Git常用命令总结

gybguohao 关注

0.97 2018.11.17 18:35:11 字数 3,034 阅读 7,879

`git config`

`git config`命令的作用是配置git的相关信息。

1. 配置全局的用户名和邮箱，mac下可通过终端输入命令`cat ~/.gitconfig`查看配置信息。
   - 设置用户名：`git config --global user.name "name"`
   - 设置用户邮箱：`git config --global user.email "eamil"`
2. 针对单个仓库配置用户名和邮箱，mac下可通过终端进入到项目的根目录中，然后输入命令`cat .git/config`查看配置信息。
   - 设置用户名：`git config user.name "name"`
   - 设置用户邮箱：`git config user.email "eamil"`
3. 查看git所有配置信息：`git config --list`
4. 查看配置的用户名：`git config user.name`
5. 查看配置的用户邮箱：`git config user.email`
6. 定义命令别名：
   - `git config --global alias.st status`：则`git status`可用`git st`代替
   - `git config --global alias.co checkout`：则`git checkout`可用`git co`代替
   - `git config --global alias.ci commit`：则`git commit`可用`git ci`代替
   - `git config --global alias.br branch`：则`git branch`可用`git br`代替

也可以自定义其他命令的别名，主要合理且方便操作即可。

`git init`

`git init`命令的作用是在当前目录中初始化仓库，并且创建一个名为`.git`的子目录，该目录含有你初始化的`Git`仓库中所有的必须文件。

`git status`

`git status`命令的作用是显示文件状态，红色表示工作目录的文件被修改但还没有提交到暂存区，绿色表示已经提交到暂存区。

1. 以极简的方式显示文件状态：`git status -s`

- A：本地新增的文件（服务器上没有）
- C：文件的一个新拷贝
- D：本地删除的文件（服务器上还在）
- M：红色为修改过未被添加进暂存区的，绿色为已经添加进暂存区的
- R：文件名被修改
- T：文件的类型被修改
- U：文件没有被合并(你需要完成合并才能进行提交)
- X：未知状态(很可能是遇到git的bug了，你可以向git提交bug report)
- ?：未被git进行管理，可以使用`git add fileName`把文件添加进来进行管理

已经被修改但还没提交到暂存区的文件，可以通过命令`git checkout -- fileName`撤销更改。

`git add`

`git add`命令的作用是将文件从工作目录添加至暂存区

1. 把所有修改的信息添加到暂存区：`git add .`
2. 把所有跟踪文件中被修改过或已删除的文件信息添加至暂存区：`git add -u`或`git add --update`，它不会处理那些没有被跟踪的文件
3. 把所有跟踪文件中被修改过或已删除文件和所有未跟踪的文件信息添加到暂存区：`git add -A`或`git add --all`

注意：`git add .`和`git add -A`在2.x版本中提交类型方面功能相同，但会因为所在目录不同产生差异：

- `git add .`只会提交当前目录或者子目录下相应文件。
- `git add -A`无论在哪个目录执行都会提交相应文件。

已经被提交到暂存区的文件，可以通过命令`git reset HEAD -- fileName`撤销提交。

`git commit`

`git commit`命令的作用是将暂存区的修改提交到本地仓库，同时会生成一个commmit-id。

1. 将暂存区的修改提交到本地仓库：`git commit -m "message"`，"message"是本次提交的简述内容，比如添加新功能或修复bug等
2. 将本地工作区中修改后还未使用`git add .`命令添加到暂存区中的文件也提交到本地仓库：`git commit -a -m "message"`，该命令相当于以下两条命令：

   - `git add .`：把所有修改的信息添加到暂存区
   - `git add -m "message"`：将暂存区的修改提交到本地仓库

3. 修改最后一次提交（可用于漏掉某个文件的提交或重新编辑信息）：`git commit --amend`

`git pull`

`git pull`命令的作用是获取远程主机某个分支的更新，再与本地指定分支合并。`git pull <远程主机名><远程分支名>:<本地分支名>`

1. 取回远程主机上的dev分支与本地的master分支合并：`git pull origin dev:master`
2. 取回远程主机上的dev分支与当前分支合并：`git pull origin dev`，该命令相当于以下两条命令：

   - `git fetch origin`：获取远程主机上所有分支的更新，也可以用`git fetch origin dev`表示获取远程主机上dev分支的更新
   - `git merge origin/dev`：当前分支合并dev分支

注意：通过`git fetch`所取回的更新，在本地主机上需要用"远程主机名/分支名"的形式读取，比如origin主机的master分支，就需要用`origin/master`来读取。

`git fetch`

`git fetch`命令的作用是将远程主机上所有分支的更新取回本地，并记录在`.git/FETCH_HEAD`中

1. 获取远程主机上master分支的代码：`git fetch origin`
2. 在本地新建test分支，并将远程主机上master分支代码下载到本地test分支：`git fetch origin master:test`

`git push`

`git push`命令的作用是将本地分支的更新推送到远程主机上。

1. 将本地`master`分支的更新推送到远程主机上：`git push origin master`

2. 删除远程dev分支：`git push origin --delete dev`

`git branch`

`git branch`命令的作用主要是做分支管理操作。

1. 查看本地分支：`git branch`
2. 查看本地和远程分支：`git branch -a`
3. 新建名字为test的分支：`git branch test`
4. 将test分支名字改为dev：`git branch -m test dev`
5. 删除名字为dev的分支：`git branch -d dev`
6. 强制删除名字为dev的分支：`git branch -D dev`

