git基本操作---跟踪文件

> 成文时间：2020-09-22
>
> 联系作者：qiao_jinming@foxmail.com
>
> 文章作者：乔金明

1. 查看当前仓库状态

- 使用命令即可查看当前的仓库状态

```
# status状态命令也有相依的显示形式，默认是长格式输出
git status [--long]

# 如果只需要看概要，那么可以以短格式输出
git status [-s | --short]

# 如果想看到文件修改的更多细节，可以添加以下命令，更多细节的查看可以通过diff，这里只是尽可能的多看一些仓库的修改细节
git status [-v | --verbose]
```

- status使用短格式时输出的相关字段的意义

```
# git使用-s后输出的格式如下，即修改的文件，或者由哪个文件修改来的，XY可以理解为状态码，其意义包括
XY path
XY orig_path -> path

# 使用以下命令进行指定文件的忽略跟踪，ignored后可以加文件或者文件夹名，也可以加模式（正则），来统一忽略跟踪某个匹配模式的数据
git status --ignored[=<mode>]
```

  1. ??：表示还没有被跟踪，即此文件还没有被git操作过
  2. ''：表示没有被修改
  3. M：表示此路径数据已经被修改
  4. A：表示此路径数据已经被添加到缓存区
  5. D：表示此路径数据已经被删除
  6. R：表示此路径数据已由源路径数据重命名
  7. C：表示此路径数据已由源路径数据复制
  8. U：表示此路径数据已经更新，但是还没有合并
  9. !!：表示已经忽略跟踪数据，一般是在使用了--ignored后

- 关于状态码XY的详细解释
  - X表示索引中的状态，即当前在缓存区中的状态
  - Y表示当前工作树的状态，即当前对此文件的修改状态
- 干净的工作区示例，由图可见当前已经没有跟踪的文件，所有的文件均已经被提交，当前的分支名称为master，仓库名称为origin，这些都是默认的名称，PS：github网站默认分支名将会修改为main，目的是为了避嫌人种歧视

```
λ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

2. 将文件添加到暂存区

- 通过git控制的工作区文件分为两种状态，分别为已跟踪和未跟踪，已跟踪表示的是已经纳入版本控制的文件，即已经生成了快照的文件，新克隆的文件目录属于已跟踪状态，因为克隆时是将快照同时克隆下来的，已经存在版本控制中

```
# 跟踪一个文件，即将一个文件添加到暂存区中，add命令也支持模式匹配，即可以通过模式来批量添加文件
git add xxx

# 将某个目录下的全部文件递归的添加到暂存区中
git add xxx/

# 显示添加文件到工作区的详细信息
git add [-v | --verbose] xxx

# 提交全部文件，注意根目录可以直接使用“.”，但是git也提供了-A操作
# add . 与add -A在git 1.x版本中前者是不会提交已经删除的文件，但是在2.x版本，两者的提交类型已经相同了
# 注意"."代表的是目录不是命令，所有在不同目录下，使用“.”仅添加当前目录与子目录的数据，而-A添加项目的全部修改
# -u表示更新，即只添加已经被跟踪的文件
# *与.的区别实际上是命令行中对这两个符号的区别，在Bash中，*是不包括以.开头的隐藏目录的
git add .
git add *
git add [-u | --update]
git add [-A | --all]

# 如果需要更细粒度的添加，因为有些调试信息是不需要添加的，配置相应的命令即可完成文件内部某块的添加
git add [-p | --patch]
# 配合的其他命令操作
y - 暂存此区块
n - 不暂存此区块
q - 退出，不暂存包括此块在内的剩余的区块
a - 暂存此块与此文件后面所有的区块
d - 不暂存此块与此文件后面所有的 区块
g - 选择并跳转至一个区块
/ - 搜索与给定正则表达示匹配的区块
j - 暂不决定，转至下一个未决定的区块
J - 暂不决定，转至一个区块
k - 暂不决定，转至上一个未决定的区块
K - 暂不决定，转至上一个区块
s - 将当前的区块分割成多个较小的区块
e - 手动编辑当前的区块
? - 输出帮助

