# 前提

配置的是你个人的用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，所以会随更新内容一起被永久纳入历史记录：

git config --global user.name "name"

git config --global user.email example@example.com

如果用了 --global 选项，那么更改的配置文件就是位于你用户主目录下的那个，以后你所有的项目都会默认使用这里配置的用户信息。如果要在某个特定的项目中使用其他名字或者电邮，只要去掉 --global选项重新配置即可，新的设定保存在当前项目的 .git/config 文件里。

[【Git】删除某个全局配置项](https://www.cnblogs.com/zqunor/p/9055262.html)

1.查看Git所有配置

git config --list

2.删除全局配置项

(1)终端执行命令：

git config --global --unset user.name

(2)编辑配置文件：

git config --global –edit

3.进入项目.git目录，使用vi 编辑器进行修改删除

vi config

# 创建新项目

1．在git命令窗口中输入：touch README.md 会生成一个README.md文件

2．继续输入git init ， 提示Initialized empty Git repository in D:/project/.git说明创建一个空的仓库成功

3．继续输入git add README.md 回车 ，继续输入git commit -m “提交日志” 回车 （可一步到位：git commit -am “描述信息”）

4．继续输入git remote add origin ssh://xxxxxxxx ，Origin后面的地址为你创建的服务器地址，可以在页面中找到（初次提交文件时使用，在此提交可忽略）

5．继续输入git push -u origin master，根据提示输入密码：admin，输入密码时不会显示出来，直接输入完成按回车

**注**：

git push -u origin master 如果当前分支与多个主机存在追踪关系，则可以使用 -u 参数指定一个默认主机，这样后面就可以不加任何参数使用git push

# 二、修改文件后提交远程仓库

1．git status 查看git是否有修改内容需要提交

2．git add 指向需要提交的内容文件  
3．git commit –m “提交说明” 提交到本地库

当我们想要对上一次的提交进行修改时，我们可以使用git commit –amend命令。git commit –amend既可以对上次提交的内容进行修改，也可以修改提交说明

git commit –am “本次提交描述” 或 git commit –a –m“本次提交描述”，此命令会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一并提交上去。相当于git add . 与git commit –m “本次提交描述”两句操作合并为一句进行使用。

4．git push origin master 提交到远程仓库

git push的一般形式为 git push <远程主机名> <本地分支名>  <远程分支名>

例如 git push origin master：refs/for/master ，即是将本地的master分支推送到远程主机origin上的对应master分支， origin 是远程主机名，

    第一个master是本地分支名，第二个master是远程分支名。

（1）git push origin master如果远程分支被省略，如上则表示将本地分支推送到与之存在追踪关系的远程分支（通常两者同名），如果该远程分支不存在，则会被新建

（2）git push origin ：refs/for/master 如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到远程分支，等同于 git push origin --delete master

（3）git push origin如果当前分支与远程分支存在追踪关系，则本地分支和远程分支都可以省略，将当前分支推送到origin主机的对应分支

（4）git push如果当前分支只有一个远程分支，那么主机名都可以省略，形如 git push，可以使用git branch -r ，查看远程的分支名

# 三、分支操作

1. 新建一个本地分支：git checkout -b [branch]

2. 查看一下现在的分支状态：git branch星号(\*)表示当前所在分支

3. 把新建的本地分支push到远程服务器，远程分支与本地分支同名（当然可以随意起名）：git push origin [branch]:[branch]，使用git branch -a查看所有分支，会看到remotes/origin/[branch]这个远程分支，说明新建远程分支成功

4. 删除远程分支，推送一个空分支到远程分支，其实就相当于删除远程分支：git push origin :[branch]或git push origin --delete [branch]

5. 删除本地分支git branch -d [branch]

6. 删除本地的远程分支：git branch -r -D origin/branch

7. 切换分支 git checkout [branch]

