链滴

Git | 查漏补缺

作者：douniwan

原文链接：https://ld246.com/article/1582188077133

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180617.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h3 id="Git与其他的版本控制工具的差异">Git 与其他的版本控制工具的差异</h3>
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，而大多数
他系统则只关心文件内容的具体差异。
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
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
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
作一连接。Git 的工作方式就像图所示：
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3
ogfile.com/file/2020/02/image-ae24ffc5.png?imageView2/2/interlace/1/format/jpg">
<h3 id="三种状态">三种状态</h3>
对于任何一个文件，在 Git 内都只有三种状态：已提交（committed），已修改（modified）
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
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。
<h3 id="三个工作区域">三个工作区域</h3>
Git 的本地数据目录，工作目录以及暂存区域。
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3
ogfile.com/file/2020/02/image-3ab28a99.png?imageView2/2/interlace/1/format/jpg">
<h3 id="工作流程">工作流程</h3>

在工作目录中修改某些文件。

对这些修改了的文件作快照，并保存到暂存区域。

提交更新，将保存在暂存区域的文件快照转储到 git 目录中。

所以，我们可以从文件所处的位置来判断状态：如果是 git 目录中保存着的特定版本文件，就属
已提交状态；如果作了修改并已放入暂存区域，就属于已暂存状态；如果自上次取出后，作了修改但
没有放到暂存区域，就是已修改状态。
<h3 id="git-diff">git diff</h3>
<code>git diff</code> 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，而不是这
工作和上次提交之间的差异。所以有时候一下子暂存了所有更新过的文件后，运行 git diff 后却什么
没有，就是这个原因。
<h3 id="git-commit--a--m--提交说明-">git commit -a -m "提交说明"</h3>
在 <code>git commit</code> 加上-a 参数可以使得被修改的文件（前提是该文件已经被追踪
）跳过暂存区直接提交。
<h3 id="移除文件">移除文件</h3>
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 <code>git rm</code> 命令完成此项工作，并连带从工作目录中删除指定的文
，这样以后就不会出现在未跟踪文件清单中了。不要手工的去删除一个文件，这样删除后 git 会显示
文件未被暂存，此时还需一次 <code>git add</code> 或执行 <code>git rm</code>。
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
首字母），以防误删除文件后丢失修改的内容。
另外一种情况是，我们想把文件从 Git 仓库中删除（亦即从暂存区域移除），但仍然希望保留在
前工作目录中。换句话说，仅是从跟踪清单中删除。比如一些大型日志文件或者一堆 .a 编译文件，
小心纳入仓库后，要移除跟踪但不删除文件，以便稍后在 .gitignore 文件中补上，用 --cached 选项
可：
<code>git rm --cached 已暂存文件</code>
<h3 id="移动文件">移动文件</h3>

利用 <code>git mv file filemv</code> 命令去移动文件，它相当于这几步：
<pre><code class="language-git highlight-chroma"><span cla
s="highlight-cl">mv file filemv //先移动文件
git rm file //移除
前的文件
git add filemv   /
追踪移动后的文件
</code></pre>
<h3 id="git-log">git log</h3>
利用 <code>git log</code> 查看提交历史，没有任何参数的时候展示所有历史，最近的更新
于最上面。

-p： 展开显示每次提交的内容差异
-n： 仅显示最近的 n 次更新
--stat：仅显示简要的增改行数统计

<h3 id="git-commit---amend">git commit --amend</h3>

如果某次 commit 以后发现提交的说明写错，那么就可以执行 <code>git commit --amend</
ode> 对该次提交的说明进行修改
如果某次 commit 以后发现，此次提交漏了几个还没有追踪的文件（本来想要一起提交）那么：
以先用 <code>git add 文件</code>,再用 <code>git commit --amend</code>。这样三条命
（算上第一个 commit）最终得到一个提交，第二个提交命令修正了第一个的提交内容

