版本控制工具应该具备的功能：

协同修改：多人协作修改服务器的同一个文件

数据备份：不仅保存目录和文件的当前状态，还能保存每一个提交过的历史状态

版本管理：保存每一个版本的文件信息但不保存重复数据以节约存储空间提高运行效率，SVN采用的是增量式管理的方法，而Git采用的是系统快照的方式

权限控制：对团队中参与开发的人员进行权限控制，对团队外开发者贡献的代码进行审核（Git独有）

历史记录：查看修改人、修改时间、修改内容、日志信息等，恢复本地文件到某一个历史状态

分支管理允许开发团队在工作过程中多条生产线同时推进任务，进一步提高效率

版本控制是一种思想：工程设计领域中使用版本控制管理工程蓝图的设计过程，在IT开发过程中使用版本控制思想管理代码的版本迭代

实现版本控制思想的版本控制工具分为集中式版本控制工具（SVN、CVS、VSS等）和分布式版本控制工具（Git、Mercurial、Bazaar、Darcs等）

集中式版本控制工具所有历史状态存在于集中的服务上，本地只保存当前状态，存在单点故障问题，分布式版本控制工具所有历史状态存在于所有参与者机器上

Git是由Linux的开发者Linus使用C语言开发分布式版本控制系统，Linux系统的开发最初1991年是由Linus手动合并代码，2002年BitMover公司将版本控制系统的商业软件BitKeeper免费授权于Linux社区使用，由于开发Samba的Andrew试图破解BitKeeper的协议，BitMover公司收回了Linux社区BitKeeper的免费使用权，2005年Linus本人使用C语言开发了分布式版本控制系统Git，2008年代码托管中心GitHub上线（即开源）

Git的官方网址：<https://git-scm.com/>

Git的优势：本地保存所有历史操作状态，不需要联网可以实现版本控制管理，完整性保证（每次提交的状态会有相应的Hash值与之对应），尽可能添加数据而不是删除或修改数据（尽可能保存每一个状态，可以恢复每一个状态），分支操作非常快捷流畅（以快照方式管理文件，通过指针管理快照实现分支），与Linux命令全面兼容

Git工作区（新建修改文件的区域）的文件需要使用git add命令添加到暂存区（临时存储文件区域），然后使用git commit命令提交到本地库（历史版本管理区域）

Git的托管中心是维护本地外的远程库，Git的托管中心局域网环境下可以搭建GitLab服务器，外网环境有官方搭建好的GitHub（国际的托管中心）和码云（中国的托管中心）

团队内部协同开发：项目创始人通过push命令推送Git本地库的文件到GitHub托管中心的远程库进行管理，项目其他人需要加入团队，获得相应权限后可以通过clone命令克隆复制GitHub远程库中的文件，修改完成后可以push推送到GitHub远程库中，已经拥有项目本地库的人可以使用pull命令拉取文件的修改内容到本地库中

跨团队协同开发：项目创始人通过push命令推送Git本地库的文件到GitHub托管中心的远程库进行管理，项目外的人可以使用fork命令复制项目的分支远程库，使用clone命令克隆复制项目的分支远程库到项目外的人的本地库中，修改后使用push命令推送到分支远程库中，分支远程库的创始人可以使用pull request命令发送拉取请求到项目远程库，项目远程库创始人可以审核请求，允许请求可以使用merge命令合并分支远程库的修改内容到项目远程库中

Git初始化：在指定的文件夹下使用git init命令会在该目录下创建.git目录，.git目录下存放的是本地库相关的文件或目录，Git的配置在.git目录下的config文件中，Git初始化完成后需要设置签名（用户名和email地址）即设置本地库的身份用于区分其他本地库，本地库的签名与登录远程库（托管中心）的账号、用户名、密码、email无关联性，仅用于区分提交文件的本地库的身份

Git的签名分为仓库/项目级别和系统用户级别，级别选择采用就近原则即仓库/项目级别优先于系统用户级别，两者签名至少拥有一种，无任何签名Git会无法识别身份拒绝部分命令请求

git config [--global] user.name/user.email 用户名/邮箱 设置[系统用户级别，默认仓库/项目级别]签名的用户名/邮箱为指定的用户名/邮箱