# 四、版本回退和穿梭

1．在git中，我们用git log命令查看, 如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline参数；--graph标记会画出一个ASCII图展示commit历史的分支结构；--abbrev-commit 仅显示 SHA-1 的前几个字符,而非所有的字符；可以用 --reverse参数来逆向显示所有日志；查找指定用户的提交日志可以使用命令--auther=xxx；指定日期，可以执行几个选项：--since 和 --before，但是你也可以用 --until 和 --after

git log --graph --pretty=oneline --abbrev-commit

git log --graph --oneline --decorate

git log --pretty=oneline --abbrev-commit

git log --reverse --oneline

git log --author=Linus --oneline -5

三周前且在2018年9月18之后的所有提交，我可以执行这个，用了 --no-merges 选项以隐藏合并提交：

git log --oneline --before={3.weeks.ago} --after={2018-09-18} --no-merges

2．在git中，用HEAD表示当前版本, 上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100

当前版本回退到上一个版本，使用git reset命令：git reset --hard HEAD^

当你用 git reset --hard HEAD^回退到上个版本时，再想恢复到原来的版本，就必须找到原来版本的commit id。Git提供了一个命令git reflog用来记录你的每一次命令

HEAD指向的版本就是当前版本， Git允许我们在版本的历史之间穿梭，使用命git reset --hard commitId。

穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。

要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

提交后，用git diff HEAD – [file]命令可以查看工作区和版本库里面最新版本的区别

git diff 有两个主要的应用场景。

尚未缓存的改动：git diff

查看已缓存的改动： git diff --cached

查看已缓存的与未缓存的所有改动：git diff HEAD

显示摘要而非整个 diff：git diff --stat

3．git是如何跟踪修改的，每次修改，如果不用git add到暂存区，那就不会加入到commit中

git checkout -- file可以丢弃工作区的修改，命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是readme.txt修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次git commit或git add时的状态

git reset HEAD [file]可以把暂存区的修改撤销掉（unstage），重新放回工作区，git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新的版本

# 五、删除文件

1．你通常直接在文件管理器中把没用的文件删了，或者用rm命令删了Git知道你删除了文件，因此，工作区和版本库就不一致了：

一是确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit

一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本， git checkout -- filename git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”

2．如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项 –f：git rm -f <file>

3．如果把文件从暂存区域移除，但仍然希望保留在当前工作目录中，换句话说，仅是从跟踪清单中删除，使用 --cached 选项即可：git rm --cached <file>

4．git mv 命令用于移动或重命名一个文件、目录、软连接。对文件重名: git mv OriginName NewName

# 六、远程仓库操作

1．把本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程，由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令；

2．git push origin master把本地master分支的最新修改推送至GitHub，现在，你就拥有了真正的分布式版本库！

3．要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；关联后，使用命令git push -u origin master第一次推送master分支的所有内容；此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；

4．要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快

# 七、分支管理

1．创建dev分支，然后切换到dev分支 git checkout -b [name]

2．用git branch命令查看当前分支，创建分支：git branch <name>

3．切换分支 git checkout <name>

4．git merge [name]命令用于合并指定分支到当前分支，Fast-forward是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快；合并完成后，就可以放心地删除dev分支了 git branch -d [name]

5．合并分支产生冲突时，Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容；用带参数的git log也可以看到分支的合并情况；合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并，合并后，我们用git log看看分支历史：git log --graph --pretty=oneline --abbrev-commit

6．git还提供了一个stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作；用git stash list命令看看

7．git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；

另一种方式是用git stash pop，恢复的同时把stash内容也删了

可以多次stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令：git stash apply stash@{0} ；如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除

# 八、多人协作

1．要查看远程库的信息，用git remote；用git remote -v显示更详细的信息

2．推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支：git push origin [branch-name]

master分支是主分支，因此要时刻与远程同步；

dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

其他分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发

3．最新提交和当前推送的提交有冲突，解决办法也很简单，git已经提示我们，先用git pull把最新的提交从origin/[branch-name]抓下来，然后，在本地合并，解决冲突，再推送

4．在本地创建和远程分支对应的分支，使用git checkout -b [branch-name] origin/[branch-name]，本地和远程分支的名称最好一致；

5．git pull也失败了，提示no tracking information，原因是没有指定本地[branch-name]分支与远程origin/[branch-name]分支的链接，根据提示，设置[branch-name]和origin/[branch-name]的链接：用命令git branch --set-upstream-to [branch-name] origin/[branch-name]

git用(HEAD -> master)和(origin/master)标识出当前分支的HEAD和远程origin的位置

7．git rebase操作可以把本地未push的分叉提交历史整理成直线；目的是使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比

# 九、标签管理

1．git tag [name]就可以打一个新标签；可以用命令git tag查看所有标签；默认标签是打在最新提交的commit上的。如果忘了打标签，找到历史提交的commit id，然后打上就可以了

