Git 使用 - 1. 文件操作和历史查看

http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif (2010-07-08 11:01:20)

[[http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif](javascript:;)转载▼](javascript:;)

|  |  |
| --- | --- |
|  | 分类： [Git](http://blog.sina.com.cn/s/articlelist_1726052317_6_1.html) |

**【简介】**

**版本控制系统**（VCS, Version Control System） 可以分为**集中式**（CVCS, Centralized Version Control System, 如 CVS, Subversion）和**分布式**（DVCS, Distributed Version Control System，如 Git 等）版本控制系统。

传统的集中式版本控制系统，本地只保存代码库的一个版本拷贝。 所有历史版本都保存在服务器。GIT 与之最大的不同是，本地不仅保存一个快照，而且保存着整个代码库（repository）。因此它可以“离线”工作。

在 Window 上安装： http://code.google.com/p/msysgit

GIT 起源于 LINUX 开发。Linux 开发早期（1992－2002）没有使用版本控制工具，直到 2002 年开始使用一个非开源的工具 BitKeeper。一开始这个工具还给 Linux 免费使用，后来想收费了，充满自由精神 Linux 成员们肯定不会屈服于这个要挟，一怒之下开发了一个自由的版本控制工具。2005 年，GIT 诞生 。

**参考：**官网：<http://git-scm.com/>

     手册：<http://www.kernel.org/pub/software/scm/git/docs/>

     参考：<http://gitref.org/>

     ProGit: [http://progit.org/book](http://progit.org/book/zh/)/

**【GIT 基础】**

GIT 使用 SHA-1 哈希码（40个字符）来标识提交，同时保证本次提交后整体（一个快照）的完整性。

**文件状态：**

文件状态分为：**未跟踪** (untracked) 和已跟踪 (tracked)，已跟踪又分为三种状态： **已暂存**（staged）**，已修改**（modified），**已提交**（committed）

一般过程如下：

1） 新建文件，该文件状态为“**未跟踪**”，位于**工作区**；

2） 用 git add a.txt  加入该文件，状态变为已跟踪的“**已暂存**”，位于**暂存区**；

3） 用 git commit a.txt -m "ha"  提交该文件，状态变为“**已提交**”，位于代码库（repository ）

或者，如果存在修改的情况，增加如下2.1和2.2两个步骤：

1） 新建文件，该文件状态为“**未跟踪**”，位于**工作区**；

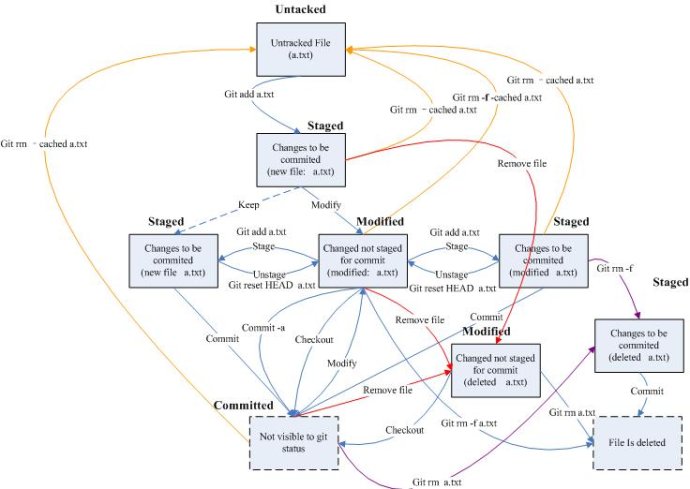
2） 用 git add a.txt  加入该文件，状态变为已跟踪的“**已暂存**”，位于**暂存区**；

2.1）编辑该文件并保存，状态变为“**已修改**”，位于**工作区**。（注意，位于暂存区的文件独立存在！）

2.2） 用 git add a.txt  加入该文件，状态变为“**已暂存**”，位于**暂存区**。（注意，原暂存区的文件被覆盖！）

3） 用 git commit a.txt -m "ha"  提交该文件，状态变为“**已提交**”，位于代码库（repository ）

考虑更广泛的情况，状态迁移图如下所示：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=66e177dd0100j4rk&url=http://s5.sinaimg.cn/orignal/66e177ddta2cfbed5b5b4)

