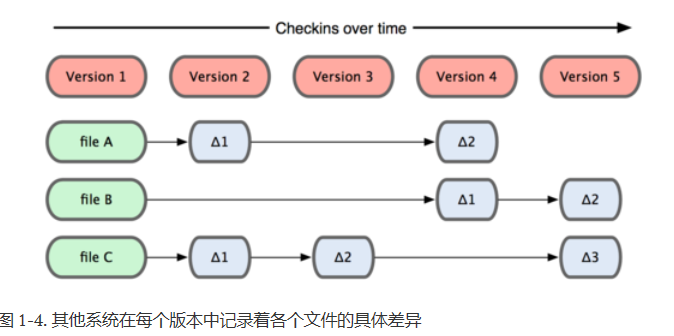
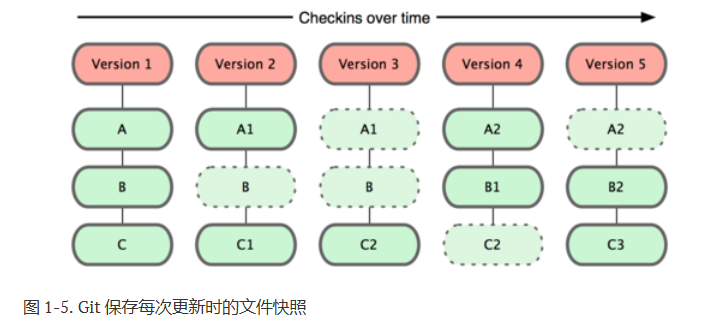
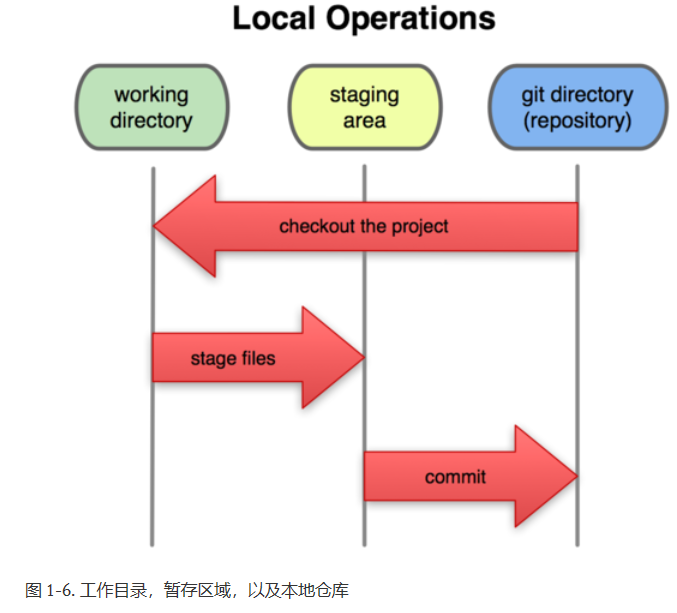
1. Git和其他版本控制系统的主要差别在于，Git只关心文件数据的整体是否发生变化，而其他系统则只关心文件内容的具体差异。这类系统（CVS,Subversion，Perforce，Bazaar等等）每次记录有哪些文件作了更新，以及都更新了哪些行的什么内容。

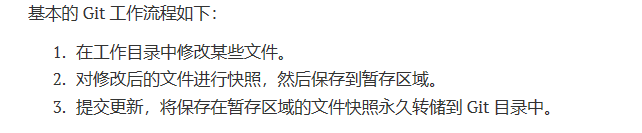
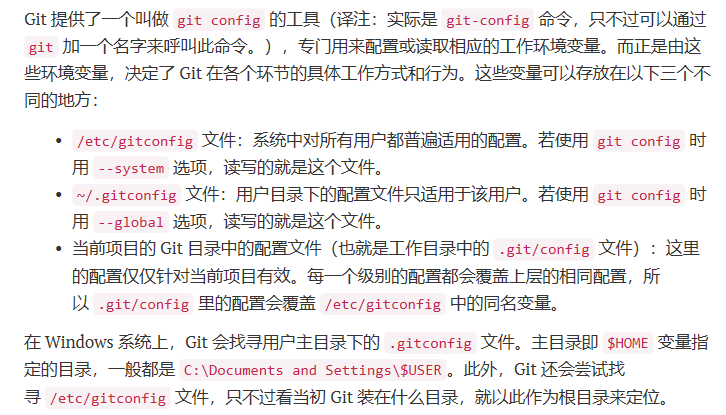


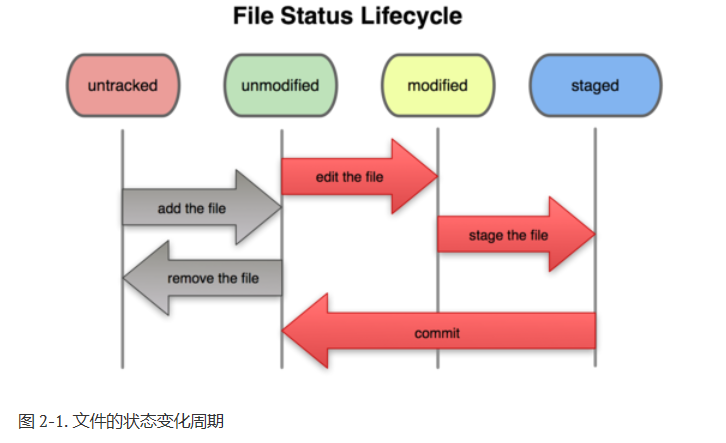
1. Git并不保存这些前后变化的差异数据。实际上，Git更像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。为提高性能，若文件没有变化，Git 不会再次保存，而只对上次保存的快照作一链接。