2．可以用git show [tag-name]查看标签信息；创建带有说明的标签，用-a指定标签名，-m指定说明文字：git tag -a v0.1 -m "描述" commitId

3．推送某个标签到远程，使用命令git push origin <tag-name>；一次性推送全部尚未推送到远程的本地标签：git push origin –tags

4．标签打错了，也可以删除：git tag -d v0.1，如果标签已经推送到远程，先从本地删除：git tag -d v0.9；然后，从远程删除。删除命令也是push，但是格式如下：git push origin :refs/tags/v0.9

# 十、配置别名

配置git的时候，加上--global是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用

git config --global alias.st status

git config --global alias.co checkout

git config --global alias.ci commit

git config --global alias.br branch

# 十一、可能问题

1．使用git时出现：warning: LF will be replaced by CRLF

windows中的换行符为 CRLF， 而在linux下的换行符为LF，所以在执行add . 时出现提示，解决办法：

$ rm -rf .git // 删除.git

$ git config --global core.autocrlf false //禁用自动转换

然后重新执行：

$ git init

$ git add .

2．Git error：

hint: Updates were rejected because the remote contains work that you do  
hint: not have locally. This is usually caused by another repository pushing  
hint: to the same ref. You may want to first integrate the remote changes  
hint: (e.g., 'git pull ...') before pushing again.  
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

解决办法：

（1）1．git pull origin master --allow-unrelated-histories

（2）git pull origin master

（3）git init（可忽略）

（4）git remote add origin ssh://git-server（可忽略）

（5）git add .（可忽略）  
（6）git commit -m 'testst'（可忽略）  
（7）git push -u origin master

3．Git远程仓库地址变更本地如何修改

通过命令直接修改远程地址

（1）进入git\_test根目录

（2）git remote 查看所有远程仓库， git remote xxx 查看指定远程仓库地址

（3）git remote set-url origin [git-server]

通过命令先删除再添加远程仓库

（1）进入git\_test根目录

（2）git remote 查看所有远程仓库， git remote xxx 查看指定远程仓库地址

（3）git remote rm origin

（4）git remote add origin git-server

直接修改配置文件

（1）进入project/.git

（2）vim config   
[core]   
repositoryformatversion = 0   
filemode = true   
logallrefupdates = true   
precomposeunicode = true   
[remote "origin"]   
url =git-server   
fetch = +refs/heads/\*:refs/remotes/origin/\*   
[branch "master"]   
remote = origin   
merge = refs/heads/master

修改 [remote “origin”]下面的url即可

4．git 更新代码到本地

正规流程

git status（查看本地分支文件信息，确保更新时不产生冲突）

git checkout – [file name] （若文件有修改，可以还原到最初状态; 若文件需要更新到服务器上，应该先merge到服务器，再更新到本地）

git branch（查看当前分支情况）

git checkout remote branch (若分支为本地分支，则需切换到服务器的远程分支)

git pull

若命令执行成功，则更新代码成功！

快速流程

上面是比较安全的做法，如果你可以确定什么都没有改过只是更新本地代码

git pull (一句命令搞定)

git branch 看看分支

git checkout aaa 切换分支aaa

git branch aaa 创建aaa分支

git checkout -b aaa 本地创建 aaa分支，同时切换到aaa分支。只有提交的时候才会在服务端上创建一个分支

5．git如何撤销所有本地修改

git checkout . #本地所有修改的。没有的提交的，都返回到原来的状态

git stash  #把所有没有提交的修改暂存到stash里面。可用git stash pop回复。

git reset --hard HASH #返回到某个节点，不保留修改。

git reset --soft HASH#返回到某个节点。保留修改

**注**：

git checkout -b develop origin/CC\_vivo\_v1.0

这表示从主线上拉一个本地分支develop出来，并切换到develop分支

生成修改的bug文件 git diff --full-index --binary > bug.patch

6．VI编辑器

vi readme.txt 进入VIM编辑器,可以新建文件也可以修改文件 如果这个文件，以前是没有的，则为新建，则下方有提示为新文件。

按ESC键 跳到命令模式，然后：

:w 保存文件但不退出vi

:w file 将修改另外保存到file中，不退出vi

:w! 强制保存，不推出vi

:wq 保存文件并退出vi

:wq! 强制保存文件，并退出vi

:q不保存文件，退出vi

:q! 不保存文件，强制退出vi

:e! 放弃所有修改，从上次保存文件开始再编辑