GIT Tools

Git 是 Linux Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的分布式版本控制软件， 它不同于Subversion、CVS这样的集中式版本控制系统。

在集中式版本控制系统中只有一个仓库（repository），许多个工作目录（working copy）， 而像Git这样的分布式版本控制系统中（其他主要的分布式版本控制系统还有BitKeeper、Mercurial、 GNU Arch、Bazaar、Darcs、SVK、Monotone等），每一个工作目录都包含一个完整仓库， 它们可以支持离线工作，本地提交可以稍后提交到服务器上。

分布式系统理论上也比集中式的单服务器系统更健壮，单服务器系统一旦服务器出现问题整个系统就不能运行了，分布式系统通常不会因为一两个节点而受到影响。

Git add

```
git add .                       # 将资料先暂存到staging area, add 之后再新增的资料,
                                  于此次commit 不会含在里面.
git add filename                # 增加文件到git库中
git add modify-file             # 修改过的档案, 也要add. (不然commit 要加上-a 的参数)
git add -u                      # 只加修改过的档案, 新增的档案不加入.
git add -i                      # 进入互动模式
```

Git rm

```
git rm filename                 # 将文件从git库中删除,这只是删除本地的文件,需要commit,还要push到远程git库中.
```

Git mv

```
git mv filename new-filename    # 将文件改名
```

Git status

```
git status                      # 看目前档案的状态
```

Git Commit

```
git commit
git commit -m 'commit message'    # 修改过的文件进行提交,但必须先要执行 git-add.
git commit -a -m 'commit -message'# 将所有修改过得档案都commit, 不需要执行 git-add命令.-a参数就是已经自动执行git-add命令
git commit -a -v                  # -v 可以看到档案哪些内容有被更改, -a 把所有修改的档案都commit
```

Git new-branch

```
git branch                      # 列出目前有多少branch
```

```
git branch                          # 列出目前有多少branch
git branch new-branch               # 产生新的branch (名称: new-branch), 若没有特别
                                      指定, 会由目前所在的branch / master 直接复制一份.
git branch new-branch master        # 由master 产生新的branch(new-branch)
git branch new-branch tag v1        # 由tag(v1) 产生新的branch(new-branch)
git branch -d new-branch            # 删除new-branch
git branch -D new-branch            # 强制删除new-branch
git checkout -b new-branch test     # 产生新的branch, 并同时切换过去new-branch
                                      与remote repository 有关
git branch -r                       # 列出所有Repository branch
git branch -a                       # 列出所有branch
git branch -av                      # 列出所有分支,还有提交的哈希值.
git branch -rD origin/name                  1       # 删除远程分支或仓库分支.
git push origin :heads/name     2
```

Git checkout-branch

```
git checkout branch-name            # 切换到branch-name
git checkout master                 # 切换到master
git checkout -b new-branch master # 从master 建立新的new-branch, 并同时切换过去new-branch
git checkout -b new-branch          # 由现在的环境为基础, 建立新的branch
git checkout -b new-branch tag v1 # 从tag v1 checkout文档,并新建一个branch.
git checkout -b new-branch origin # 基于origin 的基础, 建立新的branch
git checkout filename               # 还原档案到Repository 状态
git checkout HEAD .                 # 将所有档案都checkout 出来(最后一次commit 的版本),
                                      注意, 若有修改的档案都会被还原到上一版. (git checkout -f 亦可)
git checkout xxxx .                 # 将所有档案都checkout 出来(xxxx commit 的版本, xxxx
                                      是commit 的编号前四码), 注意, 若有修改的档案都会被还原到上一版.
git checkout -- *                   # 恢复到上一次Commit 的状态(* 改成档名, 就可以只恢复那个档案)
```

Git diff

```
git diff master                     # 与Master 有哪些资料不同
git diff --cached                   # 比较staging area 跟本来的Repository
git diff tag1 tag2                  # tag1, 与tag2 的diff
git diff tag1:file1 tag2:file2      # tag1, 与tag2 的file1, file2 的diff
git diff                            # 比较目前位置与staging area
git diff --cached                   # 比较staging area 与Repository 差异
git diff HEAD                       # 比较目前位置与Repository 差别
git diff new-branch                 # 比较目前位置与branch(new-branch) 的差别
git diff --stat
```

Git tag

```
git tag v1 ebff                     # log 是commit ebff810c461a 的内容, 设定简短好记得Tag: v1
git tag -d v1                       # 把tag v1删掉.
```

Git log

```
git log                             # 将所有log 秀出
git log --all                       # 秀出所有的log (含branch)
git log -p                          # 将所有log 和修改过得档案内容列出
git log -p filename                 # 将此档案的commit log 和修改档案内容差异部份列出
git log --name-only                 # 列出此次log 有哪些档案被修改
git log --stat --summary            # 查每个版本间的更动档案和行数
git log filename                    # 这个档案的所有log
git log directory                   # 这个目录的所有log
git log -S'foo()'                   # log 里面有foo() 这字串的.
git log --no-merges                 # 不要秀出merge 的log
git log --since="2 weeks ago"       # 最后这2周的log
git log --pretty=oneline            # 秀log 的方式
git log --pretty=short              # 秀log 的方式

git log --pretty=format:'%h was %an, %ar, message: %s'
git log --pretty=format:'%h : %s' --graph             # 会有简单的文字图形化, 分支等.
git log --pretty=format:'%h : %s' --topo-order --graph # 依照主分支排序
git log --pretty=format:'%h : %s' --date-order --graph # 依照时间排序
```

```
git log --pretty=format:"%H : %s" --date-order --graph # 依照时间排序
```

