GIT 工具使用方法

一、基本操作

1.查看所有分支及当前所在分支

查看所有分支及当前所在分支 Bash | 收起

```
git branch -a # *号代表当前所在分支 终端不一样有可能显示方式也不一样 仅供参考
```

2.查看当前修改

查看当前修改 Bash | 收起

```
git status # 如果执行了 git commit 之后再无操作 那么git status 将显示没有变化
```

3.暂存所有修改

暂存所有修改 Bash | 收起

```
git add -A  #此时只是暂存 还没有合到远程库
```

4.为当前修改添加备注

为git add 添加备注 Bash | 收起

```
git commit -m'本次修改'
```

5.提交到远程

提交到远程代码库 Bash | 收起

```
git push # 如果报错  请执行 git push origin HEAD:refs/for/master 注：这是icode固定写法 并不通用所有git
```

注：本项目中需要评审才能合入到远程

6.从远程拉取最新代码

更新本地代码 Bash | 收起

```
git pull
```

7.查看历史提交记录

查看历史提交记录 Bash | 收起

```
git log
git log --oneline  #查看历史记录的简洁的版本
git log --graph    #查看历史中什么时候出现了分支、合并
git blame ${file}  #以列表形式查看指定文件的历史修改记录
```

8.修改暂存内容

修改没有提交的暂存内容 Bash | 收起

```
git add -A  #在没有git push之前 再次执行git add 就会覆盖上次add的内容 rm
git rm --cached +文件名 #这个命令不会删除物理文件，只是将已经add 进缓存的文件删除。
git rm --f + 文件路名 #这个命令不仅将文件从缓存中删除，还会将物理文件删除，所以使用这个命令要谨慎
git rm -r --cached ${file} #删除已经添加缓存的某一个目录下所有文件的话需要添加一个参数 -r
```

注：建议使用git add -A 来修改

9.git diff 对比

git diff命令后通常需要跟两个参数，参数1是要比较的旧代码，参数2是要比较的新代码。如果只写一个参数，表示**默认跟 workspace** 中的代码作比较。git diff 显示的结果为 第二个参数所指的代码在第一个参数所指代码基础上的修改

git ditt Bash | 收起

```
#git diff命令后通常需要跟两个参数，参数1是要比较的旧代码，参数2是要比较的新代码。如果只写一个参数，表示默认跟 workspace 中的代码作比较。git diff 显示的结果为 第二个参数所指的代码在第一个参数所指代码基础上的修改

git diff           #查看 workspace 与 index 的差别
git diff --cached  #查看 index 与 local repositorty 的差别
git diff HEAD      #查看 workspace 和 local repository 的差别
#HEAD 指向的是 local repository 中的代码最新提交版本

git diff HEAD^  #是比较 workspace 与最新commit的前一次commit的差异，与git diff HEAD的是不同的
git diff HEAD~2 #是比较 workspace 与上2次commit的差异，相当于 git diff HEAD~2 HEAD~0，注意两个HEAD的位置，diff显示的结果表示 参数2(HEAD~0) 相对于参数1(HEAD~2)的修改

^与~之间的区别
当存在多个分支时，^可以用来选择分支；
HEAD~i永远只选择第i级父节点的第一个分支；
HEAD~i^2选择第i级父节点的第二个分支；
以此类推；
HEAD^=HEAD^1=HEAD~1；
如果没有分支，只有一条主线，则HEAD^^^=HEAD^1^1^1=HEAD~3，
如果该级节点有第二个分支，则表示为：HEAD^^^2 = HEAD~2^2

git diff hash值 hash值 #比较两个hash值
```

10.回退版本

回退版本 Bash | 收起

```
git reset --hard HEAD   #恢复当前版本，删除工作区和缓存区的修改
git reset --soft HEAD^  #恢复上一个版本，保留工作区，缓存区准备再次提交commit
git reset --mixed HEAD  #恢复当前版本，保留工作区，清空缓存区
git reset --hard 1094a  #切换到特定版本号，并删除工作区和缓存区的修改

#场景1：修改仅存在工作区
git checkout -- readme.txt # 单文件
#场景2：修改存在暂存区、工作区
git reset HEAD readme.txt
git checkout -- readme.txt
#场景3：修改存在版本库、暂存区、工作区
git reset --hard 1094a

参数介绍
--hard  #清空工作区与缓存区，放弃目标版本后所有的修改
--soft  #保留工作区与缓存区，但是把版本之间的差异存放在缓存区，合并多个commit
--mixed #保留工作区清空缓存区，把版本之间的差异存放在工作区 列如：1、有错误的commit需要修改；2、git reset HEAD清空缓存区
```

