# 版本控制系统

什么是版本控制

版本控制（Revision control）是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

版本控制的目的

实现跨区域多人协同开发

追踪和记载一个或者多个文件的历史记录

组织和保护你的源代码和文档

统计工作量

并行开发、提高开发效率

跟踪记录整个软件的开发过程

减轻开发人员的负担，节省时间，同时降低人为错误

简单说就是用于管理多人协同开发项目的技术

版本控制可以做什么？

自动生成备份、知道改动的地方、随时回滚

常见的版本控制器

Git

SVN（Subversion） - 集中式的额版本控制系统

CVS（Concurrent Versions System）

VSS（Micorosoft Visual SourceSafe）

TFS（Team Foundation Server）

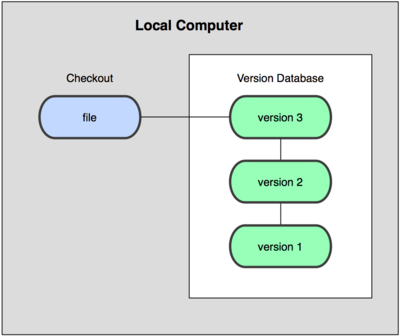
Visual Studio Online

版本控制产品非常的多（Perforce、Rational ClearCase、RCS（GNU Revision Control System）、Serena Dimention、SVK、BitKeeper、Monotone、Bazaar、Mercurial、SourceGear Vault），现在影响力大且使用广泛的是 Git 与 SVN。

## 版本控制方法类别

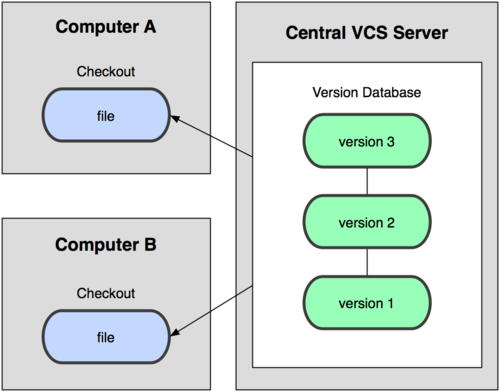
### **1、本地版本控制**

记录文件每次的更新，可以对每个版本做一个快照，或是记录补丁文件，适合个人用，如RCS。



### ****2、集中版本控制****

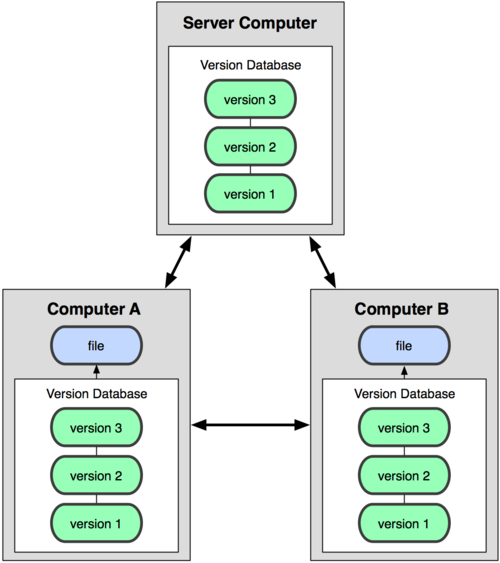
所有的版本数据都保存在服务器上，协同开发者从服务器上同步更新或上传自己的修改所有的版本数据都存在服务器上，用户的本地只有自己以前所同步的版本，如果不连网的话，用户就看不到历史版本，也无法切换版本验证问题，或在不同分支工作。而且，所有数据都保存在单一的服务器上，有很大的风险这个服务器会损坏，这样就会丢失所有的数据，当然可以定期备份。代表产品：SVN、CVS、VSS。