<h3 id="git-reset-HEAD">git reset HEAD</h3>
利用 <code>git reset HEAD 文件</code> 可以取消已经暂存的文件为未暂存。
<h3 id="git-checkout---">git checkout --</h3>
利用 <code> git checkout -- 文件</code> 可以取消已经修改的文件为未修改。（回到上一
提交的版本）
<h3 id="添加远程仓库">添加远程仓库</h3>
<code>git remote add [shortname] [url]</code>
为撒我以前一直以为[shortname] 是固定的 origin???!!!!
<h3 id="协议">协议</h3>
Git 可以使用四种主要的协议来传输数据：本地传输，SSH 协议，Git 协议和 HTTP 协议。

本地协议
最基础的就是 本地协议(Local protocol) 了，远程仓库在该协议中就是硬盘上的另一个目录。这常见
团队每一个成员都对一个共享的文件系统(例如 NFS)拥有访问权，抑或比较少见的多人共用同一台电
的时候。后者不是很理想，因为你所有的代码仓库实例都储存在同一台电脑里，增加了灾难性数据损
的可能性。如果你使用一个共享的文件系统，就可以在一个本地仓库里克隆，推送和获取。要从这样
仓库里克隆或者将其作为远程仓库添加现有工程里，可以用指向该仓库的路径作为 URL。

SSH 协议
Git 使用的传输协议中最常见的可能就是 SSH 了。这是因为大多数环境已经支持通过 SSH 对服务器
访问——即使还没有，也很容易架设。SSH 也是唯一一个同时便于读和写操作的网络协议。另外两
网络协议（HTTP 和 Git）通常都是只读的，所以虽然二者对大多数人都可用，但执行写操作时还是
要 SSH。SSH 同时也是一个验证授权的网络协议；而因为其普遍性，通常也很容易架设和使用。

Git 协议
这是一个包含在 Git 软件包中的特殊守护进程； 它会监听一个提供类似于 SSH 服务的特定端口（941），而无需任何授权。用 Git 协议运营仓库，你需要创建 git-export-daemon-ok 文件——它是协进程提供仓库服务的必要条件——但除此之外该服务没有什么安全措施。要么所有人都能克隆 Git 仓，要么谁也不能。这也意味着该协议通常不能用来进行推送。你可以允许推送操作；然而由于没有授机制，一旦允许该操作，网络上任何一个知道项目 URL 的人将都有推送权限。不用说，这是十分罕的情况。

优点
Git 协议是现存最快的传输协议。如果你在提供一个有很大访问量的公共项目，或者一个不需要对读作进行授权的庞大项目，架设一个 Git 守护进程来供应仓库是个不错的选择。它使用与 SSH 协议相的数据传输机制，但省去了加密和授权的开销。
缺点
Git 协议消极的一面是缺少授权机制。用 Git 协议作为访问项目的唯一方法通常是不可取的。一般做是，同时提供 SSH 接口，让几个开发者拥有推送（写）权限，其他人通过 <code>git://</code> 有只读权限。 Git 协议可能也是最难架设的协议。它要求有单独的守护进程，需要定制,而这就没那么易近人了。该协议还要求防火墙开放 9418 端口，而企业级防火墙一般不允许对这个非标准端口的访。大型企业级防火墙通常会封锁这个少见的端口。

HTTP/S 协议
HTTP 或 HTTPS 协议的优美之处在于架设的简便性。基本上， 只
需要把 Git 的纯仓库文件放在 HTTP 的文件根目录下，配置一个特定的 post-update 挂钩（hook）就搞定了（Git 挂钩的细节见第七章）。从此，每个能访问 Git 仓库所在服务器上的 web 服务的人都以进行克隆操作。

优点
使用 HTTP 协议的好处是易于架设
缺点
HTTP 协议的消极面在于，相对来说客户端效率更低。克隆或者下载仓库内容可能会花费更多时间，且 HTTP 传输的体积和网络开销比其他任何一个协议都大。因为它没有按需供应的能力——传输过程没有服务端的动态计算——因而 HTTP 协议经常会被称为 傻瓜(dumb) 协议。

