Git笔记

# 基本介绍

Git是一个支持分布式的版本管理系统。

Git包含本地仓库和远程仓库。

|  |
| --- |
| 图 1 |

如图1所示。Git本地仓库中包含两个部分。工作区和版本库，工作区对应着我们实际操作的文件和目录，版本库对应着我们本地仓库的.git目录。版本库中包含两个分支，一个是暂存分支stage和当前分支master。Head指向master分支。

# 本地仓库

## 创建版本库

### init

|  |
| --- |
| 图 2 |

如图2所示。通过git init命令可以在目录中创建本地仓库。

### add

|  |
| --- |
| 图 3 |

如图3所示。通过git add将文件或者目录添加到暂存分支中stage中。

### commit

|  |
| --- |
| 图 4 |

如图4所示。通过git commit将暂存stage中add的所有动态变更同步到当前分支master上。m参数是在提交时的备注信息。

add指令可以将工作区中的变更分多次进行提交到暂存分支stage中。而commit指令会将暂存分支stage中的所有变更都一次性同步到当前分支master中。如图1所示。

### status

|  |
| --- |
| 图 5 |

如图5所示。git status可以查看当前本地仓库中所有文件的状态，可以显示那些文件未被add，哪些文件未被commit。图5表示当前仓库中与当前分支没有差异。

|  |
| --- |
| 图 6 |

如图6所示，表示readme.txt在工作区并没有提交到暂存分支stage中。提示通过add指令进行添加。

|  |
| --- |
| 图 7 |

如图7所示。表示工作区中有修改readme.txt但是没有同步到暂存分支stage中。提示要么通过add将变更同步到暂存分支，要么通过checkout – file的形式撤销工作区的修改。

|  |
| --- |
| 图 8 |

如图8所示。表示readme.txt表示变更需要提交，当前分支master中没有。即暂存分支stage中有readme.txt而当前分支master中并没有，需要通过commit进行同步到master分支。

### diff

|  |
| --- |
| 图 9 |

如图9所示。通过diff指令可以查看文件与当前暂存分支stage的区别。

如果需要显示与当前分支master的区别可以使用HEAD参数。如图10所示。

|  |
| --- |
| 图 10 |

### log（版本历史记录）

|  |
| --- |
| 图 11 |

如图11所示。通过log指令可以看到当前分支master已commite的版本节点记录。每个节点都有一个commite属性，对应着本次提交的唯一标识，可用于后面的版本控制。每个节点后面都有该节点的备注信息。对应每次commite的m参数。

通过log可以看到当前master有两个版本节点。HEAD指向当前版本节点。而HEAD^标识当前版本的上一个版本，HEAD^^标识上一个版本的上一个版本。当需要追溯到之前的若干个版本时可以用HEAD~n，即标识当前版本之前的100个版本，

### reflog（所有操作历史记录）

|  |
| --- |
| 图 12 |

如图12所示。可以查看到本地仓库所有的操作历史记录。可以看到每一个commit 操作的commit id。

## 版本控制

版本控制可以通过两种方式进行控制。通过HEAD和通过commit id。

### 通过HEAD的方式。

|  |
| --- |
| 图 13 |

如图13所示。HEAD^标识当前版本之前的版本。通过reset将版本回退到了当前版本之前的版本。然后查看版本历史记录，确实回到了之前的版本。

### 通过commit id的方式。

|  |
| --- |
| 图 14 |

如图14所示。通过log指令可以看到版本历史记录中每次版本节点的commit id。再利用reset通过commit id来切换到指定的版本节点。

### 总结

|  |
| --- |
| 图 15 |

如图15所示。版本历史有append GPL、add distributed和wrote a readme file。通过reset将版本从append GPL切换到add distributed仅仅是将HEAD指针变更了指向，所以git的版本切换的效率相当高。同时并不会丢失最新的版本append GPL。如果还想要再回到append GPL。仅仅需要通过commit id再次reset即可。但是在add distributed版本下是没有append GPL的，故无法获取到append GPL的commit id信息。需要通过reflog指令来查看本地仓库所有的操作历史。

## 撤销修改

撤销修改有两种情况。撤销工作区的修改和撤销暂存区的修改。

### 撤销工作区的修改

|  |
| --- |
| 图 16 |

如图16所示。通过checkout – file指令撤销工作区的修改。这种撤销只能够撤销工作区的修改，还未提交到暂存分支stage的修改。同时checkout – file 其中的参数--是必须要加的，否则checkout就是切换分支的操作了。