### 3、分布式版本控制

所有版本信息仓库全部同步到本地的每个用户，这样就可以在本地查看所有版本历史，可以离线在本地提交，只需在连网时 push 到相应的服务器或其他用户那里。

由于每个用户那里保存的都是所有的版本数据，只要有一个用户的设备没有问题就可以恢复所有的数据，但这增加了本地存储空间的占用。





## 集中式版本控制系统SVN

SVN 教程：<https://www.runoob.com/svn/svn-tutorial.html>

### 优点和劣势

优点：

1.管理方便，逻辑明确，操作简单，上手快。

2.易于管理，集中式服务器更能保证安全性。

3.代码一致性非常高。

4.有良好的目录级权限控制系统。

劣势：

1.对服务器性能要求高，数据库容量经常暴增，体量大。

2.必须联网。如果不能连接到服务器上，基本上不可以工作，如果服务器不能连接上，就不能提交，还原，对比等等。

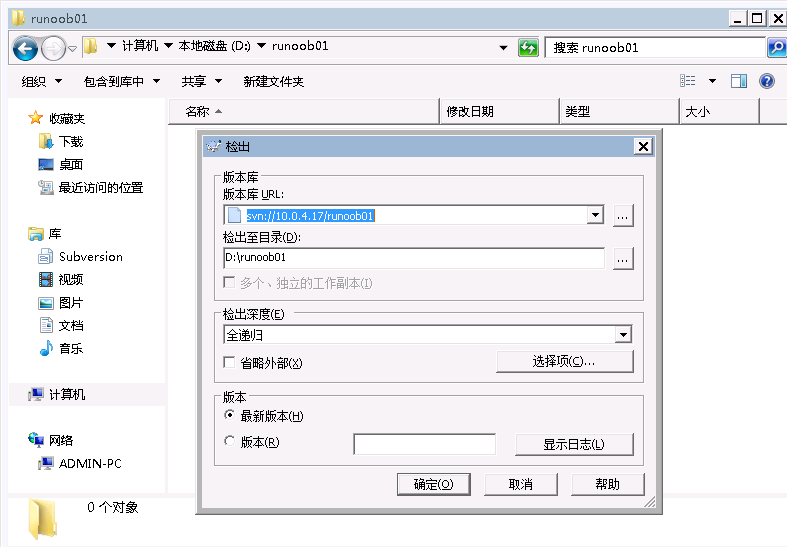
3.不适合开源开发。

4.分支的管控方式不灵活

### 操作步骤

工作目录建立步骤：

1. 建立文件夹project，作为工作目录；
2. 进入工作目录，空白处右键，SVN checkout；
3. 在版本库repository位置填入版本仓库地址 svn://10.0.4.17/project，在Checkout directory(检出至目录)填入D://project，点击ok，显示检出完成；



1. 此时，可以在工作目录下看到一个.svn的隐藏文件夹；

新增档案及目录步骤：

1. 创建目录 dir01, 在目录里新增文件file01.txt；
2. 空白处右键SVN checkout，Checkout directory(检出至目录)填入dir01，点击ok，显示加入完成；
3. 空白处右键SVN commit(提交)，在"信息"文本框中可以写入对本次 commit 的说明，点击"确认"后完成 commit 动作；
4. 可以到 project 目录中，确定是否所有的档案 icon 都有绿色勾，这样代表您的档案都正确无误的到 repository 中。

由于版本控制系统多半都是由许多人共同使用。所以，同样的档案可能还有人会去进行编辑。为了确保您工作目录中的档案与 Repository 中的档案是同步的。建议您在编辑前都先进行更新的动作。

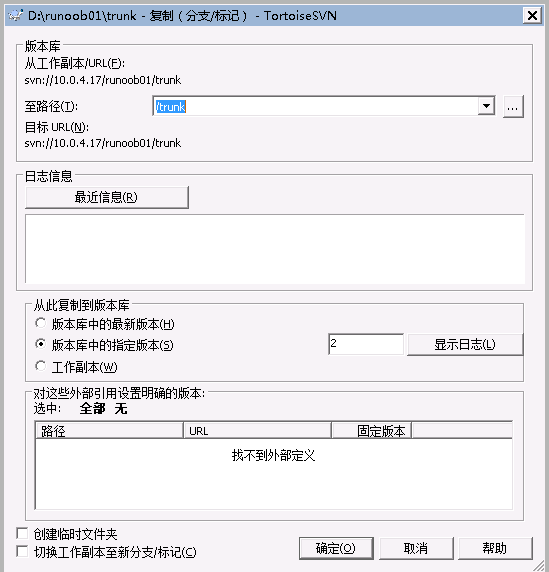
更新档案及目录步骤：

1. 在想要更新的档案或目录 icon 上面按下鼠标右键，选择 SVN Update；
2. 需要回溯至特定的日期或是版本，在想要更新的档案或目录 icon 上面按下鼠标右键，选择 TortoiseSVN->Update to revision(更新至版本)。