Git的仓库/项目级别签名设置信息保存在本地库.git/config文件中即本地库的配置文件

Git的系统用户级别签名设置信息保存在系统用户主目录/.gitconfig文件中

git status 查看工作区和暂存区的状态

git add 文件路径名 添加指定文件到暂存区

git commit [-m "提交的记录信息"] [-a]文件路径名提交指定文件到本地库管理，[直接添加提交记录信息忽略进入提交信息记录界面][在暂存区的文件被修改后直接提交]提交文件命令执行后会进入提交信息记录界面

git rm --cached 文件路径名 从暂存区删除指定文件即暂存区取消管理并非工作区删除文件

git reset HEAD 文件路径名 从暂存区取消指定文件的修改

git log [--pretty=oneline] 查看完整版本信息的历史记录[以每个版本一行进行显示]，如果显示数据过多，多屏显示的方式与Linux的less命令一致

git log [--oneline] 查看版本历史记录[以每个版本一行省略显示]

git reflog 查看带有版本指向指针移动步数的所有版本记录

git reset --hard/mixed/soft 版本索引值（索引值的部分值即可） 版本指针指向指定版本即当前版本转换成指定版本，本地库、暂存区和工作区版本指针同时移动/本地库和暂存区版本指针移动工作区版本指针不移动（即工作区文件状态不随版本本地库指针移动而变化）/本地库版本指针移动暂存区和工作区版本指针不移动（即暂存区和工作区文件状态不随本地库版本指针移动而变化），暂存区和工作区文件的状态是相对于本地库的版本的状态，不是相对于文件本身变化的状态

git reset --hard HEAD 修改暂存区和工作区版本指针状态（即文件状态）与本地库一致

git reset --hard HEAD^[^] 向后回退^个数个版本，只能进行回退

git reset --hard HEAD~数值 向后回退指定数值个版本，只能进行回退

git help 命令 查看指定命令本地文件中的帮助文档

git diff [HEAD/HEAD^/HEAD~数值] [文件路径名] 显示工作区[指定文件，默认所有文件]与暂存区[指定版本的本地库版本]对应文件的状态比较，Git比较的基本单位是行，修改一行内容相当于删除(-)并新加(+)一行内容，Git显示的比较差异是从工作区状态到达指定对比状态需要进行的操作

Git分支可以同时并行推进多个分支任务前进，提交开发效率，各个分支直接相互独立不影响

git branch -v 查看所有分支

git branch 分支名称 创建新的指定名称的分支

git checkout -- 文件路径名 在工作区取消修改

git checkout 分支名称 切换工作区到指定分支名称的分支上

git merge 分支名称 将指定分支合并到当前所在分支上即按照指定分支的修改内容更新当前分支的内容，Git合并两个分支相同文件相同位置不同修改时会产生冲突需要手动修改，Git的冲突不会产生新的文件但会在冲突文件位置处显示冲突（SVN冲突会产生新的文件），冲突位置显示从“<<<<<<< HEAD”到“=======”为当前分支内容，从“=======”到“>>>>>>> 分支名称”为合并分支的另一个分支内容

解决冲突：编辑冲突文件删除标识冲突的特殊符号即“</=/>”，修改文件内容为所需要状态，使用git add 冲突文件名命令标记文件冲突已经解决，然后使用git commit -m "日志信息"命令解决冲突

哈希是一个系列的加密算法，各个不同的哈希算法虽然加密强度不同但具有几个共同点：无论输入数据的数据量有多大，使用同一个哈希算法得到加密结果长度固定，哈希算法和输入数据确定，输出数据结果保证不变，哈希算法确定但输入数据有变化，输出数据结果一定有变化通常是很大的变化，哈希算法不可逆

Git底层采用的是SHA-1算法

集中式版本控制工具的文件管理机制：以文件变更列表的方式存储信息，系统将保存的信息看作为一组基本文件和每个文件随时间逐步累积的的差异

Git的文件管理机制：Git把书看作为小型文件系统的一组快照，每次提交更新是Git都会对当前的全部文件制作一个快照并保存这个快照的索引，为了高效如果文件没有修改，Git不再重新存储文件，而是只保留一个链接指向之前存储的文件，所以Git的工作方式可以成为快照流