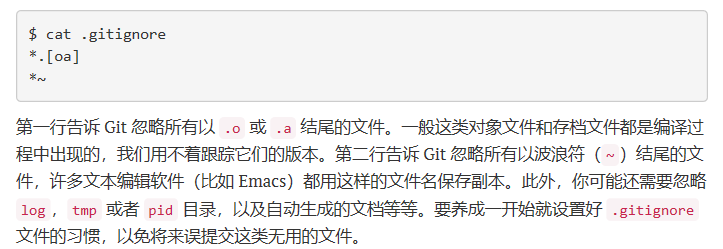
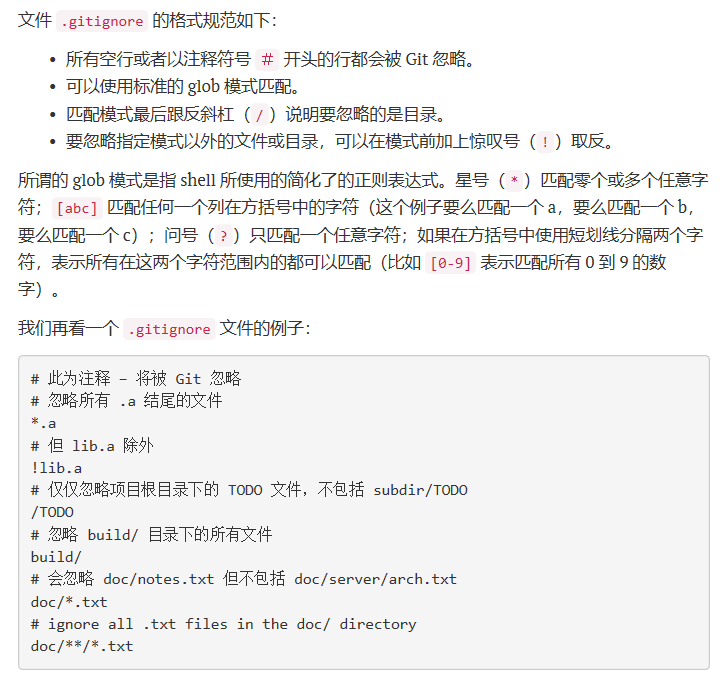
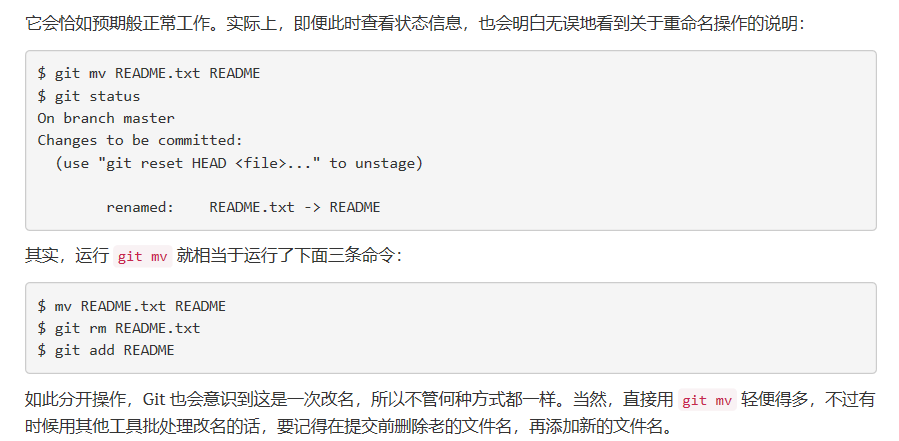
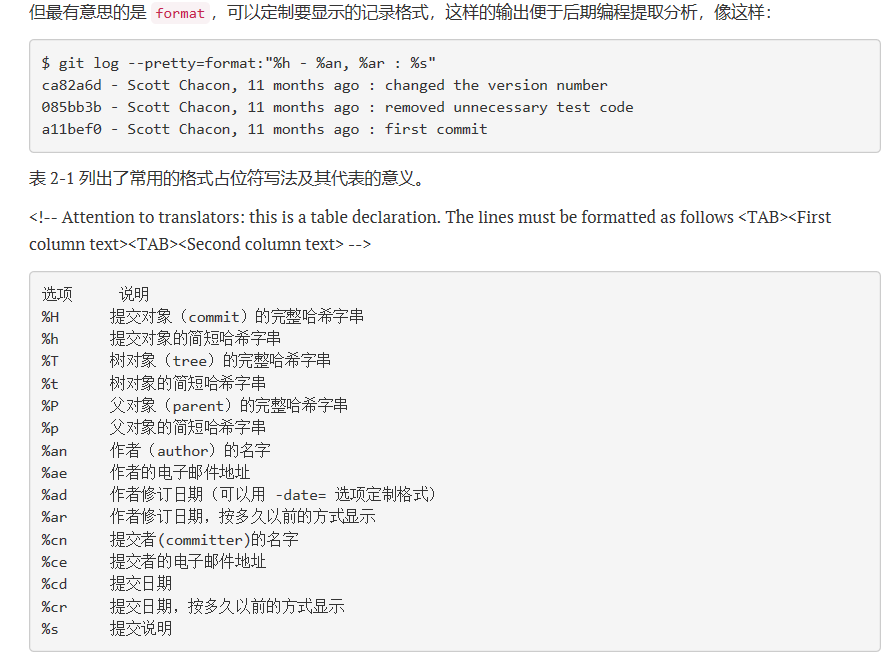


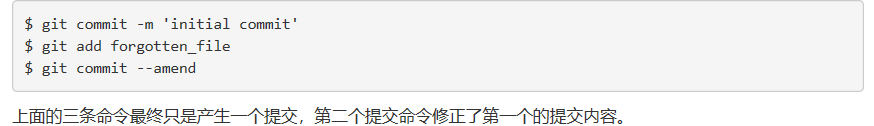
1. 几乎所有的操作都可以在本地完成，只有在需要推送远程仓库的时候联一下网就行。
2. 在保存到Git之前，所有的数据都要进行内容的校验和计算，并将此结果作为数据的唯一标识和索引。
3. 在 Git 内都只有三种状态：已提交（committed），已修改（modified）和已暂存（staged）。已提交表示该文件已经被安全地保存在本地数据库中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。



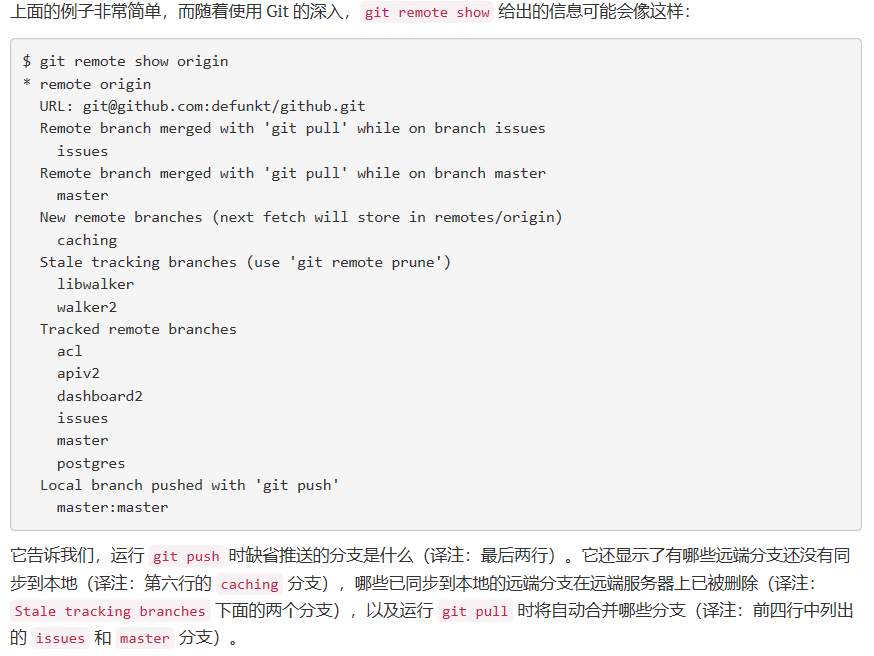
1. 
2. 
3. 
