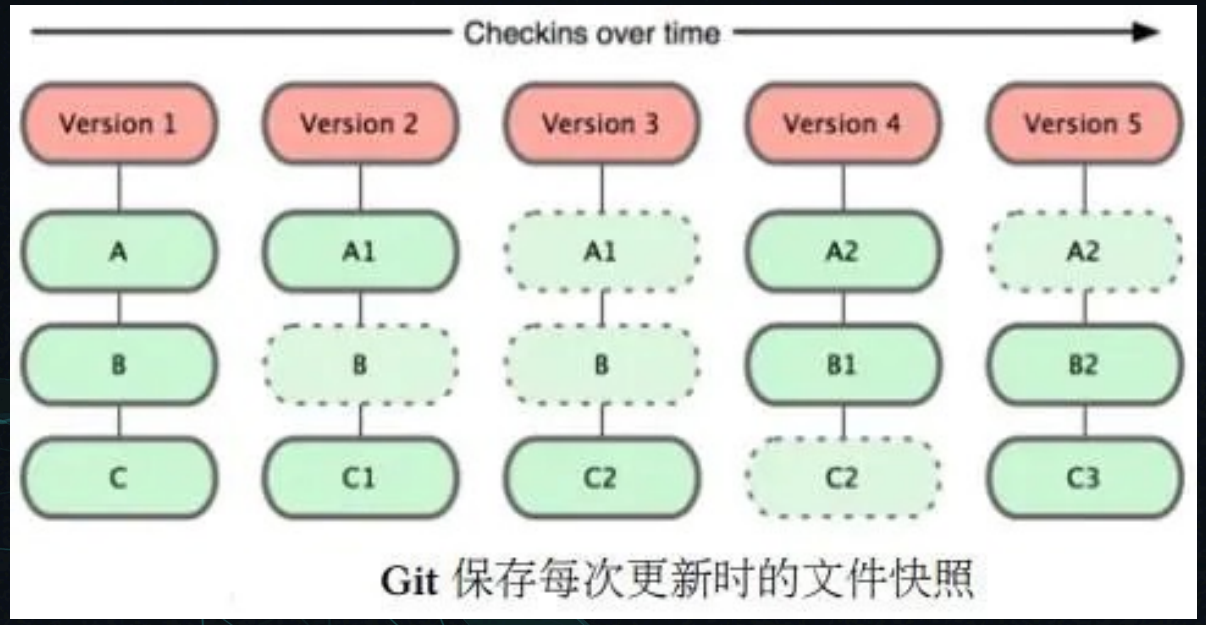
Git工作原理

X-lab
开放实验室

03 直接记录快照，而非差异比较



其他系统在每个版本中记录着各个文件的具体差异

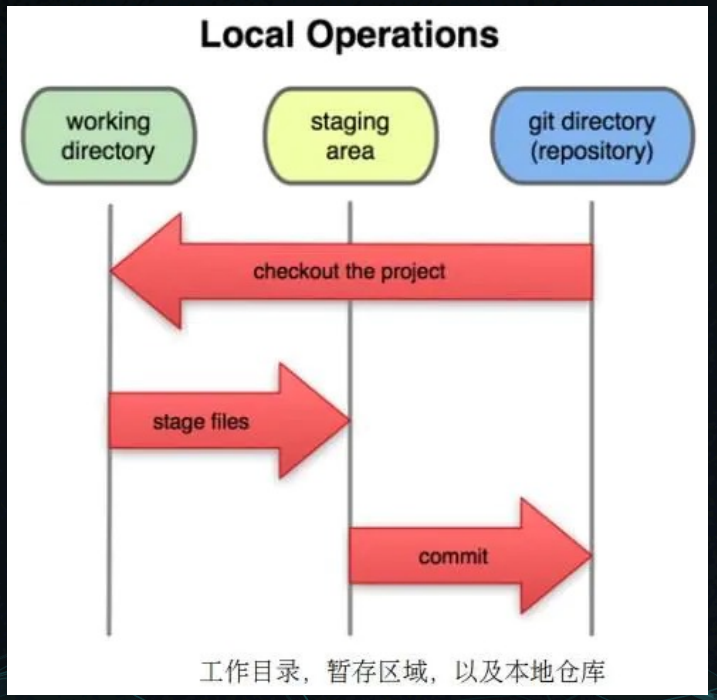


Git 保存每次更新时的文件快照

Git 并不保存这些前后变化的差异数据。实际上，Git 更像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。