很多时候您会希望有另外一个复制的目录来进行新的编修。等到确定这个分支的修改已经完毕了，再合并到原来的主要开发版本上。

复制档案及目录步骤：

1. 在 dir01 的 icon 下面按下鼠标右键选择 Branch/Tag…(分支/标记) , 确认From WC at URL(从工作副本/URL)复制的来源目录和To URL(至路径)中要复制过去的路径，Log message(日志信息)输入此次 branch 的目的，按下 OK，关闭窗口；



1. 在 branch 子目录下面进行 SVN update 就可以看到这个新增的目录了。新增的目录就与原来的目录无关了。您可以任意对他进行编辑，一直到您确认好所有在 branch 下面该做的工作都完成后，您可以选择将这个 branch merge 回原来的 trunk 目录，或者是保留它在 branch 中。

合并动作……

## 分布式版本控制系统Git

### Git的基本概念及工作流程

四个区

1.工作区，为github的文件夹，.git文件夹为版本库，版本库中有stage（或者叫index）的暂存区、第⼀个分支master，以及指向master的⼀个指针叫HEAD；

2.暂存区stage（或者叫index）；

3.本地仓库；

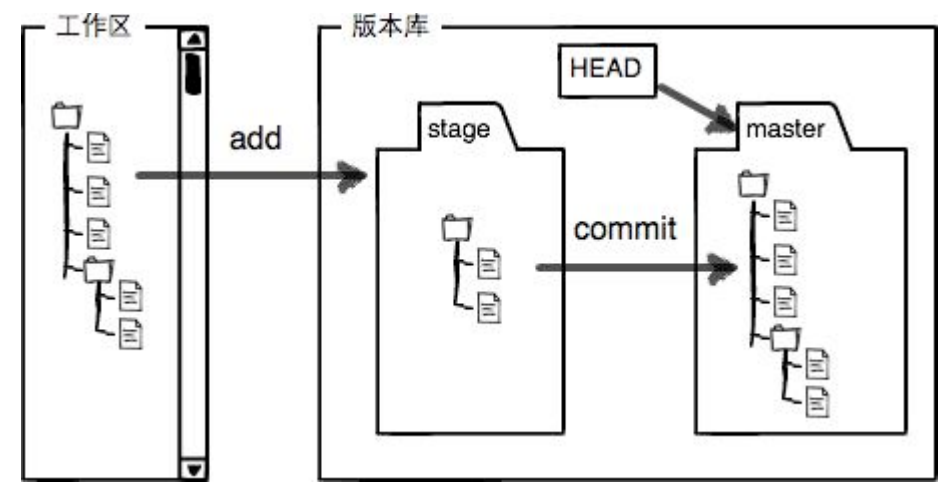
4.远端仓库。

把文件往git版本库里添加的时候，是分两步执行的：

第⼀步是⽤“git add”把⽂件添加进去，实际上就是把⽂件修改添加到暂存区；

第⼆步是⽤“git commit”提交更改，实际上就是把暂存区的所有内容提交到当前分支。

因为我们创建Git版本库时，Git自动为我们创建了唯⼀⼀个master分支，所以，现在，commit就是往master分支上提交更改。



git工作流程

1.在本地新建git仓库来存储我们的项目代码，git init；

2.将我们的项目代码交给git管理，添加至暂存区，git add filename；

3.如果确定了暂存区里的内容可以提交了，就将它提交至本地库，git commit –m “add 1 files”；

4.将本地库里的内容推送至远端仓库。

优点

Git的优缺点如下：

1．适合分布式开发，每一个个体都可以作为服务器。每一次Clone就是从服务器上pull到了所有的内容，包括版本信息。

2．公共服务器压力和数据量都不会太大。

3．速度快、灵活，分支之间可以任意切换。

4．任意两个开发者之间可以很容易的解决冲突，并且单机上就可以进行分支合并。

5．离线工作，不影响本地代码编写，等有网络连接以后可以再上传代码，并且在本地可以根据不同的需要，本地新建自己的分支。

### 常用指令

查看所有配置：git config -l

查看系统配置：git config --system --list

查看用户自己配置：git config --global --list

本地配置文件位置：C:\Users\Administrator\ .gitconfig

查看git用户名：git config user.name

查看git用户邮箱：git config user.email

配置git用户名：git config --global user.name XiaodongCai93

配置git用户邮箱：git config --global user.email [2697251342@qq.com](mailto:2697251342@qq.com)

