## 版本控制系统 version control system

**版本控制**:是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统，我们可以对任何类型的文件进行版本控制。  
优点在于：不论我们怎么更改，我们都可以轻松返回之前的版本，而工作量却非常小。

**三种版本控制系统**

1. 基于文件系统：现在已经淘汰，使用具有文件共享功能的中央服务器，这样的控制系统为当前目录创建了跟踪文件，并使得仅在单个文件系统中存储和跟踪版本成为可能
2. 集中式(Centralized)：具有单一存储库(repository) - 服务器上的项目源代码存储库，以及使用特殊协议的客户端远程访问
3. 分布式(Distributed)： 它有一个中央存储库，用户可以从中将数据下载到他们的本地存储库。 经过几个阶段的本地检查后，数据返回到中央存储库。 Git 是一个分布式系统

**Git：**

分布式工作流程：1.集中式工作流 2.集成管理者工作流 3.司令官与副官工作流

工作区：本地电脑存放项目文件的地方，比如文件夹；

暂存区（Index/Stage）：在使用git管理项目文件的时候，其本地的项目文件会多出一个.git的文件夹，将这个.git文件夹称之为版本库。其中.git文件夹中包含了两个部分，一个是暂存区（Index或者Stage）,顾名思义就是暂时存放文件的地方，通常使用add命令将工作区的文件添加到暂存区里；

本地仓库：.git文件夹里还包括git自动创建的master分支，并且将HEAD指针指向master分支。使用commit命令可以将暂存区中的文件添加到本地仓库中；

远程仓库：不是在本地仓库中，项目代码在远程git服务器上，比如项目放在github上，就是一个远程仓库，通常使用clone命令将远程仓库拷贝到本地仓库中，开发后push到远程仓库中即可；

使用clone命令，可以将远程仓库复制到本地仓库，使用init命令，可以创建本地仓库。

开发者可以使用 branch 创建一个新的分支，当前正在工作的分支作为父分支。 checkout 命令用于将工作副本设置为与所需分支一致的状态（即，检出到该分支）。 创建分支并不意味着移动到它。

工作副本完成后，开发人员使用 add 命令指定哪些文件应位于 Stage Area 中，并使用 commit 命令将更改提交到本地存储库。 使用 push 命令，可以将结果发送到远程存储库。

•git status - 显示文件系统中文件的当前状态以及有关执行编辑的分支的信息。

•git log 显示提交日志，--graph 选项以图形方式显示进行更改的分支。

•git difT - 将显示自上次提交更改以来所做的所有更改。

•git reset --hard HEAD 将重置对当前本地存储库所做的所有更改。

•git branch - 显示分支。

•git checkout - 在分支之间切换开发人员 git merge - 将多个分支合并到当前分支 git commit - 将更改提交到当前分支。 -t 选项指定将向用户显示的提交消息。

• git add - 将更改的文件添加到后续提交中，将它们放置在暂存区中。

git-flow，你将会拥有一些扩展命令。这些命令会在一个预定义的顺序下自动执行多个操作

创建一个新功能：git flow feature start feature\_name; 指令更改结束：git flow feature end feature\_name;

当在项目的根目录执行 “git flow init” 命令时,会询问用户几个问题并进行必要设置

**Subversion (SVN)：**

1. 两种储存版本数据的方法：BDB和FSFS
2. 存储库的访问：
3. svnserve(守护进程控制，它接收用户命令并对存储库进行适当的更改)
4. Apache服务器

file:///：直接访问 SVN 存储库（在本地硬盘上）

http://:通过WebDAV协议访问SVN支持的Apache服务器

https://:作为 http://，但使用 SSL 加密

svn:// :通过自己的协议访问 svnserve 服务器

svn+ssh://:作为 svn://，但通过 SSH 建立隧道

1. 多种协议对存储库进行远程访问:svn 协议或svn+https和ssh+svn
2. 客户端不只是把数据从存储库传输到客户端并返回，它还管理开发人员修改的文件的本地副本
3. Diff(di):显示两个版本或者路径的差异

svn merge：通常需要将一个分支的更改与另一个分支或者主干的更改合并，形成一个公共分支

SVN 的一些概念：

repository（源代码库）:源代码统一存放的地方

Checkout（提取）:当你手上没有源代码的时候，你需要从 repository checkout 一份

Commit（提交）:当你已经修改了代码，你就需要Commit到repository

Update (更新):当你已经 checkout 了一份源代码， update 一下你就可以和Repository上的源代码同步，你手上的代码就会有最新的变更

内容冲突：当两名开放人员修改了同一个文件中相同的行就会发生文件冲突

结构冲突：当两名开发人员移动，重命名，删除一个文件或者文件夹就会发生结构冲突

解决方案

1. 使用svn postpone来将处理推迟，或与开发者联系
2. 根据repository中的版本更改本地副本中的结构来解决冲突，也可以修改储存库中的结构的人修复达成一致

### trunk、branch、tag：

**trunk**：是开发的主体，从项目开始一直到现在。

**branch**：是从主干中的某个点派生的代码副本，用于对代码进行重大更改，同时保持主干中代码的完整性。如果主要更改按计划进行，它们通常会合并回主干。Branch 选项会给开发者创建出另外一条线路。当有人希望开发进程分开成两条不同的线路时，这个选项会非常有用。

**tag**：是您希望保留的主干或分支上的时间点。保留的两个主要原因是这是软件的主要版本，无论是 alpha、beta、RC 还是 RTM，或者这是在主干上应用重大修订之前软件的最稳定点。我们可以给某一个具体版本的代码一个更加有意义的名字。

Git cherry-pick:想在某个稳定版本上，添加一个刚开发完成的版本功能，可以使用cherry-pick，将这个功能相关的commit提取出来，何如稳定的版本分支上

需要将一个分支的所有代码变动，那么采用合并（git merge）

只需要某个分支的部分代码变动，那么使用cherry pick