Git的所有文件以及对象都会通过哈希算法对应相应的哈希值，所有文件对应各自的哈希值，Git的tree对象记录所有文件的个数以及每个文件对应的名称和哈希值，tree对象同样有自己的哈希值，Git的commit对象拥有自己的哈希值（git log命令中commit后对应的哈希值）并且包含tree对象的哈希值等信息以及上一个提交对象的哈希值信息等，每个提交对象对应一次commit命令即对应Git的一次版本快照

区块链的比特币管理交易类似Git管理文件的机制：比特币的每一次交易类似Git中的一个文件，所有比特币的交易都有自己的哈希值，并且两两做哈希算法，直到归并为一个哈希值，然后和上一个区块的哈希值以及下一个区块的哈希值构成一个区块，所有区块连在一起构成数据区块链条，数据区块链条产生后其中的任何一个数据不能发生变化，修改任何数据整个数据区块链条的哈希值全部不能校验通过

Git创建分支原理是创建一个指针指向版本快照，初始分支也是一个指针（master），当前分支指针是HEAD，切换分支只是移动HEAD指针指向不同的快照版本

git remote -v 查看远程地址别名信息

git remote add 别名 远程地址 添加远程地址的别名为指定别名

git push 远程地址或远程地址别名 分支 推送指定分支的本地库内容到指定远程地址或远程地址别名的远程库

git clone 远程仓库地址 克隆复制指定远程仓库地址的远程仓库内容到当前本地仓库，克隆远程仓库到本地仓库包含克隆远程仓库的所有文件和远程仓库记录的所有远程仓库地址别名以及初始化当前本地仓库

Windows系统使用Git登录GitHub账号时会被记住账号和密码，需要修改登录的账号和密码时可以在控制面板\所有控制面板项\凭据管理器Windows凭据中找到GitHub的凭据进行删除和修改

git fetch 远程仓库地址或别名 分支 从指定远程仓库地址或别名的远程仓库的指定分支的内容拉取到本地仓库但不更新本地仓库的工作区

git checkout 远程仓库地址或别名/分支名 切换到已经拉取到本地仓库的指定远程仓库地址或别名的指定分支名的本地暂存远程仓库，在本地暂存远程仓库中可以查看已经拉取到本地仓库的远程仓库内容

git merge 远程仓库地址或别名/分支名 将已经拉取到本地仓库的远程仓库内容合并到当前仓库中

git pull 远程地址或远程地址别名 分支 拉取指定远程地址或远程地址别名指定分支的远程仓库内容到本地仓库并更新本地仓库，命令等同fetch和merge操作

不是基于GitHub远程库最新版内容进行修改不能进行推送，需要拉取最新版内容进行修改才能提交，来取合并产生冲突解决方法与分支合并冲突解决方法一致

跨团队协作：没有被邀请加入团队的用户需要在GitHub上对指定项目进行fork复制远程仓库操作后，拉取fork后的远程仓库到本地修改完成后再推送到fork的远程仓库中，然后在GitHub上fork后的远程仓库依次进行pull requests、new pull request、create pull request，项目创建者需要在GitHub上的项目远程仓库依次进行pull requests、查看request内容、conversation查看会话/commit查看提交信息/files changed查看文件修改信息、merge pull request合并请求

GitHub设置SSH免密码登录（HTTPS的免密码登录是由于Windows凭据管理器的原因）：

使用cd ~命令进入用户家目录，使用ssh-keygen -t rsa -C GitHub账号邮箱生成指定账号邮箱的ssh密钥，将~/.ssh/id\_rsa.pub的文件内容复制到GitHub官网上账号、Settings、SSH and GPG keys中，可以将远程登录的SSH地址加入到远程地址别名中方便使用

Eclipse管理工程的文件.classpath文件、.project文件和.settings目录下所有文件，Git一般将开发工具管理工程的文件忽略，指定Git不对开发工具管理工程的文件进行管理

Git官方忽略文件格式配置文档地址<https://github.com/github/gitignore>

在~/.gitconfig文件中加入:

[core]

excludesfile=忽略文件格式的配置文档的全路径名

Git的工作流工作方式分为集中式工作流、GitFlow流和Forking工作流

GitLab服务器安装按照官方文档安装完成后使用命令gitlab-ctl reconfigure命令初始化配置GitLab，使用gitlab-ctl start/stop 启动停止GitLab服务