1. Git介绍

- 官方网站
- Pro Git书籍地址
- 博客：Git十周年，Linus专访

1.1 Git是什么

Git是一个免费开源的分布式版本控制系统，可用于高效的处理任何大大小小的项目；

1.2 Git出现的背景

1991年Linus开发了Linux系统，随后将源代码开放，此后Linux系统不断发展；

Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的开发者，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？

在2002年以前，Linux代码由Linus本人通过手工方式合并代码！

Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，与Linux的开源精神不符。

到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，开源社区的精英们也对这种方式表达了强烈不满，于是Linus选择了一个商业的**分布式版本控制系统BitKeeper**，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux开源社区免费使用这个版本控制系统。

在2005年，Samba文件服务器开发人Andrew Tridgell试图破解BitKeeper的协议，被BitMover公司发现了，于是BitMover公司想让Linus公开发邮件道歉，保证以后严格约束社区开发者，否则，收回Linux开源社区的免费使用权。Linus迫不得已，自己开发了一个分布式版本控制工具代替了BitKeeper。

Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub。

Linus在Git十周年专访中的对Git的说明

我大概花一天让Git能达到自己管理自己的程度（self-hosting），之后我就开始用Git提交程序代码了。我的大部分工作是在白天完成的，不过也有几天工作到深夜。我觉得最有趣的地方在看到Git如何快速的成形。在Git的第一次提交并没有写很多程序，但是已经实现作出提交程序代码的基本功能。写Git并不会很难，比较难的是思考如何Git组织档案的方式。

我想强调，Git从无到有大概花了我十天（包含我第一次用Git提交核心程序代码），而且我也不是焚膏继晷的完成Git。这都取决于对Git的基本概念是否很清楚，早在着手写Git前，我已经看到其他源代码控制系统的缺陷。我只是不想重蹈覆辙。

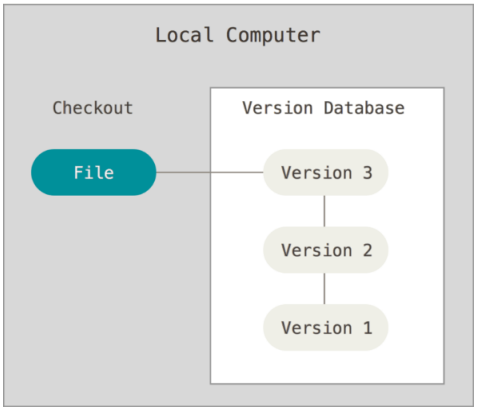
2. 版本控制系统的演化

2.1 本地版本控制

许多人习惯用复制整个项目目录的方式来保存不同的版本，或许还会改名加上备份时间以示区别。这么做唯一的好处就是简单，但是特别容易犯错，有时候会混淆所在的工作目录，一不小心会写错文件或者覆盖意想外的文件。

为了解决这个问题，人们很久以前就开发了许多种本地版本控制系统，大多都是采用某种简单的数据库来记录文件的历次更新差异。

其中最流行的一种叫做 RCS，现今许多计算机系统上都还看得到它的踪影。 RCS 的工作原理是在硬盘上保存补丁集（补丁是指文件修订前后的变化）；通过应用所有的补丁，可以重新计算出各个版本的文件内容。

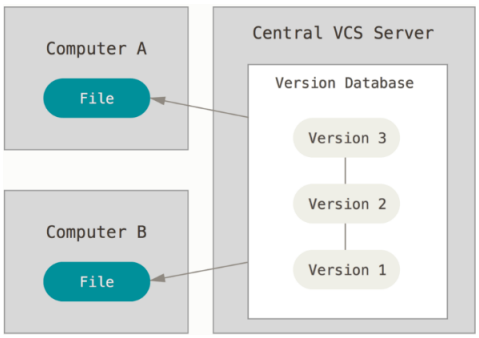


2.2 集中式版本控制

接下来人们又遇到一个问题，如何让在不同系统上的开发者协同工作？ 于是，集中化的版本控制系统（Centralized Version Control Systems，简称 CVCS）应运而生。 这类系统，诸如CVS、**Subversion**以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。 多年以来，这已成为版本控制系统的标准做法。

这种做法带来了许多好处，特别是相较于老式的本地 VCS 来说。 现在，每个人都可以在一定程度上看到项目中的其他人正在做些什么。 而管理员也可以轻松掌控每个开发者的权限，并且管理一个 CVCS 要远比在各个客户端上维护本地数据库来得轻松容易。

