



说明:

- workspace: 工作区
- staging area: 暂存区/缓存区
- local repository: 版本库或本地仓库
- remote repository: 远程仓库

一个简单的操作步骤:

```
1 $ git init
2 $ git add .
3 $ git commit
```

我们经常需要多个人共同完成一个项目，使用git可以很清晰明了的知道每个人上传的改动，因此我们需要配置好自己的用户名方便他人知道是谁更改了项目

1、查看用户名和邮箱地址:

```
1 git config user.name
2 git config user.email
```

2、修改用户名和邮箱地址

```
1 git config --global user.name "xxxx"
2 git config --global user.email "xxxx"
```

3、如果我们是直接拉下代码库

```
1 git clone 链接名
```

那么我们这个文件本身已经就是一个git文件了

4、如果我们是本地已有的文件去与远程代码库相关联的话，需要执行以下步骤:

```
1 cd 文件/ #即需要进入这个文件
2 git init #将文件初始化为git文件
3 #与远程代码库添加链接
4 git remote add origin 代码库链接
5 #后续提交过程都一样
```

5、查看本地分支

```
1 git branch
```

注:名称前面加* 号的是当前的分支

6、在本地创建新的分支并切换到该分支上

```
1 git checkout -b private
```

等价于

```
1 git branch private      #在本地新建private分支
2 git checkout private    #切换到新分支上
```

7、查看远程分支

```
1 git branch -r
```

8、查看所有的分支（包括本地分支以及远程分支）

```
1 git branch -a
```

加上-a参数可以查看远程分支，远程分支会用红色表示出来（如果你开了颜色支持的话）

9、查看本地分支与远程分支的映射关系

```
1 git branch -vv
```

10、重命名本地分支

```
1 git branch -m <oldbranch> <newbranch>
```

11、删除本地分支

```
1 git branch -d branchname
```

12、删除远程分支

```
1 git branch -r -d origin/branchname
2 git push origin:branchname #删除后还需要推送到远程
3
```

13、查看当前远程仓库信息

```
1 git remote -vv
```

14、远程新建分支后，本地查看不到，使用以下命令同步

远程新建分支，本地在未创建此新分支前便已经clone下来，现在本地查看分支时没有发现远程新建的分支，使用如下命令更新，即可查看远程新建的分支

```
1 git remote #列出远程主机
2 git remote update origin --prune #更新远程主机origin整理分支
```

15、当我们手动在远程分支上建立了一个新的分支，本地也有一个新的分支时，想要本地的新分支提交到这个远程的新分支上时，我们需要新建本地分支与远程分支的关联

```
1 git branch --set-upstream-to=origin/远程分支名 本地分支名
```

eg：我的本地新建了一个分支 hui 远程新建了一个分支v2 把二者进行关联起来

```
1 git branch --set-upstream-to=origin/v2 hui
2 或者
3 git branch -u origin/v2 hui
4 如果此时已经在本地hui分支上，可以直接
5 git branch -u origin/v2
```

输出：Branch 'hui' set up to track remote branch 'v2' from 'origin'.

16、如果本地有分支，但是远程没有分支对应，如何把本地的分支提交到远程

假设有本地分支dev_name,远端没有该分支。此时push或者pull时，就不知道跟踪的是哪个分支？使用以下指令：

```
1 git push --set-upstream origin dev_name
```

推送后远程也会出现dev_name分支,二者建立连接，注意此时的dev_name并不是你给远程分支起的名字，而是根据本地的分支推送上去的远程分支。后续push和pull时，就不用指定分支。

17、本地没有某个分支，远程仓库有此分支，怎样拉取远端分支代码到本地分支？

```
1 git checkout --track origin/branch_name
```

此时，本地会自动创建分支branch_name与远端分支同名，并与远端分支branch_name关联。建议在弄分支的时候最好本地与远程的名字相同 便于区别

18、如果我们的本地文件已经关联了远程代码仓库，现在想关联新的代码仓库

一种方法是将原来的远程仓库重新命名，另一种是删除原来的远程仓库；二者选其一，然后再新关联现在的远程仓库即可

```
1 #方法一：重命名原来的远程仓库
2 git remote rename origin old-origin
```

```
3
4 #方法二：删除原来的远程仓库
5 git remote rm origin
6
7 #上面的二选一，然后重新指定新的源
8 git remote add origin [url]
```

19、撤销本地分支与远程分支的映射关系

```
1 git branch --unset-upstream
```

此时，当前的本地分支与远程分支解除关系

20、Git在本地新建分支后，可做远程分支关联。关联目的是，如果在本地分支下进行pull和push操作时，便不需要指定远程的分支。

21、新建本地分支与远程分支相关联

```
1 git checkout -b gpf origin/gpf # 新建本地分支gpf与远程gpf分支相关联
```

22、执行push推送代码

```
1 git push origin 本地分支名:远程分支名
```

23、本地分支push到远程分支

本地分支 v2 远程分支r2

如果没有建立关系时

```
1 git push origin v2:r2
2 或者
3 git push origin r2 #本地已经在v2分支上
```

如果已经建立关联 且目前的本地分支就在v2上 我们可以直接使用

```
1 git push
```

24、多人合作提交代码

```
1 在我们自己提交之前如果有别人提交了代码 我们需要先进行合并代码 再进行push
2 方案一：合并远程分支代码
3     git fetch origin
4     (git remote update有的时候可能需要同步一下远程和本地)
5     git merge origin/远程分支名
6 方案二：合并远程分支代码
7     git pull origin 远程分支名
8     (PS: 方案一和方案二选择一个即可)
9 执行push推送代码
10    git push origin 本地分支名:远程分支名
```

