**工具理论与使用**

1. git

**1).git add 与gitstage的区别**

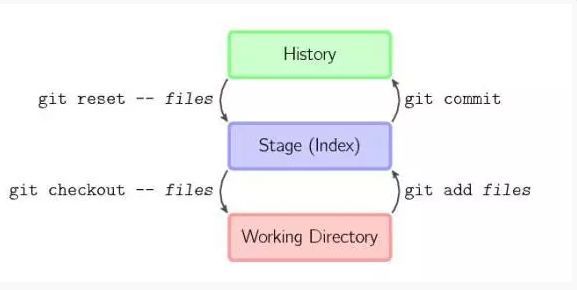
git stage只是git add的同义词，所以在使用上没有区别

i)Git仓库的三个组成部分：工作区（Working Directory）、暂存区（Stage）、历史记录区（History）

ii)工作区：在Git管理的正常目录都算是工作区，我们平时编辑工作都是在工作区完成。

iii)暂存区：临时区域。里面存放将要提交的文件快照。

历史记录区：git commit 后的记录区。



git add 和git stage,其实这两个命令是同一个意思，是因为要跟 svn add 区分，两者的功能是完全不一样的，svn add 是将某个文件加入版本控制，而 git add 则是把某个文件加入暂存区，因为在 git 出来之前大家用 svn 比较多，所以为了避免误导，git 引入了git stage，然后把 git diff --staged 做为 git diff --cached 的相同命令。基于这个原因，我们建议使用 git stage 以及 git diff –staged.

**2). git reset 、git revert和git checkout 有什么区别**

**共同点**：用来撤销代码仓库中的某些更改

**不同点**:

i). git reset可以将一个分支的末端指向前一个commit。然后再下次git执行垃圾回收的时候，会把这个commit之后的commit都扔掉;

ii). git reset还支持三种标记。用来标记reset指令的影响范围;

-mixed：会影响到暂存区和历史记录区。也是默认选项

--soft：只影响历史记录区

--hard：影响工作区，暂存区和历史记录区

因为git reset是直接删除commit记录，从而会影响其他开发人员的分支，所以不要在公共分支做这个操作

git checkout 可以将HEAD移到一个新的分支，并更新工作目录。可能会覆盖本地的修改，所以执行这个指令之前，你需要stash或者commit暂存区和工作区的更改;

git revert和git reset的目的是一样的，但是做法不一样，它会创建新的commit的方式来撤销commit，这样能保留之前的 commit 历史，比较安全。另外，同样因为可能会覆盖本地的修改，所以执行这个指令之前，你需要 stash 或者 commit 暂存区和工作区的更改;

**3).git fetch和git merge和git pull的区别**

git pull: 相当于git fetch 和 git merge，即更新远程仓库的代码到本地仓库，然后将内容合并到当前分支;

git fetch：相当于是从远程获取最新版本到本地，不会自动merge ;

git merge : 将内容合并到当前分支.

**4).git 与svn的区别**

i).git是分布式的,SVN不是: GIT跟SVN一样有自己的集中式版本库或服务器,但GIT更倾向于被使用于分布式模式，也就是每个开发人员从中心版本库/服务器上chectout代码后会在自己的机器上克隆一个自己的版本库;

ii).git把内容按元数据存储,而SVN是按文件存储: 所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。如果你把.git目录的体积大小跟.svn比较，你会发现它们差距很大。因为,.git目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等;

iii).git分支与SVN分支的不同: 分支在SVN中一点不特别，就是版本库中的另外的一个目录。如果你想知道是否合并了一个分支，你需要手工运行像这样的命令[svn propget svn:mergeinfo](http://jan.baresovi.cz/dr/en/subversion-mergeinfo)，来确认代码是否被合并, 所以，经常会发生有些分支被遗漏的情况; 然而，处理GIT的分支却是相当的简单和有趣。你可以从同一个工作目录下快速的在几个分支间切换。你很容易发现未被合并的分支，你能简单而快捷的合并这些文件.