4. 查看已有的配置信息可以使用 git config --list 命令
5. 如果想对某个开源项目出一份力，可以先把该项目的 Git 仓库复制一份出来，这就需要用到 git clone 命令。如果你熟悉其他的 VCS 比如 Subversion，你可能已经注意到这里使用的是 clone 而不是 checkout。这是个非常重要的差别，Git 收取的是项目历史的所有数据（每一个文件的每一个版本），服务器上有的数据克隆之后本地也都有了。实际上，即便服务器的磁盘发生故障，用任何一个克隆出来的客户端都可以重建服务器上的仓库，回到当初克隆时的状态
6. 请记住，工作目录下面的所有文件都不外乎这两种状态：已跟踪或未跟踪。已跟踪的文件是指本来就被纳入版本控制管理的文件，在上次快照中有它们的记录，工作一段时间后，它们的状态可能是未更新，已修改或者已放入暂存区。而所有其他文件都属于未跟踪文件。它们既没有上次更新时的快照，也不在当前的暂存区域。初次克隆某个仓库时，工作目录中的所有文件都属于已跟踪文件，且状态为未修改。
7. 在编辑过某些文件之后，Git 将这些文件标为已修改。我们逐步把这些修改过的文件放到暂存区域，直到最后一次性提交所有这些暂存起来的文件，如此重复。所以使用 Git 时的文件状态变化周期如图 2-1 所示。

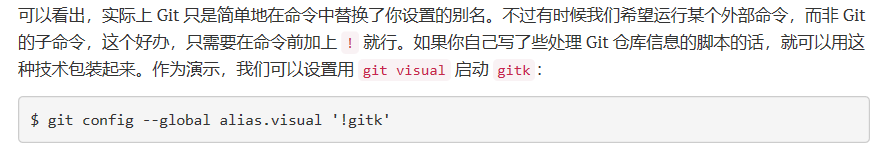
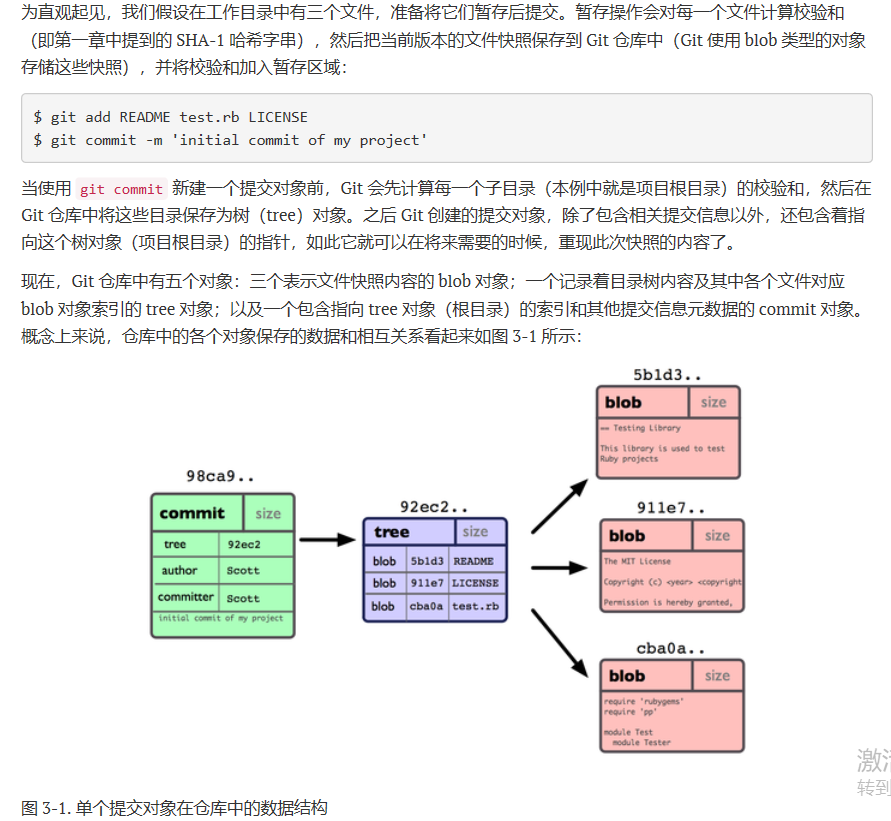
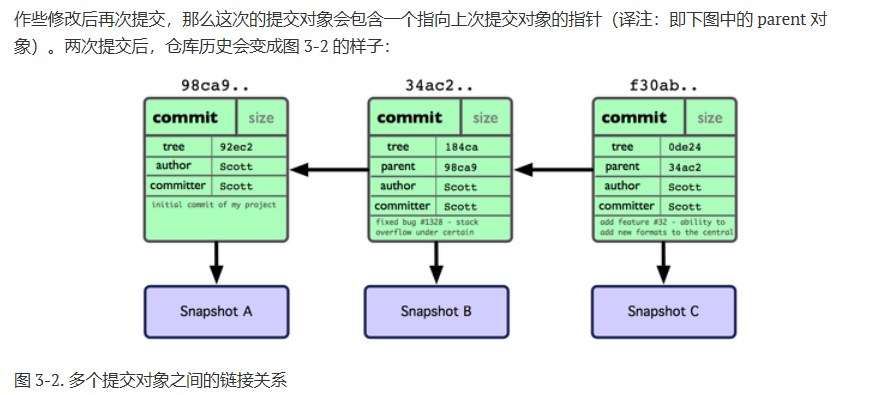