**注意：**用 git status 查看目前所有文件的状态。

**GIT 配置：**

三种**配置范围类型：**

1）所有用户 --system，   存在 /etc/gitconfig 文件中。(对windows来说，是 msysgit 的安装目录)

2）本用户 --global，    存在 ~/.gitconfig  文件中。（对windows 来说，是 C:\Documents and Settings\$USER）

3）本项目               存在 .git/config 文件中。

**注意：**后面的重载前面的。例如：用--global设置，可以在所有项目中使用这个设置；如果有一个项目你想使用一个特别的设置，就可以使用不带参数的 git config 重新设置，则它只作用于这个项目。如果用 git config --list 查看这些设置，可能会列出多个，但最后那个起作用。

**用户信息**

$ git config --global user.name "John Doe"  
$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

**文本编辑器**

$ git config --global core.editor emacs

**查看配置信息**

$ git config --list

或者只看一个信息：

$ git config <key>

**【常用操作】**

（GIT 1.7.4 on  Windows)

获取帮助：                       $ git help <verb>

初始化仓库：                     $ git init

*（说明： 如果该目录有文件，则都会处于“未跟踪”状态的。）*

克隆仓库：                       $ git clone git://github.com/schacon/grit.git

克隆仓库，并换个名字：           $ git clone git://github.com/schacon/grit.git mygrit

跟踪一个新文件：                 $ git add readme.txt

*说明：*

*1） 跟踪之后，该文件状态是“已暂存”的。 （Changes to be committed:）*

*2）修改一个已暂存的文件，该文件会出现在两个记录区中。*

*3） 如果跟踪错了，想把他删除（不删除工作区的），则用 git rm --cached readme.txt。状态变成“未跟踪”，如果该文件已经修改了，则加  -f 参数强行删除暂存区的文件（已修改的文件不被覆盖）。*

跟踪一组新文件：                 $ git add \*.txt

跟踪一个目录中的所有文件：       $ git add mydir

*说明：如果该目录下有子目录，则进行递归操作。*

修改之后，暂存一个文件：                           $ git add readme.txt

*说明：取消已暂存的文件（已暂存到已修改）：         $ git reset HEAD readme.txt*

**忽略某些文件**：                   在.gitignore 文件中加入下面两行:    \*.[oa]    \*~

\*.a       # 忽略所有 .a 结尾的文件  
!lib.a    # 但 lib.a 除外  
/TODO     # 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO  
build/    # 忽略 build/ 目录下的所有文件  
doc/\*.txt # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt

**注意：**.gitignore 文件放在工程的根目录即可，但是要把它用 git add 加入跟踪或者提交。

**比较：**

查看尚未暂存的更新：                   $ git diff

*说明：比较“已修改”和“已暂存（或已提交）”。*

查看尚未提交的更新:                    $ git diff --cached

      或                               $ git diff --staged

*说明：比较“已暂存”和已提交。*

**提交：**

提交更新（只更新暂存中的）：           $ git commit

提交更新（只更新暂存中的）：           $ git commit -m "task12, added implementation for usr mrg."

直接提交更新(更新暂存中的和已修改的)： $ git commit -a -m "task12, added implementation forg."

从文件中取注释：                       $ git commit --file notefile

**重新获取：**

抛弃已修改，用已提交覆盖               $ git checkout A.java

*说明：此命令对已暂存文件没作用。*

**删除文件：**

从工作区可以直接删除文件，这是运行 git status， 显示：

  Changes not staged for commit:

  (use "git add/rm <file>..." to update what will be committed)

  (use "git checkout -- <file>..." to discard changes in working directory)

      deleted:    a

可以用 git rm 删除 a。注意：如果 a 是已提交的文件，则还需要运行 git commit 以确保在代码库中删除。如果该文件尚未提交，则无需运行git commit。

