**Git命令总结和常见问题解决方案**

##说明  
本篇文章是看[Pro Git](https://git-scm.com/book/zh/v2/)一书按照自己的思维模式对Git命令进行了分类。  
##一、Git配置  
**配置Git用户名**  
git config --global user.name ""  
**配置Git邮箱**  
git config --global user.email ""  
**设置文本编译器**  
git config --global core.editor  
**差异分析工具**  
git config --global merge.tool vimdiff  
**设置git log的提交时间格式**  
git config --global log.date iso  
**查看配置信息**  
git config --list  
git config --list --show-origin  
**查询某项配置设置的值**  
git config user.email  
git config --list --show-origin|grep "user.email"

注意事项：  
参数–system、–global和–local依次为针对所有用户的配置、针对当前用户的配置、针对当前git仓库的配置，  
如果存在同样的配置优先级从高低为：–local、–global、–system。

##二、创建Git仓库的两种方式  
**初始化当前目录为Git仓库**  
git init  
**克隆远程仓库到本地**  
git clone git@server-name:path/repo-name.git  
**不使用默认的origin仓库名称来跟着远程仓库，而是使用，其实就是给远程开发起个别名。**  
git clone git@server-name:path/repo-name.git -o <name>  
**获取远程服务器上的裸仓库，即复制远程仓库的.git目录以及其中的文件到本地**  
git clone git@server-name:path/repo-name.git --bare  
##三、本地仓库与远程仓库  
**将远程仓库的引用添加到本地仓库中，origin为远程仓库别名**  
git remote add origin git@server-name:path/repo-name.git  
**查看本地仓库添加的远程库引用名称**  
git remote  
**查看本地仓库添加的远程库引用名称和对应远程仓库URL**  
git remote -v  
**查看远程分支详细信息(远程分支列表以及状态、本地分支pull和push对应的远程分支等)**  
git remote show [远程仓库引用名称]  
**删除本地仓库中的远程仓库引用名称**  
git remote rm [远程仓库引用名称]  
**重命名远程仓库引用名称。注意：之前origin/master访问远程分支，修改后origin-test/master访问远程分支。**  
git remote rename [原有远程仓库引用名称] [新的远程仓库引用名称]  
##四、提交  
**添加文件到暂存区**  
git add <file-name>  
**将暂存区中指定文件从暂存区提交到本地Git仓库**  
git commit -m "commit desc" <file-name>  
**将暂存区中所有文件从暂存区提交到本地Git仓库**  
git commit  
\**跳过暂存区直接提交到本地仓库，相当于git add ;git commit.。注意此命令不会将未跟踪(Untracked files)的文件提交到本地仓库中。*  
git commit -a -m "commit desc"等同于git commit -am "commit desc"  
**查看本地Git仓库状态**  
git status  
##五、对比  
**工作区和暂存区的文件不同对比**  
git diff   
**暂存区和本地Git仓库文件最后一次提交不同对比**  
git diff --cached <file-name> 等同于 git diff --staged <file-name>  
**工作区和本地Git仓库文件最后一次提交不同对比**  
git diff HEAD <file-name>  
**比较当branch-a和branch-b两个分支最新一次提交内容的不同，其实比较的是两个提交的差异，应为branch-a和branch-b都是指向最新的一次提交，也可以根据需要指定SHA-1来比较两次提交差异**  
git diff <branch-a> <branch-b>  
**查可能存在的空白字符问题**  
git diff --check <file-name>  
**过滤掉空白字符的差异**  
git diff -b <file-name>  
**查看当前Git可使用以及支持哪些diff插件**  
git difftool --tool-help  
**使用指定的diff插件比较文件差异**  
git difftool --tool=vimdiff3  
##六、撤销&回退  
**回退到上一次提交**  
git reset --soft HEAD~  
解析：其实就是移动了HEAD的指向的commit对象  
**将已暂存的文件移除暂存区**  
git reset HEAD  
解析：其实就是将HEAD指向最新一次更新，然后将仓库最新一次提交更新到索引区，加上路径会将提交快照中指定的文件移除暂存区。  
**将工作目录所有文件回退到仓库最新一次提交**  
git reset --hard HEAD  
解析：其实就是将HEAD指向最新一次更新，然后将仓库最新一次提交更新到索引区，将索引区内容更新到工作目录)