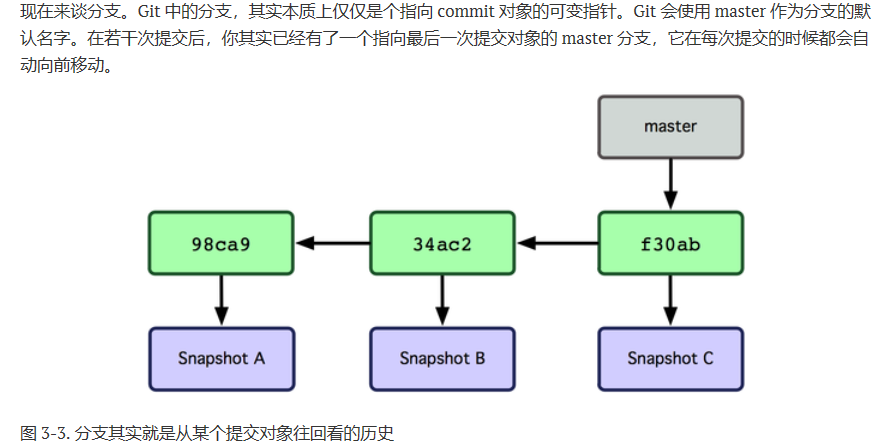
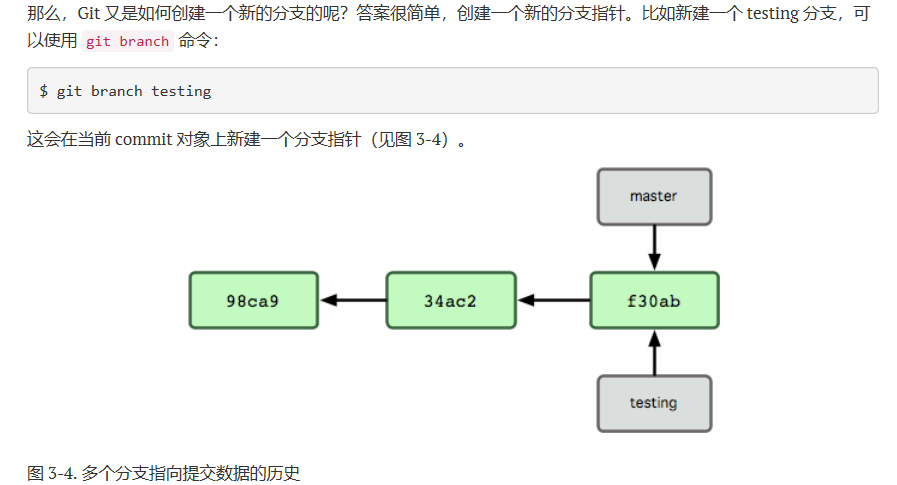
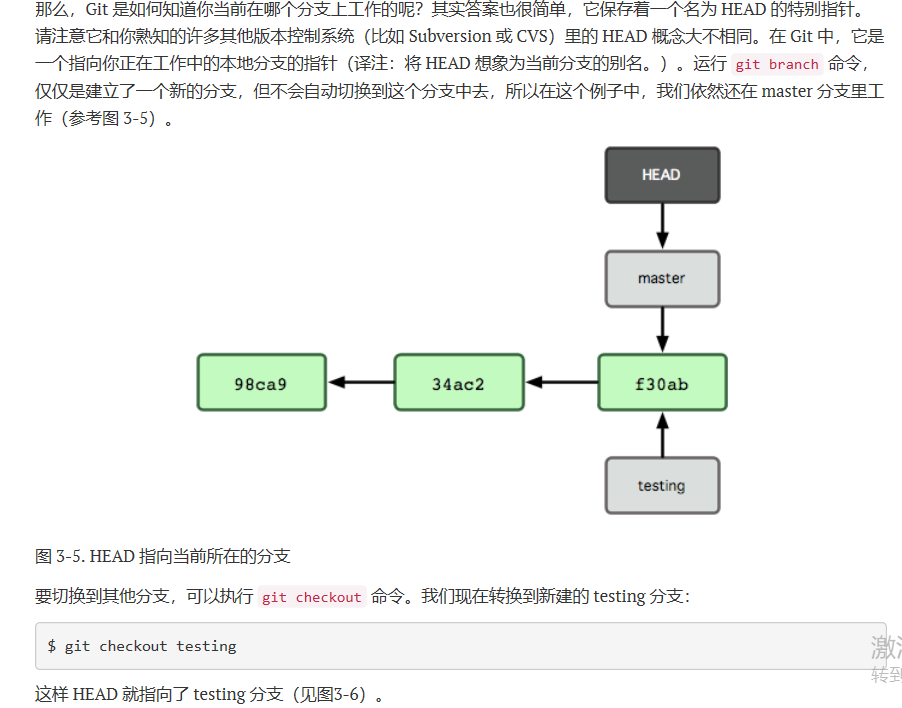
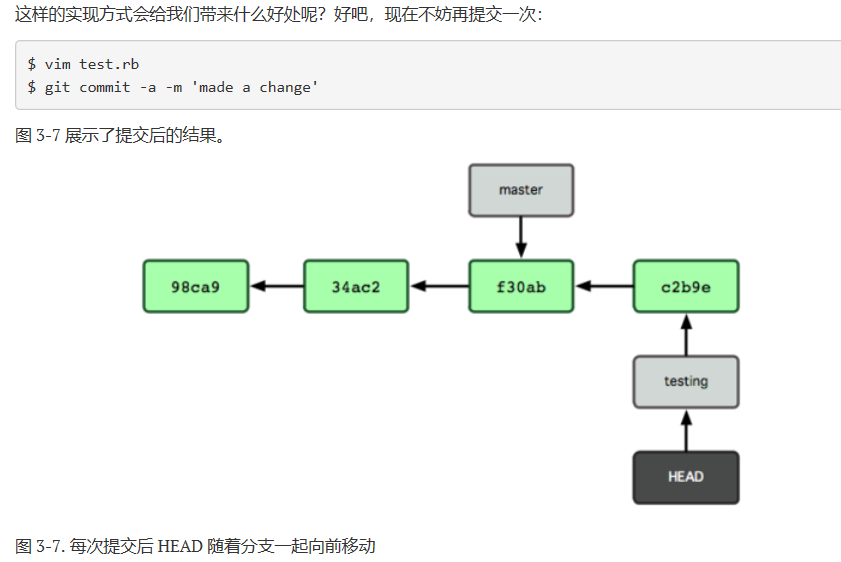
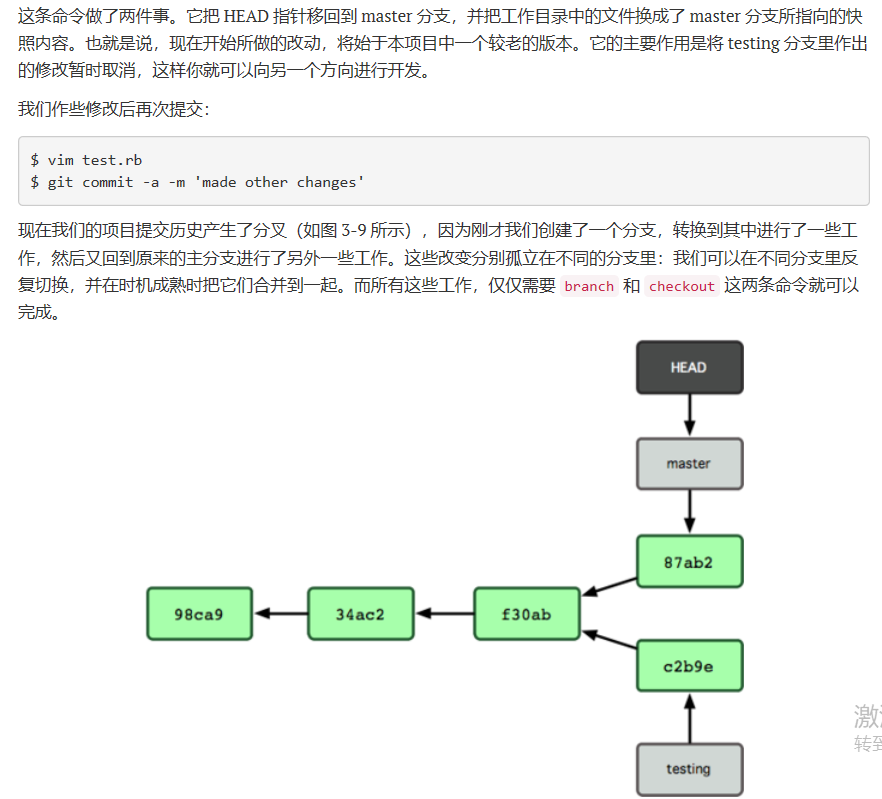
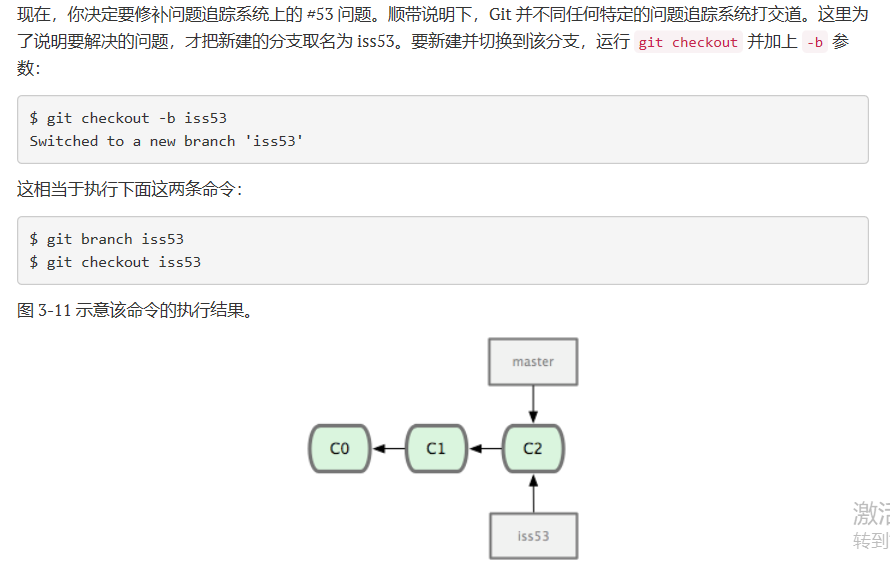
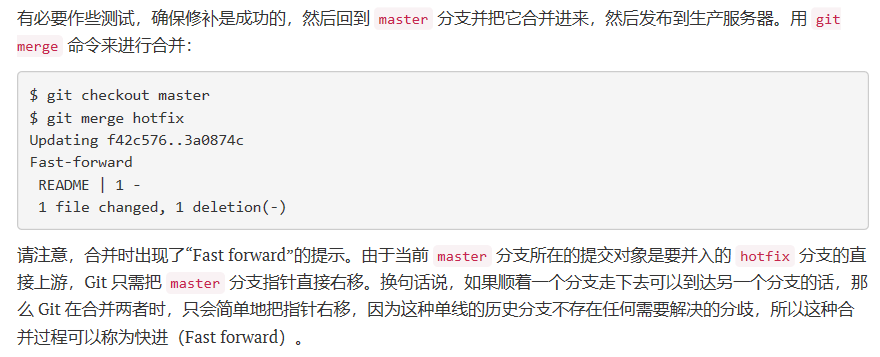
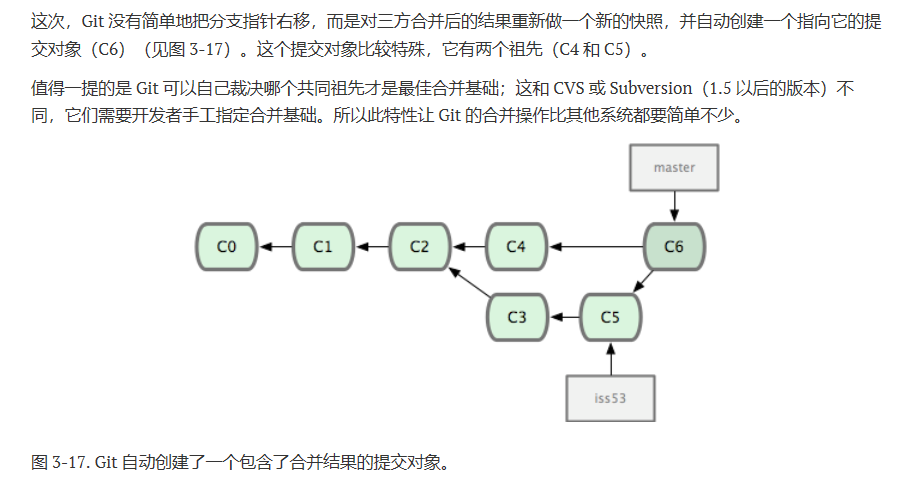
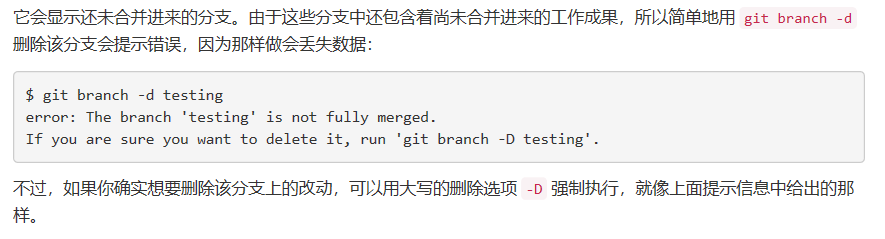
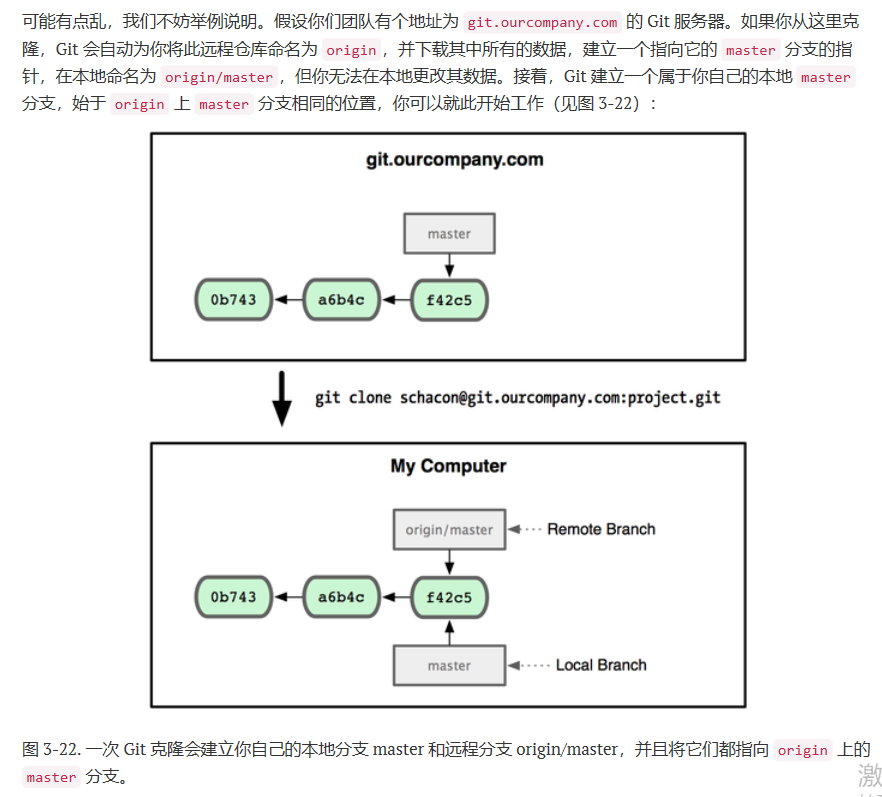
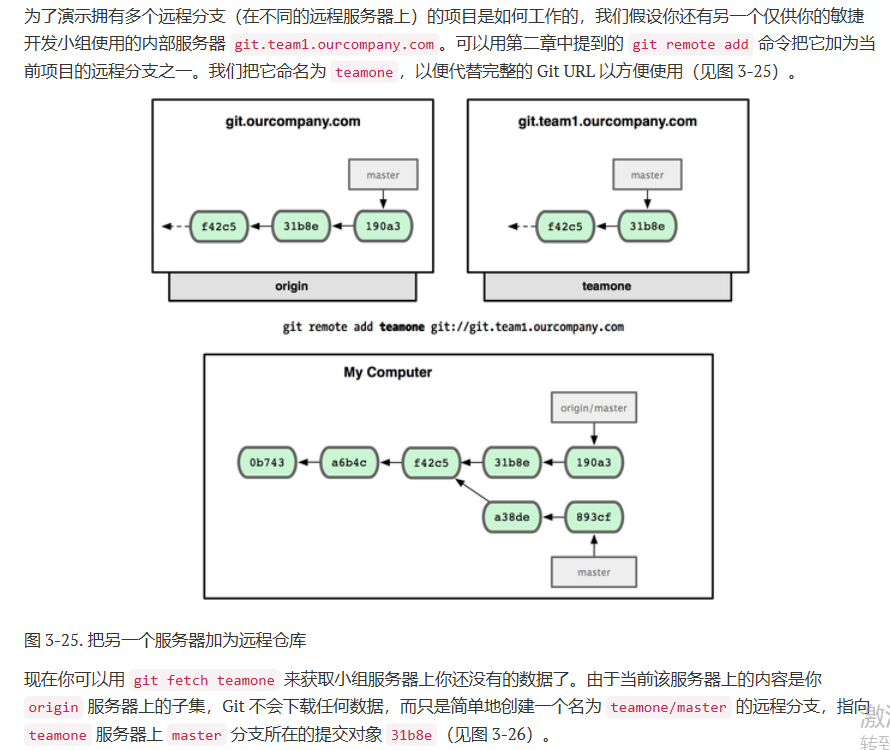
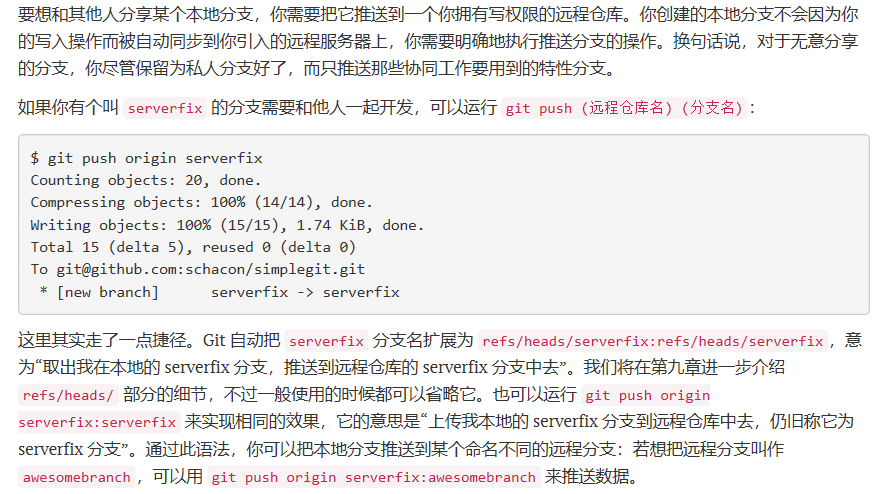
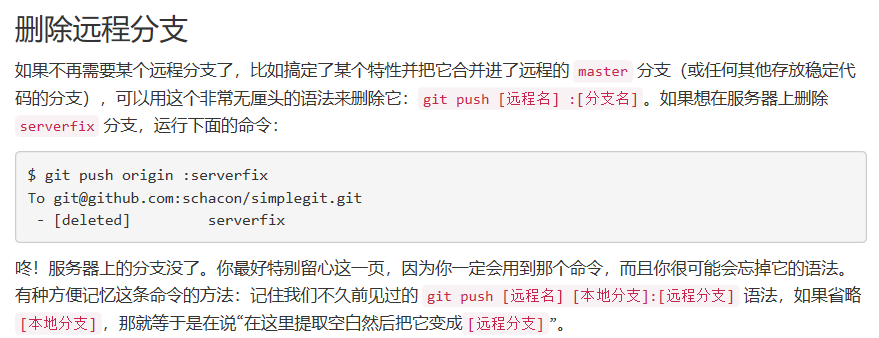
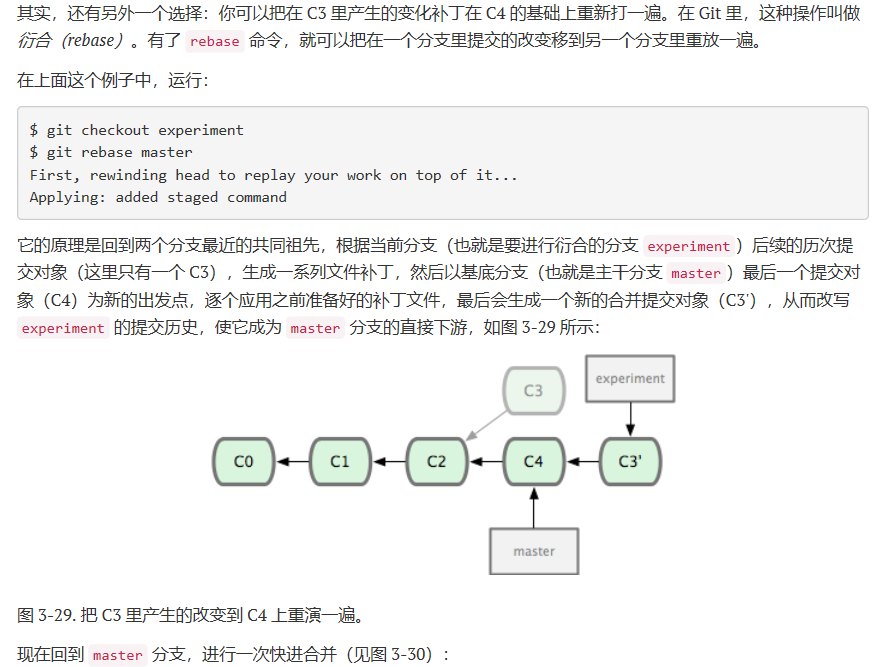
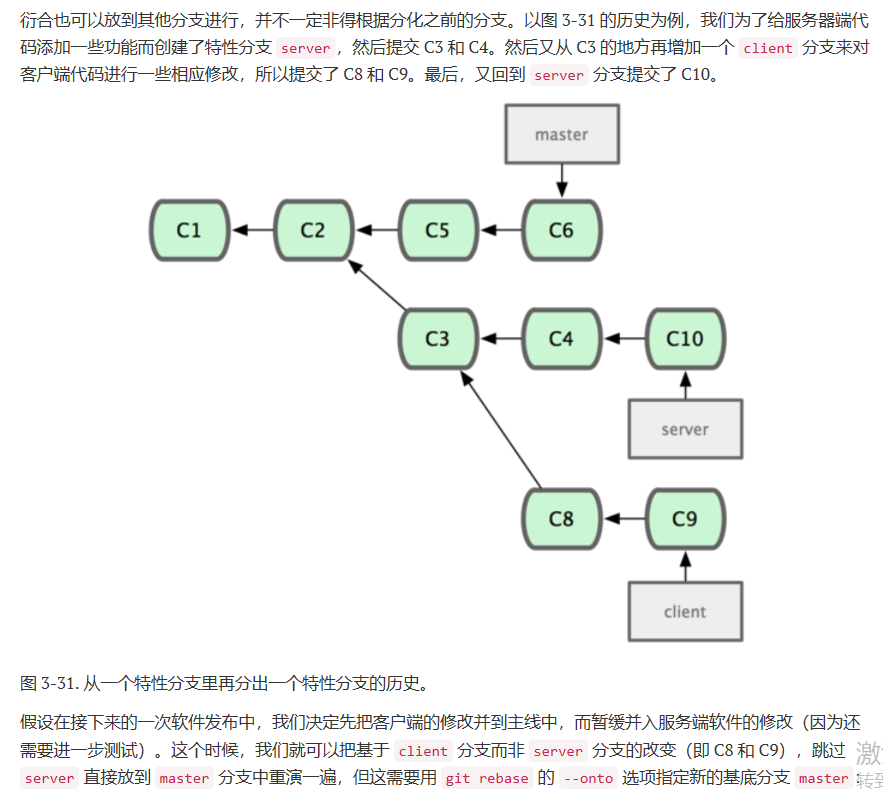


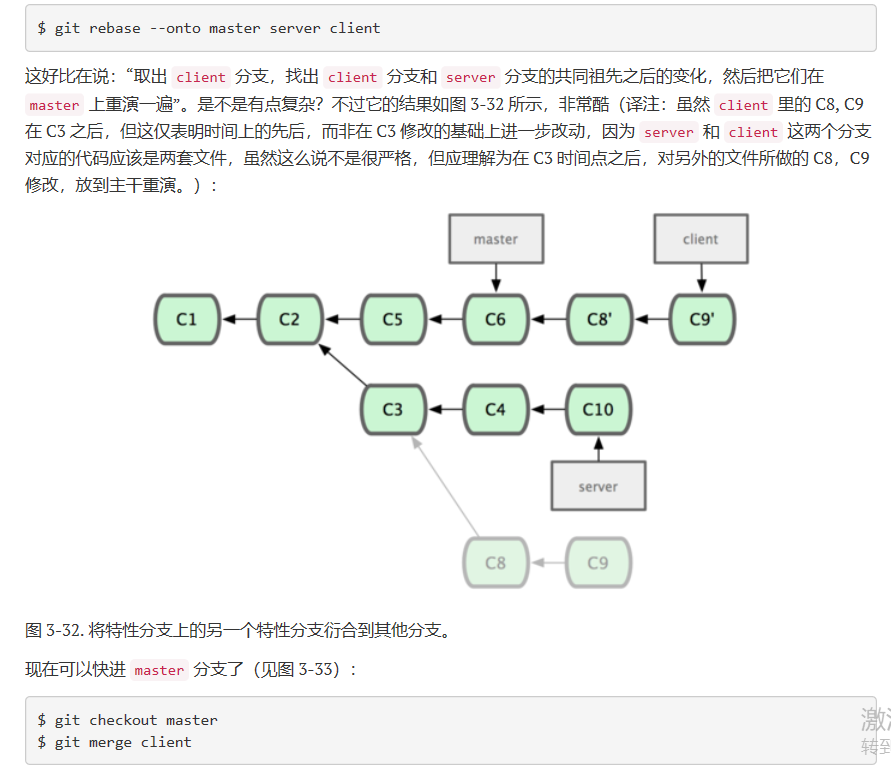
1. 
2. 没有跟踪的新文件可以通过git add 命令加入到暂存当中，这样就是跟踪状态，之后commit提交。