也可以直接使用 git rm 删除文件，但如何文件处于“已修改” 则需要加 -f

移除已跟踪文件，恢复到“未跟踪”       $ git rm --cached A.java

*说明：如果文件已修改，则需 -f ，并且不覆盖修改过的内容。*

强行移除修改后文件(从暂存区和工作区中删除)：      $ git rm -f a.a

移除目录下的所有文件（递归）：                    $ git rm log/\\*.log

移除目录下的所有文件（无递归）：                  $ git rm log/\*.log

改名（只改工作区和暂存区）：                      $ git mv file1 file2

相当于：                                          $ mv README.txt README  
                                                  $ git rm README.txt  
                                                  $ git add README

**历史查看 ：**

查看提交历史(全部）：                             $ git log

查看提交历史，并显示统计信息：                    $ git log --stat

查看提交历史并查看差异：                          $ git log -p

查看最近2次提交历史并查看差异：                   $ git log -p -2

查看最近2周内提交历史：                           $ git log --since=2.weeks

查看某个时刻之后的提交历史：                      $ git log --since="2008-09-14"

查看某个时刻以前的提交历史：                      $ git log --until="2008-09-14"

查看某个作者的提交历史：                          $ git log --author="stupid"

其他：

-(n) 仅显示最近的 n 条提交  
--since, --after 仅显示指定时间之后的提交。  
--until, --before 仅显示指定时间之前的提交。  
--author 仅显示指定作者相关的提交。  
--committer 仅显示指定提交者相关的提交。

例如：

$ git log --pretty="%h:%s" --author=gitster --since="2008-10-01" \  
   --before="2008-11-01" --no-merges -- t/

查看提交历史，并单行显示：                        $ git log --pretty=oneline

查看提交历史，并格式化显示：                      $ git log --pretty=format:"%h - %an, %ar : %s"

例如：

$ git log --pretty=format:"%h - %an, %ar : %s"  
ca82a6d - Scott Chacon, 11 months ago : changed the version number  
085bb3b - Scott Chacon, 11 months ago : removed unnecessary test code  
a11bef0 - Scott Chacon, 11 months ago : first commit

%H 提交对象（commit）的完整哈希字串  
%h 提交对象的简短哈希字串  
%T 树对象（tree）的完整哈希字串  
%t 树对象的简短哈希字串  
%P 父对象（parent）的完整哈希字串  
%p 父对象的简短哈希字串  
%an 作者（author）的名字  
� 作者的电子邮件地址  
� 作者修订日期（可以用 -date= 选项定制格式）  
%ar 作者修订日期，按多久以前的方式显示  
%cn 提交者(committer)的名字  
� 提交者的电子邮件地址  
� 提交日期  
%cr 提交日期，按多久以前的方式显示  
%s 提交说明

查看提交历史，并图形化显示：               $ git log --pretty=format:"%h %s" --graph

例如：

$ git log --pretty=format:"%h %s" --graph  
\* 2d3acf9 ignore errors from SIGCHLD on trap  
\*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit  
|\  
| \* 420eac9 Added a method for getting the current branch.  
\* | 30e367c timeout code and tests  
\* | 5a09431 add timeout protection to grit  
\* | e1193f8 support for heads with slashes in them  
|/  
\* d6016bc require time for xmlschema  
\*  11d191e Merge branch 'defunkt' into local

其它：

-p 按补丁格式显示每个更新之间的差异。  
--stat 显示每次更新的文件修改统计信息。  
--shortstat 只显示 --stat 中最后的行数修改添加移除统计。  
--name-only 仅在提交信息后显示已修改的文件清单。  
--name-status 显示新增、修改、删除的文件清单。  
--abbrev-commit 仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。  
--relative-date 使用较短的相对时间显示（比如，“2 weeks ago”）。  
--graph 显示 ASCII 图形表示的分支合并历史。  
--pretty 使用其他格式显示历史提交信息。可用的选项包括 oneline，short，full，fuller 和 format（后跟指定格式）。

**最重要的是，**如果你觉着上面这些**查看历史**的命令太难用，你可以使用图形化工具：gitk，一切都清晰了。