以上命令都是针对本地仓库操作，不影响远程仓库。

`git checkout`

`git checkout`命令最常用的情形是创建和切换分支以及撤销工作区的修改。

1. 切换到tag为v1.0.0时对应的代码：`git checkout v1.0.0`
2. 在tag为v1.0.0的基础上创建分支名为test的分支：`git checkout -b test v1.0.0`。该命令相当于以下两条命令：
   - `git branch test v1.0.0`：在v1.0.0的基础上创建分支test
   - `git checkout v1.0.0`：切换到分支test
3. 把当前目录所有修改的文件从HEAD中移除并且把它恢复成未修改时的样子：`git checkout .`
4. 撤销工作目录中文件的修改（文件有改动但还未`git add`）：`git checkout -- fileName`，或者撤销所有修改使用`git checkout .`

`git tag`

`git tag`命令主要是对项目标签进行管理。

1. 查看已有的标签历史记录：`git tag`
2. 给当前最新的commit打上标签：`git tag <标签的定义>`
3. 给对应的commit id打上标签：`git tag <标签定义> <commit id>`

`git log`

`git log`命令的作用是查看历史提交记录

1. 查看历史提交记录：`git log`
2. 将每条历史提交记录展示成一行：`git log --oneline`
3. 查看某个人的提交记录：`git log --author="name"`
4. 显示ASCII图形表示的分支合并历史：`git log --graph`
5. 显示前n条记录：`git log -n`
6. 显示某个日期之后的记录：`git log --after="2018-10-1"`，包含2018年10月1号的记录
7. 显示某个日期之前的记录：`git log --before="2018-10-1`，包含2018年10月1号的记录
8. 显示某两个日期之间的记录：`git log --after="2018-10-1" --before="2018-10-7"`

`git reset`

`git reset`命令的作用是撤销暂存区的修改或本地仓库的提交。

1. 撤销已经提交到暂存区的文件（已经`git add`但还未`git commit`）：
   - 撤销已经提交到暂存区的文件：`git reset HEAD fileName`或`git reset --mixed HEAD fileName`
   - 撤销所有提交：`git reset HEAD .`或`git reset --mixed HEAD .`
2. 对已经提交到本地仓库做撤销（已经`git commit`但还未`git push`）：
   - 将头指针恢复，已经提交到暂存区以及工作区的内容都不变：`git reset --soft commit-id`或`git reset --soft HEAD~1`
   - 将头指针恢复并且撤销暂存区的提交，但是工作区的内容不变：`git reset --mixed commit-id`或`git reset --mixed HEAD~1`
   - 将所有内容恢复到指定版本：`git reset --hard commit-id`或`git reset --hard HEAD~1`

注意：`commit-id`可通过`git log`查看（取前六位即可），`HEAD~1`表示前一次提交（可以此类推）。

`git remote`

`git remote`命令的作用主要是管理远程仓库。

1. 查看关联的远程仓库的名称：`git remote`
2. 查看关联的远程仓库的详细信息：`git remote -v`
3. 添加远程仓库的关联：`git remote add origin <远程仓库地址>`

4. 删除远程仓库的关联：`git remote remove <远程仓库名称>`
5. 修改远程仓库的关联：`git remote set-url origin <新的远程仓库地址>`
6. 更新远程仓库的分支：`git remote update origin --prune`

`git merge`

`git merge`命令的作用主要是分支的合并。

1：如果当前是master分支，需要合并dev分支：`git merge dev`

`git stash`

`git stash`命令的作用主要如果当前分支所做的修改你还不想提交，但又需要切换到其他分支去查看，就可以使用`git stash`保存当前的修改。

1. 保存当前进度：`git stash`
2. 查看已经保存的历史记录：`git stash list`
3. 重新应用某个已经保存的进度，并且删除进度记录：`git stash pop <历史进度id>`，
4. 重新应用某个已经保存的进度，但不删除进度记录：`git stash apply <历史进度id>`，如果直接使用`git stash`默认是使用最近的保存
5. 删除某个历史进度：`git stash drop <历史进度id>`
6. 删除所有的历史进度：`git stash clear`

`gitignore`

`.gitignore`文件的作用是忽略那些没必要的提交，比如系统环境或程序运行时产生的文件。GitHub为我们提供了各个语言的gitignore合集github/gitignore，其中也包括Android.gitignore。

将本地新建的项目提交到远程仓库的步骤

- 初始化本地仓库`git init`
- 将本地内容添加至git本地暂存区中`git add .`
- 将暂存区添加至本地仓库中`git commit -m "first commit"`
- 添加远程仓库路径`git remote add origin https://github.com/gybguohao/test.git`
- 将本地内容push至远程仓库中`git push -u origin master`

秘钥配置

- mac终端查看是否已经存在SSH密钥：`cd ~/.ssh`，如果没有密钥则不会有此文件夹。

- 生成新的秘钥, 命令如下

  `ssh-keygen -t rsa -C "eamil"`

  你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，因为这个个Key仅仅用于简单的服务，所以也无需设置密码。

  完成后会有如下显示

  Enter passphrase (empty for no passphrase):
  Enter same passphrase again:
  Your identification has been saved in /Users/gybguohao/.ssh/id_rsa。
  Your public key has been saved in /Users/gybguohao/.ssh/id_rsa.pub。
  The key fingerprint is:
  SHA256:5V6ZCQNS/3bVdl0GjGgQpWMFLazxTslnKbW2B1mbC+E eamil

  如果服务器端需要公钥, 直接复制.ssh目录下的id_rsa.pub内容即可。