3. 修改已跟踪的文件，也需要git add把修改后的文件加入到暂存当中，再commit提交。
4. 一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式。
5. 
6. 
7. 单单 git diff 不过是显示还没有暂存起来的改动，而不是这次工作和上次提交之间的差异。所以有时候你一下子暂存了所有更新过的文件后，运行 git diff 后却什么也没有，就是这个原因。此命令比较的是工作目录中当前文件和暂存区域快照之间的差异。
8. 若要看已经暂存起来的文件和上次提交时的快照之间的差异，可以用 git diff --cached 命令。（Git 1.6.1 及更高版本还允许使用 git diff --staged，效果是相同的，但更好记些。）
9. Git 提供了一个跳过使用暂存区域的方式，只要在提交的时候，给 git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤。
10. 要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交。可以用 git rm 命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单中了。
11. 如果只是简单地从工作目录中手工删除文件，运行 git status 时就会在 “Changes not staged for commit” 部分（也就是未暂存清单），然后再运行 git rm 记录此次移除文件的操作，最后提交的时候，该文件就不再纳入版本管理了。如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项 -f（译注：即 force 的首字母），以防误删除文件后丢失修改的内容。
12. 
13. 
14. 默认不用任何参数的话，git log 会按提交时间列出所有的更新，最近的更新排在最上面。看到了吗，每次更新都有一个 SHA-1 校验和、作者的名字和电子邮件地址、提交时间，最后缩进一个段落显示提交说明。
15. git log 有许多选项可以帮助你搜寻感兴趣的提交，接下来我们介绍些最常用的。我们常用 -p 选项展开显示每次提交的内容差异，用 -2 则仅显示最近的两次更新。
16. 某些时候，单词层面的对比，比行层面的对比，更加容易观察。Git 提供了 --word-diff 选项。我们可以将其添加到 git log -p 命令的后面，从而获取单词层面上的对比。在程序代码中进行单词层面的对比常常是没什么用的。不过当你需要在书籍、论文这种很大的文本文件上进行对比的时候，这个功能就显出用武之地了。
17. 另外，git log 还提供了许多摘要选项可以用，比如 --stat，仅显示简要的增改行数统计。每个提交都列出了修改过的文件，以及其中添加和移除的行数，并在最后列出所有增减行数小计。 还有个常用的 --pretty 选项，可以指定使用完全不同于默认格式的方式展示提交历史。比如用 oneline 将每个提交放在一行显示，这在提交数很大时非常有用。另外还有 short，full 和 fuller 可以用，展示的信息或多或少有些不同。
18. 