注意:此命令没有加上路径的形式(git reset –hard HEAD fileA.txt)。

**将工作目录指定文件回退到仓库最新一次提交快照中的文件**  
git checkout HEAD fiel.txt  
**切换分支**  
git checkout beanch-name  
**修改最近一次提交的文件和提说明。此命令执行后暂存区的文件将会被提交，并可以重新填写说明，生成一个新的commit来覆盖最近一次commit**  
git commit --amend  
##七、提交记录查看  
**列出HEAD所指向的位置发生的变化(git commit、git checkout dev等命令会改变HEAD所指向的分支已经分支所指向的commit对象，reflog会列出这种变化)**  
**其实就是你向git本地仓库保存数据时产生一个SHA-1时，Git就会记录这个操作，并可以使用git reflog 命令显示这些操作历史。**  
git reflog  
**查看第2条命令之后的命令记录**  
git reflog HEAD@{2}  
**查看dev分支的历史**  
git reflog dev  
**查看dev分支从第2条命令之后的命令记录**  
git reflog dev@{2}  
**当前仓库所有分支的提交历史**  
git log --all  
**以reflog的方式查看提交历史(提交历史中多了Reflog信息)**  
git log -g  
**查看当前分支提交历史以一个简短的SHA-1值和提交说明**  
git log --oneline  
**查看合并的提交记录(在提交历史中查看以合并的提交记录，也就是有两个父commit对象的提交)**  
git log --merges  
**查看未合并的提交历史记录(在提交历史中查看未合并的提交记录，也就是除了有两个父commit对象的提交之外的提交)**  
git log --no-merges

注意：–merges和–no-merges并不能查看某个分支是否合并到了当前分支，而是对已经合并的历史记录做过滤而已。

**查看存在于iss11分支中的提交不在master提交(即有iss33分支有哪些提交为合并到master中)**  
git log masster..iss11  
**等同于git log masster…iss11**  
git log ^master..iss11  
**存在iss11分支中的提交不存在master分支中的提交或存在于dev中的提交不存在master中的提交**  
git log iss11 dev ^master \*\*存在各自分支中的并不同时包含的提交(两个分支的并集)\*\* git log iss11...master  
**查询当前目录24到25号两天的提交记录，并且以oneline形式显示提交记录、未合并的(–no-merges)、范围在当前目录下(./)的文件提交记录**  
git log --after="2018-07-24" --before="2018-07-25" --pretty=oneline --no-merges -- ./  
**命令操作历史记录**  
gitk  
**按照提交时间显示所有提交记录**  
git log  
git log选项说明：

------显示形式相关参数：

-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）。

format选项说明：

%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

%ae 作者的电子邮件地址

%ad 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

%ce 提交者的电子邮件地址

%cd 提交日期

%cr 提交日期，按多久以前的方式显示

%s 提交说明

--date:指定提交历史中的提交时间显示格式，--date=iso、--date=local

------过滤条件相关

-(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。

--grep:仅显示含指定关键字的提交

-S:仅显示添加或移除了某个关键字的提交

##八、从本地Git移除文件(不在进行Git版本管理)  
**从磁盘和版本库中删除文件。最后执行git commit - m "commit desc"命令级将rm操作提交到版本库**  
git rm <file-name>  
**仅从版本库中删除文件，并不删除磁盘上的文件。最后执行git commit - m "commit desc"命令级将rm操作提交到版本库**  
git rm --cached <file-name>  
**文件重命名**  
git mv <file-name>

##九、本地仓库推送到远程仓库  
**推送当前分支到分支关联的远程仓库分支**  
git push  
**推送本地仓库分支到远程仓库分支，如果远程仓库没有此分支会新建分支**  
git push [远程仓库名称] [本地分支名称]:[远程分支名称]  
**将本地分支推送到远程仓库分支。如果远程仓库没有此分支会新建分支，并建立本地分支和远程仓库分支的关联关系**  
git push --set-upstream [远程仓库名称] [本地分支名称]:[远程分支名称]  
等同  
git push -u [远程仓库名称] [本地分支名称]:[远程分支名称]

**删除远程仓库中的分支和本地对此远程分支的引用,并不删除本地的分支。从命令格式可以看出推送一个空的仓库到远程分支就达到了删除的目的。**  
git push [远程仓库名称] :[远程分支名称]  
##十、远程仓库拉取到本地仓库  
**拉取当前分支关联的远程分支内容到本地**  
git pull  
**拉取远程仓库所有分支以及分支内容到本地仓库中的分支(做合并)**  
git pull [远程分支名称]  
**拉取特定分支**  
git pull [远程分支名称] [本地分支]:[远程分支名称]

注意:git pull相当于git fetch+git merge。拉取过程中提示失败，可能是本地Git仓库内容和远程仓库内容不一致，会进入mergeing模式，使用git status命令会提示出哪些文件需要merge，人工进行merge后git commit即可。