Git show

```
git show ebff                    # 查log 是commit ebff81 的内容
git show v1                      # 查tag:v1 的修改内容
git show v1:test.txt             # 查tag:v1 的test.txt 档案修改内容
git show HEAD                    # 此版本修改的资料
git show HEAD^                   # 前一版修改的资料
git show HEAD^^                  # 前前一版修改的资料
git show HEAD~4                  # 前前前前一版修改的资料
```

Git reset

```
git reset --hard HEAD            # 还原到最前面
git reset --hard HEAD~3
git reset --soft HEAD~3
git reset HEAD filename          # 从staging area 状态回到unstaging 或untracked (档案内容并不会改变)
```

Git grep

```
git grep "te" v1                 # 查v1 是否有"te" 的字串
git grep "te"                    # 查现在版本是否有"te" 的字串
```

Git stash

```
git stash                        # 丢进暂存区
git stash list                   # 列出所有暂存区的资料
git stash pop                    # 取出最新的一笔, 并移除.
git stash apply                  # 取出最新的一笔stash 暂存资料. 但是stash 资料不移除
git stash clear                  # 把stash 都清掉
```

Git merge

```
git merge
git merge master
git merge new-branch
```

下述转载自:Git版本控制系统(2)开branch分支和操作远端

> Straight merge 预设的合并模式，会有全部的被合并的branch commits 记录加上一个merge-commit，看线图会有两条 Parents 线，并保留所有commit log。 Squashed commit 压缩成只有一个merge-commit，不会有被合并的log。SVN 的 merge 即是如此。 cherry-pick 只合并指定的commit rebase 变更branch 的分支点：找到要合并的两个branch 的共同的祖先，然后先只用要被merge 的branch 来commit 一遍，然后再用目前branch 再commit 上去。这方式仅适合还没分享给别人的local branch，因为等于砍掉重练commit log。

指令操作:

```
git merge <branch_name>          # 合并另一个branch, 若没有conflict 冲突会直接commit。
                                   若需要解决冲突则会再多一个commit。
git merge --squash <branch_name> # 将另一个branch 的commit 合并为一笔, 特别适合需要做实
                                   验的fixes bug 或new feature, 最后只留结果。合并完不会帮你先commit。
git cherry-pick 321d76f          # 只合并特定其中一个commit。如果要合并多个, 可以加上-n
                                   指令就不会先帮你commit, 这样可以多pick几个要合并的commit, 最后再git commit 即可。
```

Git blame

```
git blame filename                    # 关于此档案的所有commit 纪录
```

Git ls-files

```
git ls-files -d                       # 查看已删除的档案
git ls-files -d | xargs git checkout --
                                      # 将已删除的档案还原
```

Git gc

```
git gc                                # 整理前和整理后的差异, 可由: git count-objects 看到.
git fsck --full
```

Git patch

```
git format-patch -1                   # 提取最近的一个提交,生成patch文件.
git format-patch  master              # 如果站在develop分支上,将与master不同的分支列出来

git am <patch name>                   # 针对format-patch生成的patch.

git apply <patch name>                # 针对传统的diff命令生成的patch.但可以也打format-patch.
```

Git revert

```
git revert HEAD                       # 回到前一次commit 的状态
git revert HEAD^                      # 回到前前一次commit 的状态
git reset HEAD filename               # 从staging area 状态回到unstaging 或untracked (档案内容并不会改变)
git checkout filename                 # 从unstaging 状态回到最初Repository 的档案(档案内容变回修改前)
```

Git config

```
git config color.ui true              # 打开所有颜色显示
```

Git Rollback

```
git reset --soft HEAD^
                                      # 对文件进行编辑,完成后执行:
git add filename
git commit -m 'rollback'

git commit -a --amend                 # 可以用一个命令搞定.
```

Git remote

与远端Repository相关:

```
git remote
git remote add new-branch http://git.example.com.tw/project.git
                                    # 增加远端Repository 的branch(origin -> project)
git remote show                     # 秀出现在有多少Repository
git remote rm new-branch            # 删掉远程分支
git remote update                   # 更新所有Repository branch
git branch -r                       # 列出所有Repository branch
```

抓取/切换Repository的branch:

```
git fetch origin
git checkout --track -b reps-branch origin/reps-branch
                                    # 抓取reps-branch, 并将此branch 建立于local 的reps-branch
```

将本地分支同步到远程的Repository的branch,如果远程没有分支将会新建一个分支:

```
git push origin [local-branch-name]:[remote-branch-name]
```

删除远程的Repository中的branch分支:

```
git branch -rD [branch-name]
git push origin :heads/[branch-name]
```

增加远程服务器

```
git remote add master git-url
git fetch master
git push master master
git pull origin
```

更新上次commit信息

在实际的使用中,会常常遇到这个的问题.我们已经commit了,但没有push到远程git库里.但是又修改了,只需要将这次的修改增加到上次我们commit里. 可以用下面的命令:

```
git add modified filename
git commint --amend
```

git cherry-pick

将别的分支上的提交拿到自己的分支上来.:

```
git cherry-pick  [HASH值]               # 将这个patch合并到这个分支上来,重新生成HASH值.
```

git clean

清除没有在GIT库中的文件.:

```
git clean -nd                       # 查看当前多的文件和目录或将要被删除的文件和目录.
git clean -fd                       # 将多的文件和目录从git目录里删除.
```

参考文档

[GIT] http://git-scm.com/

[Pro git] http://git-scm.com/2010/06/09/pro-git-zh.html

[Chinese Wiki] http://www.baike.com/wiki/GIT

讨论