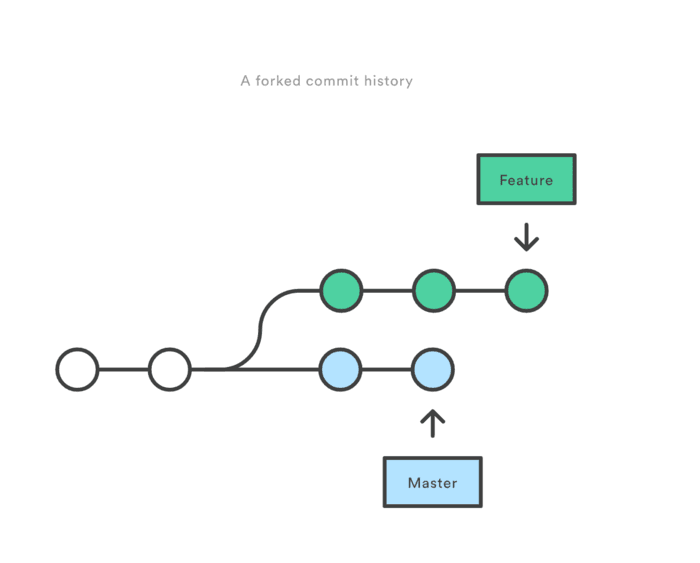
iv).git没有一个全局的版本号,而SVN有;

v).git内容的完整性要优于SVN: GIT的内容存储使用的是[SHA-1](http://en.wikipedia.org/wiki/SHA-1)哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏.

**5).git merge与git rebase的区别**

git rebase 和 git merge 一样都是用于从一个分支获取并且合并到当前分支，但是他们采取不同的工作方式，以下面的一个工作场景说明其区别:

如图所示：你在一个feature分支进行新特性的开发，与此同时，master 分支的也有新的提交:



为了将master 上新的提交合并到你的feature分支上，你有两种选择：merging or rebase

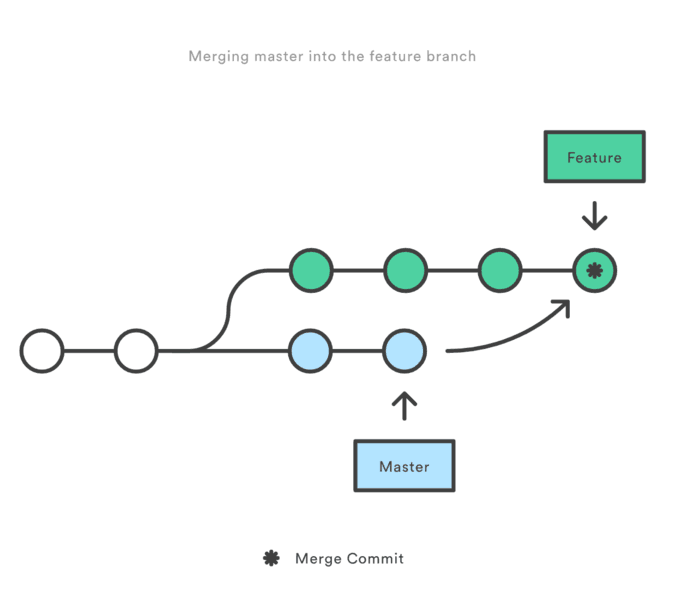
**用merge执行:**

git checkout feature

git merge master

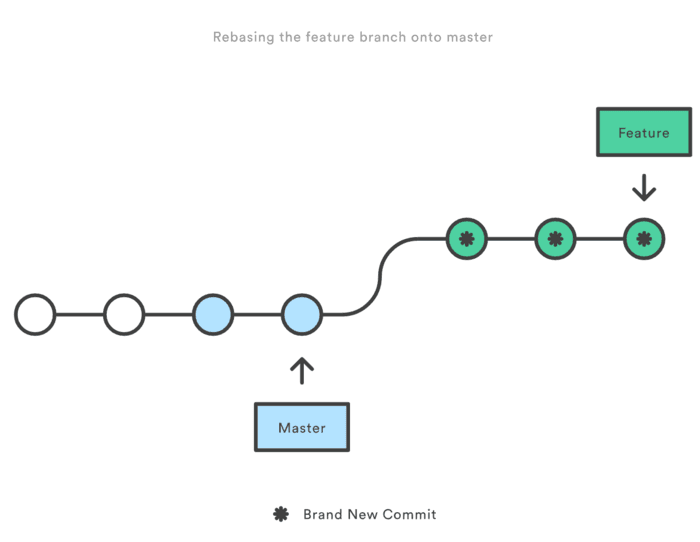
或者: git merge master feature

此时在feature上git 自动会产生一个新的commit(merge commit):



marge 特点：自动创建一个新的commit  
如果合并的时候遇到冲突，仅需要修改后重新commit  
优点：记录了真实的commit情况，包括每个分支的详情  
缺点：因为每次merge会自动产生一个merge commit，所以在使用一些git 的GUI tools，特别是commit比较频繁时，看到分支很杂乱  
**用rebase执行(本质是寻找公共祖先):**

git checkout feature

git rebase master  


rebase 特点：会合并之前的commit历史  
优点：得到更简洁的项目历史，去掉了merge commit  
缺点：如果合并出现代码问题不容易定位，因为re-write了history

合并时如果出现冲突需要按照如下步骤解决:

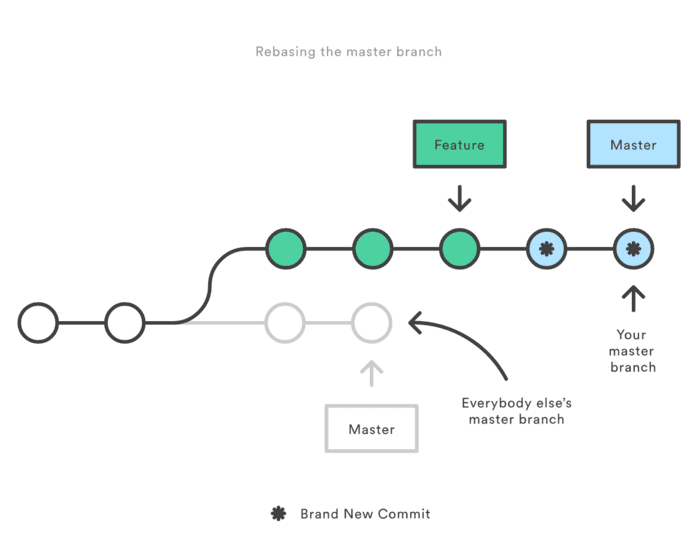
i). 修改冲突部分

ii). git add

iii). git rebase –continue

iv). 如果第三步无效可以执行 git rebase –skip

**git rebase的问题**: 不能再公共分支上执行该命令, 如果你rebase master 到你的feature分支, rebase 将所有master的commit移动到你的feature 的顶端。问题是：其他人还在original master上开发，由于你使用了rebase移动了master，git 会认为你的主分支的历史与其他人的有分歧，会产生冲突.



总结: 如果你想要一个干净的，没有merge commit的线性历史树，那么你应该选择git rebase;如果你想保留完整的历史记录，并且想要避免重写commit history的风险，你应该选择使用git merge.