**抓取当前分支关联的远程分支内容到本地仓库中的远程分支引用分支**  
git fetch  
**从远程仓库抓取所有分支以及内容到本地仓库中的远程分支引用分支**  
git fetch [远程仓库名称]  
**抓取特定分支**  
git fetch [远程仓库名称] [远程分支名]  
**如果远程仓库分支在本地没有远程分支引用会在本地新建这个远程分支引用。例如：拉取远程仓库fetch-test分支到本地仓库，本地仓库不存在gitee-gitTest01/fetch-test则会新建。**

Administrator@USER-20171018VX MINGW64 /d/gitTest/gitTest01 (iss11)

$ git fetch gitee-gitTest01 fetch-test

From https:gitee.com/superman-zhangxy/gitTest01

\* branch fetch-test -> FETCH\_HEAD

\* [new branch] fetch-test -> gitee-gitTest01/fetch-test

##十一、分支  
git branch <branch-name>：创建分支  
git checkout <branch-name>：切换分支  
git checkout -b <branch-name>：创建分支并切换到创建的分支  
git branch：查看所有分支

git branch -v：查看每一个分支最后一次提交  
git branch -vv：查看本地仓库分支和远程仓库分支对应关系以及每一个分支最后一次提交  
git branch -r：查看远程仓库引用  
git branch -a：查看本地分支和远程仓库分支引用(即列出所有分支)  
git branch -vva：git branch -vv + git branch -a。  
git branch --merge：查看哪些分支已经合并到当前分支。如果分支已经合并到当前分支删除以合并的分支，删除后也不会丢失文件版本，因为已经合并到当前分支。  
git branch --no-merges：查看哪些分支没有合并到当前分支

git branch -d branch name：删除分支(如果分支有文件修改而没有merge会不让删除)  
git branch -D branch name：强制删除分支

git checkout -b dev origin/dev：在本地创建和远程仓库中对应的分支。

**在本地仓库分支和远程仓库分支建立关联关系。也就是说git pull，git fetch，git push如果不指定远程仓库分支会作用于默认的本地仓库分支对应的远程仓库分支。**  
git branch --set-upstream-to [远程仓库名称]/[分支名称] [本地分支名称]

##十二、合并  
git merge <branch-name>：将指定分支合并到当前分支，Fast-forward为“快进模式”，直接将master指向指定分支(即)。此情况可以顺利执行，不会提示有冲突：你在创建完分支并进行文件修改期间没有其他人向master提交，也就说在你merge时你是在最新版本上进行修改的，否则会提示文件有冲突，冲突的情况下需要手工修改文件解决冲突在进行merge。解决冲突后会将合并好的文件提交到版本库。  
git merge --no-ff -m "<desc>" dev：不使用Fast-forward模式。  
git merge --allow-unrelated-histories [分支名称]：强制合并两个不相关的分支。例如：我在本地创建了项目A，并添加了若干代码，有在远程仓库创建了A仓库，现在想将远程A仓库的master分支合并到本地的master分支会报错：fatal: refusing to merge unrelated histories(致命的：拒绝合并不相关的历史),使用此命令可解决报错。

##十三、储藏  
git stash：将当前工作区(已跟踪文件)和暂存区储藏  
git stash save "test stash"：作用同 git stash，不过可以额外增加一些说明文字  
git stash --keep-index：将当前工作区(已跟踪文件)储藏，不储藏暂存区中的文件  
git stash -u 或 git stash --include-untracked：将当前工作区(已跟踪文件和未跟踪)和暂存区储藏  
git stash --all:工作目录(未跟踪、已跟踪)、忽略文件和暂存区都会被储藏  
git stash list：查储藏工作区列表  
git stash pop：回复并删除储藏区  
git stash apply：回复储藏区。添加 --index 参数会将暂存区的文件恢复到暂存区，否则所有文件将恢复到工作区，这不是我们想看到的。  
git stash drop：删除储藏区  
git stash show -p stash@{0}：储藏中的文件与它上一个版本之间的差异比较(diff)，并不是与当前工作目录或git仓库最后一次提交进行差异比较，而是储藏文件时这些文件与他上一个版本之间的差异  
git stash show -p stash@{0} | git apply -R：取消恢复的储藏。stash@{0}为储藏名称，不加名词默认取消最近回复的储藏。  
git stash branch <branch-name>：根据储藏创建一个新分支

git stash应用场景：当前分支功能开发到一半时，又想去其它分支进行其它功能开发，又不想将当前分支文件提交到版本库，使用 git stash储藏当前分支文件，然后切换到其它分支工作。