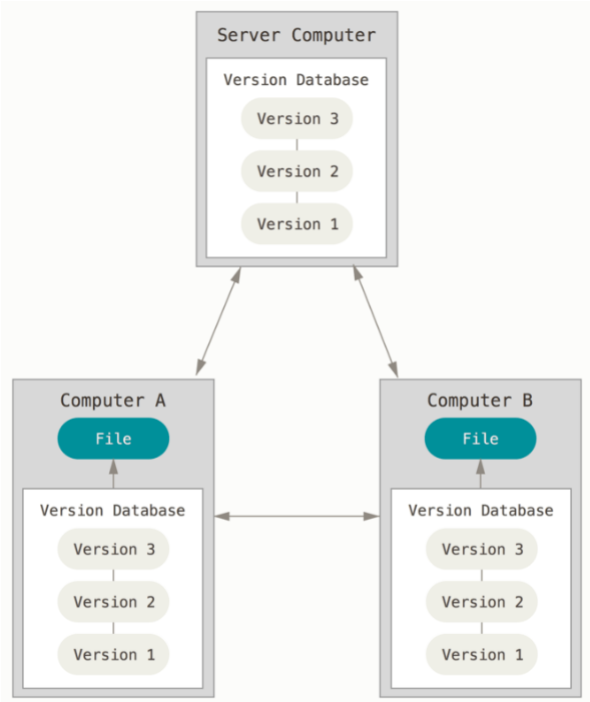
这么做最明显的缺点是**中央服务器的单点故障**， 如果宕机一小时，那么在这一小时内，谁都无法提交更新，也就无法协同工作。 如果中心数据库所在的磁盘发生损坏，又没有做恰当备份，毫无疑问你将丢失所有数据——包括项目的整个变更历史，只剩下人们在各自机器上保留的单独快照。 本地版本控制系统也存在类似问题，只要整个项目的历史记录被保存在单一位置，就有丢失所有历史更新记录的风险。



2.3 分布式版本控制

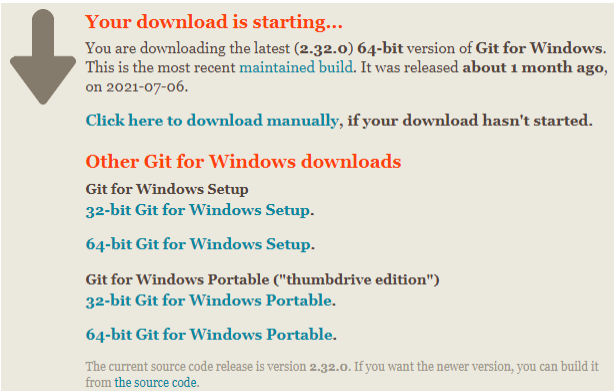
于是分布式版本控制系统（Distributed Version Control System，简称 DVCS）面世了。 在这类系统中，像 Git、Mercurial、Bazaar 以及 Darcs 等，客户端并不只提取最新版本的文件快照， 而是把代码仓库完整地镜像下来，包括完整的历史记录。 这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。 因为每一次的克隆操作，实际上都是一次对代码仓库的完整备份。

许多这类系统都可以指定和若干不同的远端代码仓库进行交互。这样就可以在同一个项目中，分别和不同工作小组的人相互协作。 你可以根据需要设定不同的协作流程，比如层次模型式的工作流，而这在以前的集中式系统中是无法实现的。



3. Git安装 - Windows版

- 下载地址：地址



4. Git的三大区域

1. 工作区
2. 暂存区（index、stage）

3. 版本库

   本地版本库

   远程版本库

5. 掌握几个Linux命令

1、**cd** - 切换目录change directory

- **绝对路径**

  Linux、Mac OS：以根目录 / 开始的文件名或目录名称，如/opt、/home

  Windows：C:/windows、D:/java

- **相对路径**

  相对于当前路径的文件名写法。

  Linux、Mac OS：如./lib/sudo/ 或 ../lib/sudo/等，可以认为开头不是/就是相对路径；

  Windows：./windows

- **4个特殊目录**

  .（或./）：当前目录

  ..（或../）：上一级目录

  ~：用户主目录(用户根目录), C:/users/用户名/

  -：上一次所在目录

2、**pwd** - 显示当前所在目录

3、**ls** - 显示某个目录下的内容

- ls -l：显示详细信息
- ls -a：显示所有文件（包括隐藏文件）

4、**cat** - 查看文件内容

5、**touch** - 创建文件

6、**echo +** **>> 或 >**

   echo "文件内容" >> 文件

   echo "文件内容" > 文件

   > 和 >>区别 ：

   >：覆盖原有内容

   >>：在原有内容上追加内容

7、**mkdir** - 创建目录

   参数-p：递归创建子目录

```
mkdir -p test/test2/test3
```

8、**rm** - 删除文件

```
rm -rf 文件\目录
```

9、**mv** - 重命名\移动文件、目录

mv 原文件名 新文件名

mv 原文件(目录) 指定的目录