X-lab 开放实验室

03 文件的三种状态

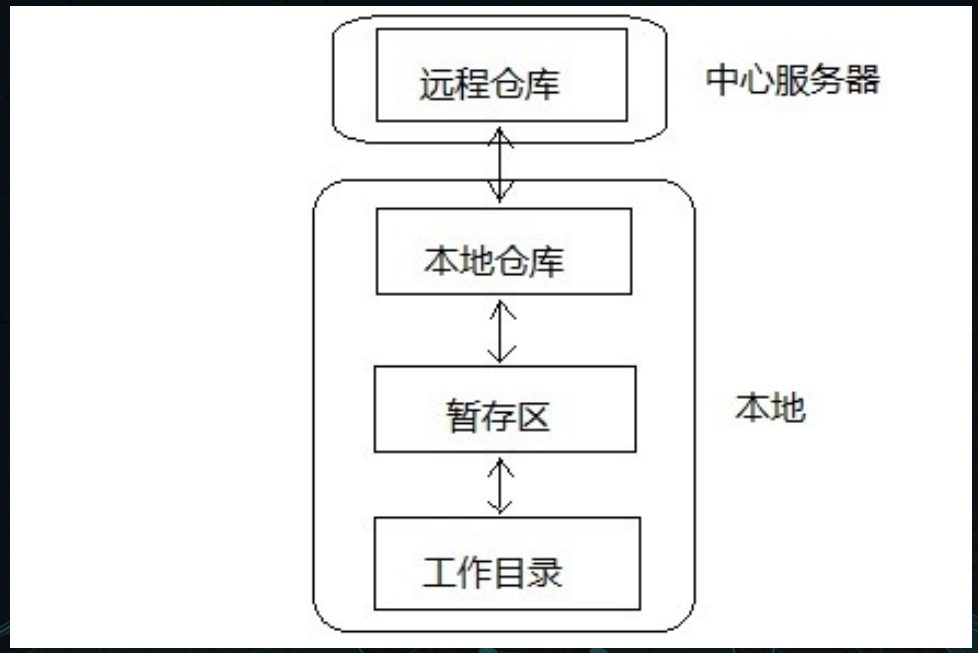


工作目录，暂存区域，以及本地仓库

对于任何一个文件，在 Git 内都只有三种状态: 已提交(committed)，已修改(modified)和已暂存(staged)。
已提交表示该文件已经被安全地保存在本地数据库中了；
已修改表示修改了某个文件，但还没有提交保存；
已暂存表示把已修改的文件放在下次提交时要保存的清单中。

X-lab 开放实验室

03 Git的分层结构



由此我们看到 Git 管理项目时，文件流转的三个工作区域：
Git的工作目录(working directory)，暂存区域(staging area)，以及本地仓库(repository)。

X-lab
开放实验室

03 Git工作原理

近乎所有操作都是本地执行

在 Git 中的绝大多数操作都只需要访问本地文件和资源，不用连网。因为 Git 在本地磁盘上就保存着所有当前项目的历史更新，所以处理速度飞快。

时刻保持数据完整性

在保存到 Git 之前，所有数据都要进行内容的校验和(checksum)计算，并将此结果作为数据的唯一标识和索引。这项特性作为 Git的设计哲学，建在整体架构的最底层。所以如果文件在传输时变得不完整，或者磁盘损坏导致文件数据缺失，Git 都能立即察觉。

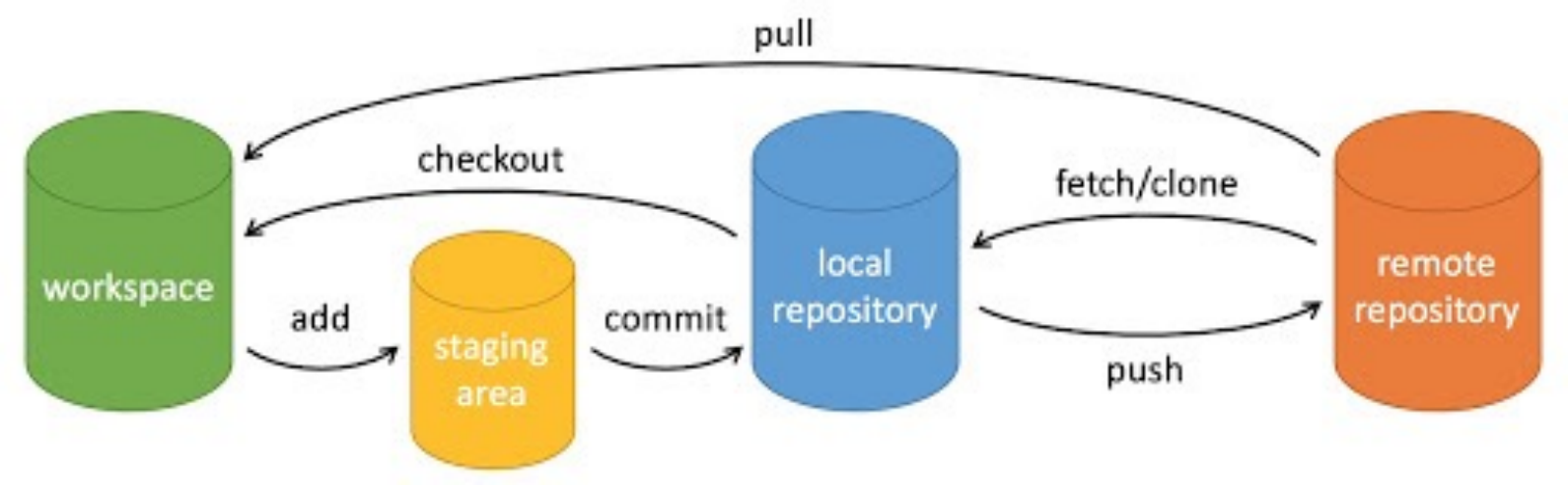
多数操作仅添加数据

你执行的 Git 操作，几乎只往 Git 数据库中增加数据。 很难让 Git 执行任何不可逆操作，或者让它以任何方式清除数据。

Git基本操作

X-lab
开放实验室

04 Git基本操作



workspace：工作区
staging area：暂存区/缓存区
local repository：版本库或本地仓库
remote repository：远程仓库

创建仓库命令：

git init	初始化仓库
git clone	拷贝一份远程仓库，也就是下载一个项目

X-lab 开放实验室

04 Git基本操作

提交与修改：

git add	添加文件到仓库
git status	查看仓库当前的状态，显示有变更的文件。
git diff	比较文件的不同，即暂存区和工作区的差异。
git commit	提交暂存区到本地仓库。
git reset	回退版本。
git rm	删除工作区文件。
git mv	移动或重命名工作区文件。

04 Git基本操作

提交日志

git log	查看历史提交记录
git blame <file>	以列表形式查看指定文件的历史修改记录

远程操作

git remote	远程仓库操作
git fetch	从远程获取代码库
git pull	下载远程代码并合并
git push	上传远程代码并合并

X-lab
开放实验室

THANK YOU

方孝君