查看版本信息：git --version

2.17.1版本之前的版本更新：git update

2.17.1版本之后的版本更新：git update-git-for-window

初始化一个本地仓库：git init

将工作区里的内容添加至暂存区：git add README.md

将暂存区内容提交至本地库：git commit -m “first commit”

删除本地仓库版本库的文件：git rm filename

将本地仓库与远端仓库建立链接：git remote add origin <https://github.com/XiaodongCai93/papers.git>

其中，origin为远程仓库的命名，<https://github.com/XiaodongCai93/papers.git>为远程仓库的地址

查看远程仓库地址：git remote –v

将本地仓库内容推送到远端仓库的master分支：git push -u origin master

强制推送到远程分支：git push -f

查看远程所有分支：git branch -a

拉取远程仓库origin的master分支，并与本地仓库brantest分支合并：git pull origin master:brantest

删除远程仓库：git remote rm origin

修改仓库名：git remote rename old\_name new\_name

查看工作区和暂存区的状态：git status

查看修改内容：git diff 1.txt

查看详细的提交历史记录：git log

查看精简的提交历史记录： git log –oneline

查看版本回滚历史：git reflog

取消已缓存的内容：git reset HEAD

将工作区和暂存区回滚到上个版本：git reset --hard HEAD^或者git reset --hard HEAD~1

将工作区和暂存区回滚到指定版本：git reset --hard 89e6f38f

切换本地仓库的分支，并替换工作区的版本：git checkout branchName

### 密钥

远程仓库通过SSH加密，配置文件在C:\Users\Administrator\.ssh下，id\_rsa和id\_rsa.pub两个⽂件是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放⼼地告诉任何⼈。

创建SSH Key：ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)

把GitHub的“Account settings”的“SSH Keys”的“Add SSH Key”的Key⽂本框⾥粘贴id\_rsa.pub⽂件的内容，点“Add Key”。

把本地仓库和GitHub的远程仓库关联：git remote add origin [git@github.com](mailto:git@github.com):michaelliao/learngit.git

第⼀次把本地库的所有内容推送到远程库上：git push -u origin master，实际上是把当前分支master推送到远程，非第一次，可以去掉-u。

克隆远程库到本地：git clone [git@github.com:michaelliao/gitskills.git](mailto:git@github.com:michaelliao/gitskills.git)