### 撤销暂存区的修改

|  |
| --- |
| 图 17 |

如图17所示。通过reset HEAD的方式直接撤销了暂存分支stage中的修改。

## 删除文件

|  |
| --- |
| 图 18 |

如图18所示。删除文件如果需要同步删除当前分支中的文件。通过git rm 删除暂存分支stage中的文件信息，再commit到当前分支master中。

# 远程仓库

## 生成SSH密钥

|  |
| --- |
| 图 19 |

需要在用户主目录中查看是否有.ssh目录，没有则表示未生成ssh密钥。需要通过如图19所示的ssh-keygen来生成ssh密钥。之后保存。

## 克隆远程仓库

|  |
| --- |
| 图 20 |

如图20所示。通过git clone来克隆远程仓库到本地。

# 分支管理

|  |
| --- |
| 图 21 |

如图21所示。整个主分支就是一条时间线。当不断的提交操作，整个时间线会越来越长，而master指向最新的提交。而HEAD指向master。表示当前版本。

|  |
| --- |
| 图 22 |

如图22所示。当创建新的分支时。仅仅时创建一个新的指针也指向最新的提交。切换到新的分支后会将HEAD指向dev。可以发现，git的分支创建和切换都是相当高效的。仅仅时需要变更指针引用就好。

|  |
| --- |
| 图 23 |

如图23所示。在新分支进行提交后，master还是指向主分支的最新提交不进行变动。dev则指向当前新分支的最新提交，HEAD指向dev表示当前版本。

|  |
| --- |
| 图 24 |

如图24所示。在分支合并时，如果没有冲突，仅仅时将指向主分支最新提交的master也指向dev分支的最新提交，同时将HEAD重新指向master就好。所以git的分支合并也是相当高效的。

## 查看分支

|  |
| --- |
| 图 25 |

如图25所示。可以通过git branch来查看当前仓库所包含的所有分支。其中\*表示当前使用的分支。

## 创建分支

创建分支有两种方式，通过branch和checkout都可以进行创建分支。

### 通过branch创建分支。

|  |
| --- |
| 图 26 |

如图26所示。通过branch命令进行创建分支。

### 通过checkout创建并切换分支

|  |
| --- |
| 图 27 |

如图27所示。通过checkout –b 指令创建新分支同时切换的新分支。

## 切换分支

|  |
| --- |
| 图 28 |

如图28所示。通过checkout指令来切换分支。

## 合并分支

|  |
| --- |
| 图 29 |

如图29所示。通过merge指令来将当前分支与目标分支进行合并。在合并之前需要确保两个分支没有冲突。

## 解决冲突

|  |
| --- |
| 图 30 |

如图30所示。当在合并分支的时候出现差异时，合并会失败。需要通过git status来查看有冲突的文件。

然后堆冲突文件进行修改，来确立最终的版本。最后进行add,commit来解决冲突文件。

## 删除分支

|  |
| --- |
| 图 31 |

如图31所示。通过branch –d来删除目标分支。

## 分支管理策略

|  |
| --- |
| 图 32 |

如图32所示。通过普通的merage进行分支合并后，是看不到分支信息了。这种合并的方式是Fast-Forward。如果想要在合并后保留分支信息。可以取消Fast-Forward模式。通过—no-ff来禁用Fast-Forward。如图33所示。

|  |
| --- |
| 图 33 |

在平时开发的过程中。Masteer分支应该是非常稳定的。

通常开发工作都是在自己的分支上进行。时不时的将自己的分支合并到dev分支上即可。等到需要发版的是再将dev分支合并到master上。

建议没做一个新的功能时就创建要给单独的func分支，功能完毕后再与dev进行合并。

# 缓存工作区

|  |
| --- |
| 图 34 |

缓存工作区主要是通过stash指令来将工作区中的所有变化文件状态进行缓存到stash队列中。

当需要恢复时可以通过stash pop来恢复队尾所缓存的工作区。如图34所示。这种方式会将队尾中的缓存记录删除。

通过stash list来查看缓存队列中所缓存的所有记录。通过stash apply stash@{index}来根据需要蓝恢复指定的缓存工作区。如图35所示。这种方式不会删除队列中的缓存记录，需要手动通过stash drop来进行删除。

|  |
| --- |
| 图 35 |

# 异常问题

|  |
| --- |
|  |

windows中的换行符为 CRLF， 而在[**Linux**](http://lib.csdn.net/base/linux)下的换行符为LF，所以在执行add . 时出现该提示。需要首先删除原有的.git目录。然后通过设置禁用自动转换功能。

git config --global core.autocrlf false