25、git fetch 与 git pull的区别

git fetch是将远程主机的最新内容拉到本地，用户在检查了以后决定是否合并到工作本机分支中。而git pull 则是将远程主机的最新内容拉下来后直接合并，即：git pull = git fetch + git merge，这样可能会产生冲突。

26、git fetch的常见命令如下：

```
1 git fetch <远程主机名> //这个命令将某个远程主机的更新全部取回本地，一般远程主机名为orig
2 git fetch <远程主机名> <分支名> //注意之间有空格,返回的是特定分支的更新
3 例如：返回origin主机的master分支的更新 git fetch origin master
```

取回更新后，会返回一个FETCH_HEAD，指的是某个branch在服务器上的最新状态，我们可以在本地通过它查看刚取回的更新信息：

```
1 git log -p FETCH_HEAD
```

可以看到返回的信息包括更新的文件名，更新的作者和时间，以及更新的代码（红色[删除]和绿色[新增]）。我们可以通过这些信息来判断是否产生冲突，以确定是否将更新merge到当前分支。

如果我们需要合并的话执行以下代码

```
1 git merge FETCH_HEAD
```

总结：fetch合并到分支需要两步

```
1 git fetch origin master //从远程主机的master分支拉取最新内容
2 git merge FETCH_HEAD //将拉取下来的最新内容合并到当前所在的分支中
```

27、git pull的常见用法

将远程主机的某个分支的更新取回，并与本地指定的分支合并

```
1 git pull <远程主机名> <远程分支名>:<本地分支名>
```

如果需要合并的本地分支就是目前的分支，则后面的本地分支名可以省略

28、还原代码至某个版本

```
1 git reset --hard 版本号
```

如果不加版本号，默认恢复上一个版本

29、合并分支到master上

首先切换到master分支上

```
1 git checkout master
```

如果是多人开发的话 需要把远程master上的代码pull下来

```
1 git pull origin master
```

然后我们把dev分支的代码合并到master上

```
1 git merge dev
```

30、查看状态

```
1 git status
```

31、git配置的一些其他的命令

```
1 git status # 查看状态
2 git log # 查看提交历史
3 git config core.ignorecase false # 设置大小写敏感
4 git config --global user.name "YOUR NAME" # 设置用户名
5 git config --global user.email "YOUR EMAIL ADDRESS" # 设置邮箱
```

32、git的提交

```
1 git diff # 查看变更内容
2 git add . # 跟踪所有改动过的文件
3 git add # 跟踪指定的文件
4 git mv # 文件改名
5 git rm # 删除文件
6 git rm --cached # 停止跟踪文件但不删除
7 git commit -m "commit message" # 提交所有更新过的文件
8 git commit --amend # 修改最后一次提交
```

33.查看历史

```
1 git log # 查看提交历史
2 git log -p # 查看指定文件的提交历史
3 git blame # 以列表方式查看指定文件的提交历史
```

34.撤销

```
1 git reset --hard HEAD # 撤销工作目录中所有未提交文件的修改内容
2 git reset --hard # 撤销到某个特定版本
3 git checkout HEAD # 撤销指定的未提交文件的修改内容
4 git checkout -- # 同上一个命令
```

```
5 git revert
```

```
# 撤消指定的提交分支与标签
```

35.分支与标签

```
1 git branch
```

```
# 显示所有本地分支
```

```
2 git checkout
```

```
# 切换到指定分支或标签
```

```
3 git branch
```

```
# 创建新分支
```

```
4 git branch -d
```

```
# 删除本地分支
```

```
5 git tag
```

```
# 列出所有本地标签
```

```
6 git tag
```

```
# 基于最新提交创建标签
```

```
7 git tag -a "v1.0" -m "一些说明" # -a指定标签名称, -m指定标签说明
```

```
8 git tag -d
```

```
# 删除标签
```

```
9 git checkout dev
```

```
# 合并特定的commit到dev分支上
```

```
10 git cherry-pick 62ecb3
```

36.合并与衍合

```
1 git merge
```

```
# 合并指定分支到当前分支
```

```
2 git merge --abort
```

```
# 取消当前合并, 重建合并前状态
```

```
3 git merge dev -Xtheirs
```

```
# 以合并dev分支到当前分支, 有冲突则以dev分支为准
```

```
4 git rebase
```

```
# 衍合指定分支到当前分支
```

37.远程操作

```
1 git remote -v
```

```
# 查看远程版本库信息
```

```
2 git remote show
```

```
# 查看指定远程版本库信息
```

```
3 git remote add
```

```
# 添加远程版本库
```

```
4 git remote remove
```

```
# 删除指定的远程版本库
```

```
5 git fetch
```

```
# 从远程库获取代码
```

```
6 git pull
```

```
# 下载代码及快速合并
```

```
7 git push
```

```
# 上传代码及快速合并
```

```
8
```

38.打包

```
1 git archive --format=zip --output ../file.zip master # 将master分支打包成file.zip
```

```
2 git archive --format=zip --output ../v1.2.zip v1.2 # 打包v1.2标签的文件, 保存在
```

```
3 git archive --format=zip v1.2 > ../v1.2.zip # 作用同上一条命令
```

39.远程与本地合并

```
1 git init
```

```
# 初始化本地代码仓
```

```
2 git add .
```

```
# 添加本地代码
```



```
3 git commit -m "add local source" # 提交本地代码
4 git pull origin master # 下载远程代码git merge master
5 git push -u origin master # 上传代码
```