### 分支管理

创建dev分支，然后切换到dev分支：git checkout -b dev

其中git checkout命令加上-b参数表⽰创建并切换，相当于以下两条命令：git branch dev和git checkout dev

查看当前分支：git branch

切换回master分支：git checkout master

把dev分支合并到master分支上：git merge dev

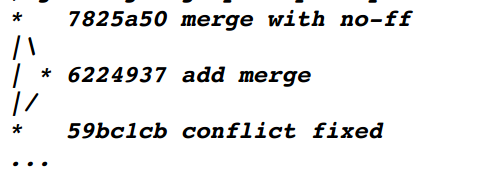
删除dev分支：git branch -d dev

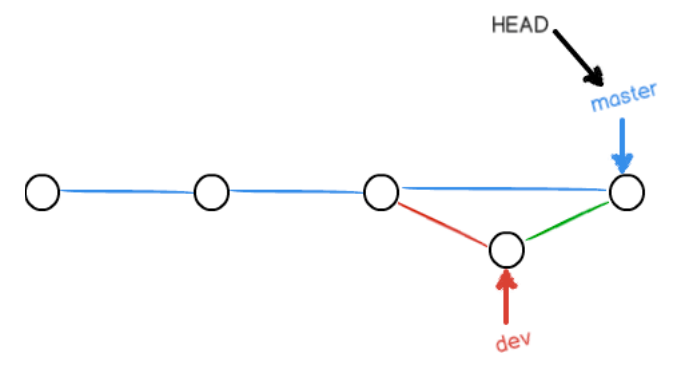
查看分支的合并情况：git log --graph --pretty=oneline --abbrev-commit

通常，合并分支时，如果可能，Git会⽤“Fast forward”模式，但这种模式下，删除分支后，会丢掉分支信息。

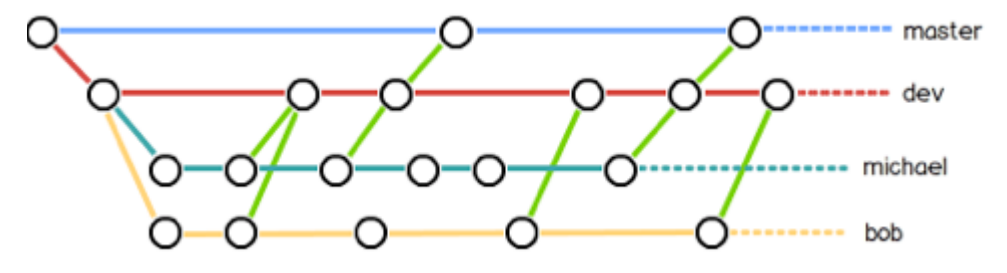
如果要强制禁⽤“Fast forward”模式，Git就会在merge时⽣成⼀个新的commit，这样，从分支历史上就可以看出分支信息。

--no-ff⽅式的merge：git merge --no-ff -m "merge with no-ff" dev





分支策略，团队合作的分支看起来就像这样：



#### Bug分支

当你接到⼀个修复⼀个代号101的bug的任务时，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作：

把当前工作现场“储藏”起来：git stash

切换会dev分支：git checkout dev

查看工作现场：git stash list

恢复现场：git stash apply

删除现场：git stash drop

恢复的时候删除：git stash pop

### 多人协作

查看远程库的信息：git remote –v

把分支推送到远程库对应的远程分支上：git push origin dev

master分支是主分支，因此要时刻与远程同步；dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；bug分支只⽤于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

git clone [git@github.com:michaelliao/learngit.git](mailto:git@github.com:michaelliao/learngit.git)

git checkout -b dev origin/dev

设置dev和origin/dev的链接：git branch --set-upstream dev origin/dev

把最新的提交从origin/dev抓下来：git pull

git commit -m "add /usr/bin/env"

git commit -m "merge & fix hello.py"

git push origin dev

### 标签管理

创建标签

切换到需要打标签的分支上：git branch

打⼀个新标签：git tag v1.0

⽤命令git tag查看所有标签：git tag

历史提交的commit id打上标签：git tag v0.9 6224937

创建带有说明的标签，-a指定标签名，-m指定说明⽂字：git tag -a v0.1 -m "version 0.1 released" 3628164

查看标签信息：git show v0.9

通过-s⽤私钥签名⼀个标签：git tag -s v0.2 -m "signed version 0.2 released" fec145a

操作标签

删除标签：git tag -d v0.1

推送某个标签到远程：git push origin v1.0

⼀次性推送全部尚未推送到远程的本地标签：git push origin –tags

删除远程标签：git tag -d v0.9

git push origin :refs/tags/v0.9 to [git@github.com:michaelliao/learngit.git](mailto:git@github.com:michaelliao/learngit.git) - [deleted] v0.9

Gitlab/gitlab-ce