# 如果设置了不需要添加的文件，可以通过以下命令强制添加
git add [-f | --force] xxx
```

- 如果某个文件已经添加到了暂存区，那么这个文件就可以正式提交到版本库中，倘若在未提交之前进行修改，那么这个文件就会有两种状态，一种是工作区的修改状态，一种是暂存区的历史状态，此时提交时会提交暂存区中那个状态的文件，即没有经过暂存区的文件状态不会被提交到版本库中

- 如果需要把二次修改的文件提交到版本库，只需要再次执行add命令即可，此时git会报一个警告，即文件状态已经更新到了最新跟踪的状态

2. 如何设置忽略文件

- git添加跟踪文件时不是所有文件都想提交的，例如生成的log文件，独立配置文件，临时文件或者动态文件等，如果每次通过命令进行忽略将会很不方便，git提供了一种方式，即创建一个.gitignore文件，里面可以对需要过滤的文件模式进行声明
- git忽略文件有几种方式，优先级由高到低为：
    1. 直接从命令行进行指定忽略模式的文件
    2. 项目中.gitignore文件指定的相关文件模式
    3. .git/info/exclude文件中指定的需要忽略的文件
    4. 从配置文件中的core.excludesFile指定的文件中获得需要进行忽略的文件
- 忽略文件的这几种方式的区别：
    1. .gitignore文件是所有开发者所共享的，作为项目的一部分进行提交，所有人都要遵循此文件进行相应文件的忽略
    2. 而.git/info/exclude中是特定的库用的忽略方式，是不需要与其他相关库进行共享的，例如有些用户的特定辅助文件
    3. 通过配置文件来指定的用户全局忽略文件，配置是在本地进行的，所以通过配置这个文件，可以对本地需要进行忽略的文件进行全局忽略，是用户希望在任何情况都进行的忽略
- .gitignore匹配的模式
    1. 空行不匹配任何模式，以#号开头的作为注释，“\”表示转义
    2. 可以使用简化的正则表达式，即glob模式进行匹配
    3. 以“A/B”表示目录，即匹配A目录下的B项目，如果以“/”结尾代表整个目录均不提交，如果模式中不存在“/”则代表直接匹配该文件
    4. “*”表示通配符，一个“*”表示匹配任意多字符，包括空字符，两个“*”表示匹配任意中间目录
    5. “?”表示通配符，匹配任意一个字符
    6. “[]”表示匹配括号内的范围的任意一个字符，例如[ab]表示匹配一个a或者一个b，[0-9]或者[a-z]表示匹配任意一个数字或者任意一个小写字母
    7. “!”表示不忽略的文件或者目录，如果已经被上面忽略掉的父文件夹的文件不可以通过这个模式进行不忽略，简而言之就是父文件夹已经被忽略了，那么这里面的子文件就没有存在的意义了，就是要求子文件不被忽略也不行
    8. 忽略文件的规则是由上至下进行的，如果前面的范围更大，那么后面的小范围就不会生效
    9. 如果在.gitignore文件创建前就已经push过文件，那么远程就存在了该文件的版本控制，所以过滤规则看起来是不起作用的，可以通过版本控制从远程仓库删除这些文件进行pull或者本地删除进行push，重新使已经提交的过滤规则文件不纳入版本控制

```
# 可以通过这条命令来查看是哪个规则导致了xxx被过滤掉，如果没有被过滤则不输出
git check-ignore -v|--verbose xxx
```

- 子目录下的.gitignore文件仅作用于所在目录，所以一个项目可能会存在多个.gitgnore文件

3. 查看文件的修改细节

- 通过status命令可以初步查看文件的状态，可以通过diff进行文件本身修改部分的查看，同样三种状态对应也有相关的查看命令
    1. 比较工作目录与暂存区

```
# 比较缓存区与当前工作目录的不同，如果不声明需要查看的文件，则显示全部文件的不同
git diff [--] [<filename>...]