19. 
20. 
21. 有时候我们提交完了才发现漏掉了几个文件没有加，或者提交信息写错了。想要撤消刚才的提交操作，可以使用 --amend 选项重新提交此命令将使用当前的暂存区域快照提交。如果刚才提交完没有作任何改动，直接运行此命令的话，相当于有机会重新编辑提交说明，但将要提交的文件快照和之前的一样。启动文本编辑器后，会看到上次提交时的说明，编辑它确认没问题后保存退出，就会使用新的提交说明覆盖刚才失误的提交。如果刚才提交时忘了暂存某些修改，可以先补上暂存操作，然后再运行 --amend 提交

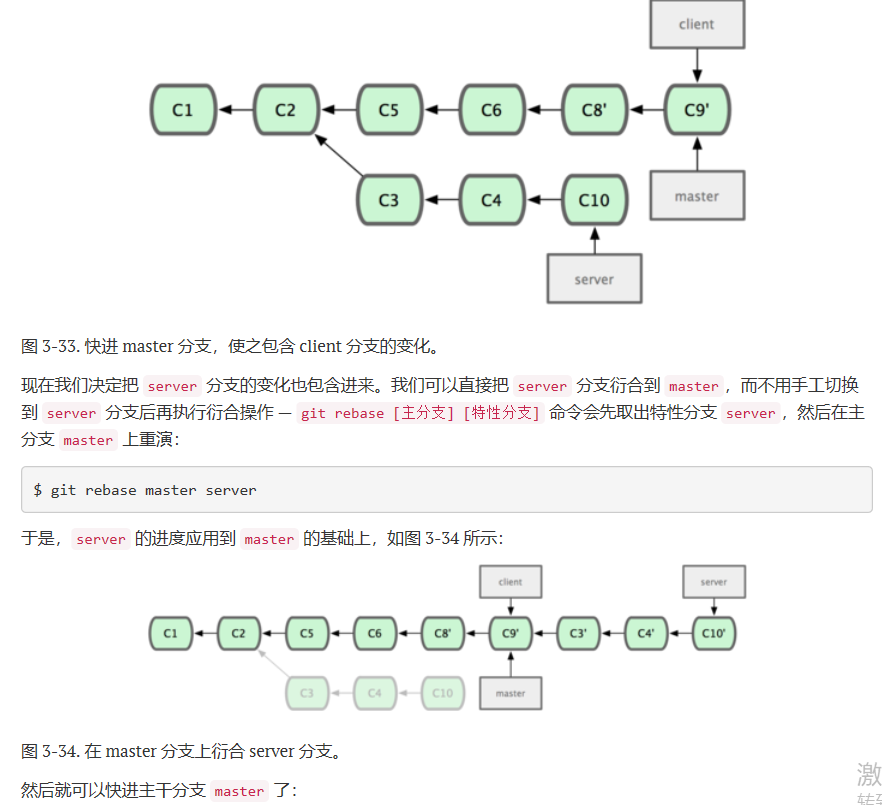


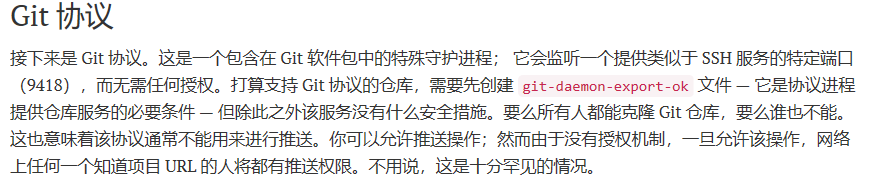
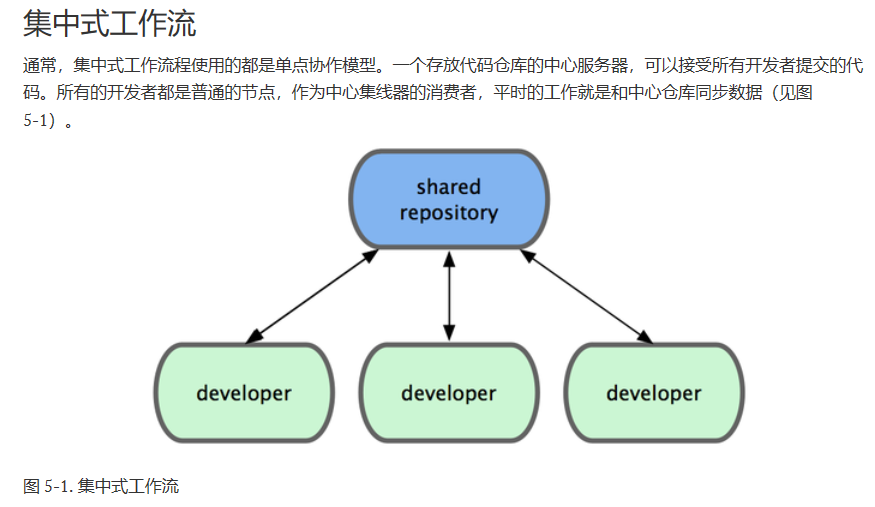
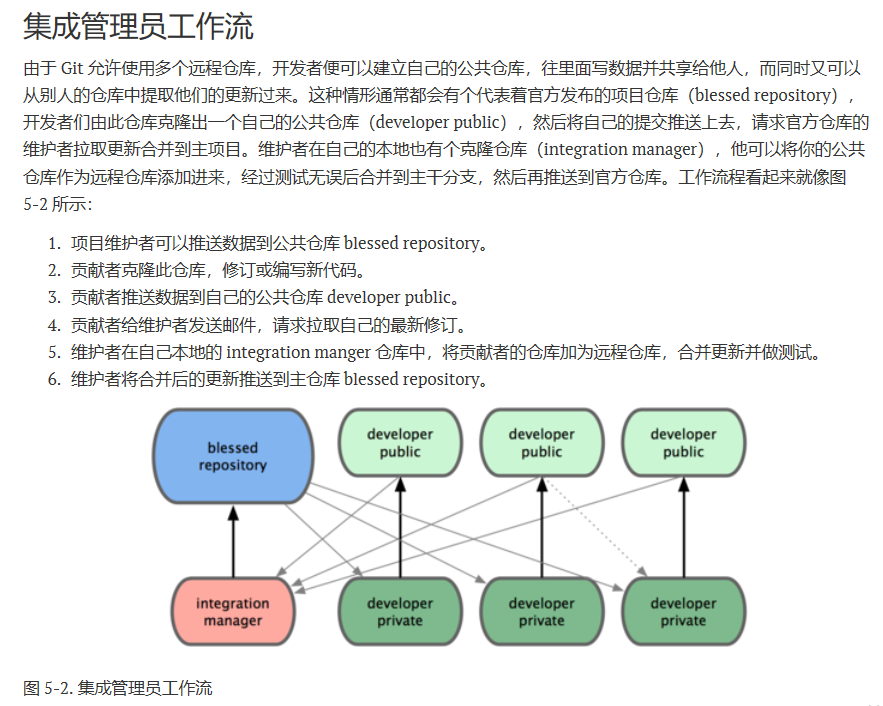
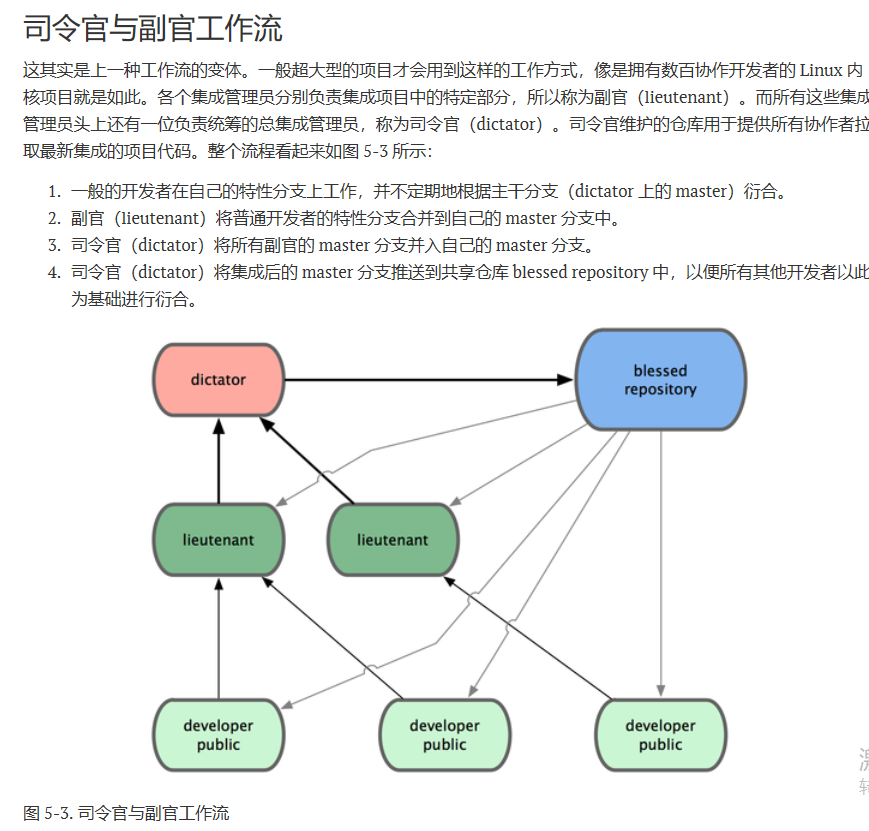
1. 该如何撤消暂存其中的一个文件呢？其实，git status 的命令输出已经告诉了我们该怎么做，可以使用 git reset HEAD <file>... 的方式取消暂存。
2. 如果觉得刚才修改完全没有必要，该如何取消修改，回到之前的状态（也就是修改之前的版本）呢？git status 同样提示了具体的撤消方法，use "git checkout -- <file>..." to discard changes in working directory。
3. 要查看当前配置有哪些远程仓库，可以用 git remote 命令，它会列出每个远程库的简短名字。在克隆完某个项目后，至少可以看到一个名为 origin 的远程库，Git 默认使用这个名字来标识你所克隆的原始仓库。也可以加上 -v 选项（译注：此为 --verbose 的简写，取首字母），显示对应的克隆地址。
4. 要添加一个新的远程仓库，可以指定一个简单的名字，以便将来引用，运行 git remote add [shortname] [url]，现在可以用字符串 比如pb 指代对应的仓库地址了。比如说，要抓取所有 Paul 有的，但本地仓库没有的信息，可以运行 git fetch pb。$ git fetch [remote-name]
5. 如果是克隆了一个仓库，此命令会自动将远程仓库归于 origin 名下。所以，git fetch origin 会抓取从你上次克隆以来别人上传到此远程仓库中的所有更新（或是上次 fetch 以来别人提交的更新）。有一点很重要，需要记住，fetch 命令只是将远端的数据拉到本地仓库，并不自动合并到当前工作分支，只有当你确实准备好了，才能手工合并。
6. 如果设置了某个分支用于跟踪某个远端仓库的分支，可以使用 git pull 命令自动抓取数据下来，然后将远端分支自动合并到本地仓库中当前分支。在日常工作中我们经常这么用，既快且好。实际上，默认情况下 git clone 命令本质上就是自动创建了本地的 master 分支用于跟踪远程仓库中的 master 分支（假设远程仓库确实有 master 分支）。所以一般我们运行 git pull，目的都是要从原始克隆的远端仓库中抓取数据后，合并到工作目录中的当前分支。
7. git push [remote-name] [branch-name]。如果要把本地的 master 分支推送到 origin 服务器上（再次说明下，克隆操作会自动使用默认的 master 和 origin 名字）。只有在所克隆的服务器上有写权限，或者同一时刻没有其他人在推数据，这条命令才会如期完成任务。如果在你推数据前，已经有其他人推送了若干更新，那你的推送操作就会被驳回。你必须先把他们的更新抓取到本地，合并到自己的项目中，然后才可以再次推送。
8. 我们可以通过命令 git remote show [remote-name] 查看某个远程仓库的详细信息