二、创建仓库

1.初始化仓库

初始化仓库 Bash | 收起

```
git init #使用当前目录作为 Git 仓库，只需使它初始化。该命令执行完后会在当前目录生成一个 .git 目录。
git init ${directory} #使用指定的目录当做仓库，初始化后，会在 ${directory} 目录下会出现一个名为 .git 的目录，所有 Git 需要的数据和资源都存放在这个目录中,如果当前目录下有几个文件想要纳入版本控制，需要先用 git add 命令告诉 Git 开始对这些文件进行跟踪，然后提交：
git add *.c
git add README
git commit -m '初始化项目版本'
#以上命令将目录下以 .c 结尾及 README 文件提交到仓库中
注：linux系统中 commit 信息使用单引号 ' ' ,Windows 系统，commit 信息使用双引号 " "
```

2.使用git clone 从现有的git库中拷贝项目

git clone Bash | 收起

```
git clone ${git仓库}
示例:
git clone git://github.com/schacon/grit.git #执行该命令后，会在当前目录下创建一个名为grit的目录，其中包含一个 .git 的目录，用于保存下载下来的所有版本记录

如果要自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字:
git clone git://github.com/schacon/grit.git mygrit
```

3.设置git 配置

Bash | 收起

```
git config #显示当前配置信息
git config --list #显示当前配置信息

编辑配置文件:
git config -e #针对当前仓库
git config -e --global #针对系统上所有仓库

设置提交代码时的用户信息
git config --global user.name "runoob"
git config --global user.email test@runoob.com
```

三、分支管理

1.查看所有分支及当前所在分支

查看所有分支及当前所在分支 Bash | 收起

```
git branch -a # *号代表当前所在分支 终端不一样有可能显示方式也不一样 仅供参考
```

2.创建分支

创建分支 Bash | 收起

```
git branch ${branchname}
```

3.切换分支

切换分支 Bash | 收起

```
git checkout ${branchname}
```

4.创建分支并切换到此分支下面

创建并切换 Bash | 收起

```
git checkout -b ${branchname}
```

5.删除分支

删除分支 Bash | 收起

```
git branch -d ${branchname}
```

6.合并分支

合并分支 Bash | 收起

```
git merge ${需要被合并的branchname}

示例:
```

```
$ git branch
* master
  newtest
$ ls
README        test.txt
$ git merge newtest
Updating 3e92c19..c1501a2
Fast-forward
 runoob.php | 0
 test.txt   | 1 -
 2 files changed, 1 deletion(-)
 create mode 100644 runoob.php
 delete mode 100644 test.txt
$ ls
README        runoob.php     #合并完成后 test.txt 被删除
```

7.合并冲突解决方法

解决方法 Bash | 收起

```
在git 中可以通过add 和 commit 来解决分支合并冲突
git add
git commit
```

四、GIT标签

标签定义：如果你达到一个重要的阶段，并希望永远记住那个特别的提交快照，你可以使用 git tag 给它打上标签。

比如说，我们想为我们的 runoob 项目发布一个"1.0"版本。 我们可以用 git tag -a v1.0 命令给最新一次提交打上（HEAD）"v1.0"的标签。

-a 选项意为"创建一个带注解的标签"。 不用 -a 选项也可以执行的，但它不会记录这标签是啥时候打的，谁打的，也不会让你添加个标签的注解。 我推荐一直创建带注解的标签。

1.打标签

打tag Bash | 收起

```
git tag -a v1.0  #当你执行 git tag -a 命令时，Git 会打开你的编辑器，让你写一句标签注解，就像你给提交写注解一样。
现在执行 git log --decorate 就可以看到刚才的标签了
```

2.给已经提交的版本打tag

给已提交的版本打tag Bash | 收起

```
git tag -a ${version} ${历史提交id}  #历史提交id 可以用 git log 查看
```

3.指定标签信息命令

指定标签信息命令 Bash | 收起

```
git tag -a ${version} -m '标签信息'
```

4.查看所有tag

查看所有tag Bash | 收起

```
git tag
```

5.查看某个版本修改的内容

查看版本修改的内容 Bash | 收起

```
git show ${version}
```

6.删除标签

删除标签

Bash 收起

```bash
git tag -d ${version}
```