**6).git常用命令**

**git init** ：创建git库

**git status** ：查看当前仓库的状态

**git diff** ：查看本次修改与上次修改的内容的区别

**git add 文件名** ：把现在所要添加的文件放到暂存区中

**git commit** ：把git add到暂存区的内容提交到代码区中

**git clone** ：从远程仓库拷贝代码到本地

**git branch** ：查看当前的分支名称

**git checkout** ：切换分支

**7).git冲突的解决**

**冲突的产生**: 很多命令都可能出现冲突，但从根本上来讲，都是merge 和 patch（应用补丁）时产生冲突。而rebase就是重新设置基准，然后应用补丁的过程，所以也会冲突。

git pull会自动merge，repo sync会自动rebase，所以git pull和repo sync也会产生冲突;

**常见冲突:** 两个用户修改了同一个文件的同一块区域，git会报告内容冲突(内容冲突);

**冲突场景:**

准备新的feature1分支，继续我们的新分支开发：

$ git checkout -b feature1

Switched to a new branch 'feature1'

修改readme.txt最后一行，改为：

Creating a new branch is quick AND simple.

在feature1分支上提交：

$ git add readme.txt

$ git commit -m "AND simple"

[feature1 14096d0] AND simple

1 file changed, 1 insertion(+), 1 deletion(-)

切换到master分支：

$ git checkout master

Switched to branch 'master'

Your branch is ahead of 'origin/master' by 1 commit.

(use "git push" to publish your local commits)

Git还会自动提示我们当前master分支比远程的master分支要超前1个提交。在master分支上把readme.txt文件的最后一行改为：

Creating a new branch is quick & simple.

提交：

$ git add readme.txt

$ git commit -m "& simple"

[master 5dc6824] & simple

1 file changed, 1 insertion(+), 1 deletion(-)

现在，master分支和feature1分支各自都分别有新的提交，变成了这样：



这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，我们试试看：

$ git merge feature1

Auto-merging readme.txt

CONFLICT (content): Merge conflict in readme.txt

Automatic merge failed; fix conflicts and then commit the result.

果然冲突了！Git告诉我们，readme.txt文件存在冲突，必须手动解决冲突后再提交。git status也可以告诉我们冲突的文件：

$ git status

On branch master

Your branch is ahead of 'origin/master' by 2 commits.

(use "git push" to publish your local commits)

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: readme.txt

no changes added to commit (use "git add" and/or "git commit -a")

我们可以直接查看readme.txt的内容：

Git is a distributed version control system.

Git is free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

<<<<<<< HEAD

Creating a new branch is quick & simple.

=======

Creating a new branch is quick AND simple.

>>>>>>> feature1

Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，我们修改如下后保存：

Creating a new branch is quick and simple.

再提交：

$ git add readme.txt

$ git commit -m "conflict fixed"

[master cf810e4] conflict fixed

现在，master分支和feature1分支变成了下图所示：



用带参数的git log也可以看到分支的合并情况：

$ git log --graph --pretty=oneline --abbrev-commit

\* cf810e4 (HEAD -> master) conflict fixed

|\

| \* 14096d0 (feature1) AND simple

\* | 5dc6824 & simple

|/

\* b17d20e branch test

\* d46f35e (origin/master) remove test.txt

\* b84166e add test.txt

\* 519219b git tracks changes

\* e43a48b understand how stage works

\* 1094adb append GPL

\* e475afc add distributed

\* eaadf4e wrote a readme file

1. maven

**1).maven的优点**

简化了项目依赖管理：   
易于上手，对于新手可能一个”mvn clean package”命令就可能满足他的工作   
便于与持续集成工具（jenkins）整合   
便于项目升级，无论是项目本身升级还是项目使用的依赖升级。   
有助于多模块项目的开发，一个模块开发好后，发布到仓库，依赖该模块时可以直接从仓库更新，而不用自己去编译。   
maven有很多插件，便于功能扩展，比如生产站点，自动发布版本等

**2).maven的缺点**

maven是一个庞大的构建系统，学习难度大   
maven采用约定优于配置的策略（convention over configuration），虽然上手容易，但是一旦出了问题，难于调试.

**3).maven坐标**

一般maven使用[groupID,artifactId,version，packaging]来表示一个项目的某个版本，有时还会使用classifier来表示项目的附属构建，常见的附属构建有javadoc和sources包