1. 在新版 Git 中可以用 git remote rename 命令修改某个远程仓库在本地的简称，碰到远端仓库服务器迁移，或者原来的克隆镜像不再使用，又或者某个参与者不再贡献代码，那么需要移除对应的远端仓库，可以运行 git remote rm 命令。
2. 列出现有标签的命令非常简单，直接运行 git tag 即可。
3. Git 使用的标签有两种类型：轻量级的（lightweight）和含附注的（annotated）。轻量级标签就像是个不会变化的分支，实际上它就是个指向特定提交对象的引用。而含附注标签，实际上是存储在仓库中的一个独立对象，它有自身的校验和信息，包含着标签的名字，电子邮件地址和日期，以及标签说明，标签本身也允许使用 GNU Privacy Guard (GPG) 来签署或验证。一般我们都建议使用含附注型的标签，以便保留相关信息；当然，如果只是临时性加注标签，或者不需要旁注额外信息，用轻量级标签也没问题。
4. 
5. 如果你有自己的私钥，还可以用 GPG 来签署标签，只需要把之前的 -a 改为 -s （译注： 取 signed 的首字母）即可，轻量级标签实际上就是一个保存着对应提交对象的校验和信息的文件。要创建这样的标签，一个 -a，-s 或 -m 选项都不用，直接给出标签名字即可。
6. 可以使用 git tag -v [tag-name] （译注：取 verify 的首字母）的方式验证已经签署的标签。此命令会调用 GPG 来验证签名，所以你需要有签署者的公钥，存放在 keyring 中，才能验证。
7. 
8. 默认情况下，git push 并不会把标签传送到远端服务器上，只有通过显式命令才能分享标签到远端仓库。其命令格式如同推送分支，运行 git push origin [tagname] 即可，如果要一次推送所有本地新增的标签上去，可以使用 --tags 选项。
9. 
10. 
11. 
12. 在 Git 中提交时，会保存一个提交（commit）对象，该对象包含一个指向暂存内容快照的指针，包含本次提交的作者等相关附属信息，包含零个或多个指向该提交对象的父对象指针：首次提交是没有直接祖先的，普通提交有一个祖先，由两个或多个分支合并产生的提交则有多个祖先。
13. 
14. 
15. 
16. 
17. 
18. 
19. 
20. 
21. 
22. 由于当前 hotfix 分支和 master 都指向相同的提交对象，所以 hotfix 已经完成了历史使命，可以删掉了。使用 git branch 的 -d 选项执行删除操作。
23. 
24. 任何包含未解决冲突的文件都会以未合并（unmerged）的状态列出。Git 会在有冲突的文件里加入标准的冲突解决标记，可以通过它们来手工定位并解决这些冲突。在解决了所有文件里的所有冲突后，运行 git add 将把它们标记为已解决状态（译注：实际上就是来一次快照保存到暂存区域。）。因为一旦暂存，就表示冲突已经解决。如果你想用一个有图形界面的工具来解决这些问题，不妨运行 git mergetool，它会调用一个可视化的合并工具并引导你解决所有冲突。
25. 
26. 
27. 
28. 
29. 
30. 
31. 
32. 
33. 
34. 





1. 一旦分支中的提交对象发布到公共仓库，就千万不要对该分支进行衍合操作。在进行衍合的时候，实际上抛弃了一些现存的提交对象而创造了一些类似但不同的新的提交对象。如果你把原来分支中的提交对象发布出去，并且其他人更新下载后在其基础上开展工作，而稍后你又用 git rebase 抛弃这些提交对象，把新的重演后的提交对象发布出去的话，你的合作者就不得不重新合并他们的工作，这样当你再次从他们那里获取内容时，提交历史就会变得一团糟。
2. 
3. 
4. 
5. 
6. 在服务器上部署Git就是用git clone --bare 命令把现有的仓库生成一个裸仓库，然后把这个裸仓库移到服务器上。
7. 
8. 
9. 
10. 
11. 
12. 