# 显示简要信息，即直接显示修改的简略信息，而不是显示整个文件修改的具体内容
```

```
git diff --stat [--] [<filename>...]

# 比较当前分支下的文件与指定<branch>分支下的不同
git diff <branch> -- <filename>

# 可以比较不是git空间的数据，即直接比较两个文件有何不同
git diff --no-index [--] <path> <path>

# 可以显示二进制文件的不同
git diff --binary [--] [<filename>...]
```

2. 比较暂存区与提交的git仓库

```
# 比较暂存区与最后一次提交的指定文件的修改后的不同，其中commit为指定的版本哈希值，默认为与最后一次提交相比较
git diff <--cached | --staged> [<commit>] [--] [<path>...]

# 如果增加了commit的哈希值，则默认比较缓存区与最后一次提交，不用再加--cached
git diff <commit> [--] [<path>...]

# 比较n次提交到git仓库之间的不同
git diff <commit> <commit> [--] [<path>...]
```

3. 比较工作区与提交的git仓库

```
# 比较当前工作区与最后一次提交到git仓库的内容的比较
git diff HEAD

# 比较当前工作区与上n次的提交到git仓库的内容比较，n代表上几次，等价于^的个数
git diff <HEAD~n | HEAD^^^...>
```

4. 对比输出结果的解释
   - 输出的结果分为四部分，第一部分表示git哈希值的之间的比较与对象模式

     ```
     # 表示两个版本git的哈希值由bc73e0d对象对比09958af对象，最后表示对象模式，10064即为普通模式下的644权限
     index bc73e0d..09958af 100644
     ```

   - 第二部分是文件的基本信息

     ```
     # 例子中‘---’表示变动前的文件，‘+++’表示变动后的文件
     --- a/test.txt
     +++ b/test.txt
     ```

   - 第三部分是概述变动前文件与变动后文件的变动情况

     ```
     # 以‘@@’作为开始和结尾，中间以空格进行分割成几组数字对，每一组数字对是‘x,y’的形式，具体含义为：以下例子分为-2,3和+2,5两组，-表示第一个变动前文件，+表示第二个变动后文件，逗号前数字表示变动的第几行，逗号后的文件表示变动了多少行，比如-2,3表示第一个文件从第二行开始，连续变动了3行，注意如果不是连续变动，会用多组数字对进行表示
     @@ -2,3 +2,5 @@
     ```

   - 第四部分表示显示具体的修改，其中‘+’表示在第一个文件添加的哪些行，‘-’表示在第一个文件删除的哪些行

4.撤销以执行的操作

- 任何一个阶段，都有可能想要撤消某些操作
- 注意，有些撤消操作是不可逆的，这是在使用 Git 的过程中，会因为操作失误而导致之前的工作丢失的少有的几个地方之一

```
# 提交完了才发现漏掉了几个文件没有添加，或者提交信息写错了。 此时，可以运行带有 --amend 选项的提交命令来重新提交
git commit --amend

# 提交后发现忘记了暂存某些需要的修改
git commit -m 'initial commit'
git add forgotten_file
git commit --amend
```

- 已经修改了两个文件并且想要将它们作为两次独立的修改提交， 但是却意外地暂存了它们两个，此时取消其中的一个操作为

```
# 可以这样来取消暂存，reset是个危险的命令，如果加上了--hard 选项则更加危险。 然而在上述场景中，工作目录中的文件尚未修改，因此相对安全一些。
git reset HEAD ***
```

- 撤销对文件的修改，方便地撤消修改，将它还原成上次提交时的样子

```
# git checkout -- <file> 是一个危险的命令，对那个文件在本地的任何修改都会消失——Git 会用最近提交的版本覆盖掉它。 除非确实清楚不想要对那个文件的本地修改了，否则请不要使用这个命令。
# 在Git中任何已提交的东西几乎总是可以恢复的，甚至那些被删除的分支中的提交或使用 --amend 选项覆盖的提交也可以恢复
git checkout -- CONTRIBUTING.md
git status
```