注意：在恢复储藏时分支的工作目录不一定是需要干净的，如果储藏中的文件和工作目录中有冲突，git会提示我们合并。我们也可将储藏应用在其它分支上。也就是说应用储藏在一个干净的工作目录和应用在创建储藏所在的分支并成功应用储藏的先决条件。

##十四、标签  
**给最近的commit打一个标签(轻量标签,lightweight)**  
git tag <tag-name>  
**给最近的commit打一个标签，并给标签添加描述(附注标签,annotated)**  
git -a tag-name -m "tag desc"  
**给某次commit打一个标签**  
git tag <tag-name> commit的SHA-1  
**查看所有标签**  
git tag  
**查看符合条件的标签名称。查看标签名称以1.1开头的标签**  
git tag -l "1.1\*"  
**删除标签**  
git tag -d <tag-name>  
**推送指定标签到远程仓库**  
git push [远程仓库引用名称] [标签名称]  
**推送所有标签到远程仓库**  
git push origin --tags

一个轻量标签很像一个不会改变的分支 - 它只是一个特定提交的引用。  
然而，附注标签是存储在 Git 数据库中的一个完整对象。 它们是可以被校验的；其中包含打标签者的名字、电子邮件地址、日期时间；还有一个标签信息；并且可以使用 GNU Privacy Guard （GPG）签名与验证。 通常建议创建附注标签，这样你可以拥有以上所有信息；但是如果你只是想用一个临时的标签，或者因为某些原因不想要保存那些信息，轻量标签也是可用的。

##十五、子模块

项目中的子目录不受项目git版本管理，有独立的git版本管理。进入子模块执行git add或git commit等命令是作用在子模块的git版本管理，项目中(上层项目或主项目)文件不受影响，反之也是。

**1,克隆一个子模块**  
git submodule add [远程仓库地址] 远程仓库别名  
**2,克隆一个带子模块的项目**  
2.1 进入子模块目录执行如下命令初始化子模块版本控制相关文件  
git submodule init  
2.2 进入子模块目录执行如下命令更新子模块中的文件  
git submodule update  
##十六、其它  
**交互式暂存命令**  
git add -i  
**获取Git命令格式帮助**  
git help <verb>  
git <verb> --help  
**列出Git在各种场景下常用的命令**  
git help  
##十五、常见问题  
**如何从每个提交快照中擦除指定文件**  
我们可能又有存心大意将带有数据库密码的配置文件提交到了远程仓库，可以使用git filter-branch命令来从每个提交快照中擦除配置文件。如下事例：执行git filter-branch --tree-filter 'rm -f src/main/resources/database.properties' HEAD从每个快照中擦除database.properties文件。

Administrator@USER-20171018VX MINGW64 /d/gitee/baseMyBatis (temp)

$ pwd

/d/gitee/baseMyBatis

Administrator@USER-20171018VX MINGW64 /d/gitee/baseMyBatis (temp)

$ git filter-branch --tree-filter 'rm -f src/main/resources/database.properties' HEAD

Rewrite b1f7663b07a824436689522250472b4c2f8db4e4 (1/5) (0 seconds passed, remaining 0 predicted)

Rewrite 082a9c28d23f7953040a2e7d0a61d6c20170f8c8 (2/5) (2 seconds passed, remaining 3 predicted)

Rewrite e364f012ec4b99cb7029683b7b6514031ee3cba7 (2/5) (2 seconds passed, remaining 3 predicted)

Rewrite 5ca1207a2420cc9e138c6d82bd5bed30e211136e (4/5) (5 seconds passed, remaining 1 predicted)

Rewrite dd39df7876853977f9cb7396d89ec0001ec95d52 (5/5) (7 seconds passed, remaining 0 predicted)

Ref 'refs/heads/temp' was rewritten

Administrator@USER-20171018VX MINGW64 /d/gitee/baseMyBatis (temp)

$ ls src/main/resources/

mybatis3/ mybatis-config.xml

Administrator@USER-20171018VX MINGW64 /d/gitee/baseMyBatis (temp)

$ git log --oneline

e3d1b11 (HEAD -> temp) 对项目添加说明信息

c12432e 不对IntelliJ IDEA IDE产生的文件进行Git关联

56c17f1 Merge remote-tracking branch 'baseMyBatis/master' into temp

5d9f28e 一个使用MyBatis框架访问数据库的java程序

082a9c2 Initial commit

注意：1，不要在master分支做文件的擦除操作，要checkout一个新的分支，在新分支上做。2，在新分支擦除指定的文件后可以强行覆盖master分支(即git reset --hard temp),如果使用git merge temp到master分支，会出现如图的提交记录，这并不是我们想要的。  
