git 配置

git config --global user.name "Your Name"

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

可以直接使用命令 gitk 使用图形界面来使用git，以及更好查看Git仓库的变更。

对于任何Git命令，都可以使用-h参数查看详细用法。

填空

1、创建版本库的过程：

项目目录还未创建的情况下，可以：

第一步，进入想要创建项目目录的上级目录。

第二步，使用命令 git init “project name” 即可在当前目录下创建一个项目仓库

如果项目目录已经创建，可以：

第一步，进入项目目录。

第二步，通过 git init 命令把这个目录变成Git可以管理的仓库。

Git把仓库建好后，可以发现当前目录下多了一个.git的目录（隐藏的，使用ls -al可以看到），这个目录是Git来跟踪管理版本库的，随着管理版本的推进，这个目录会变得越来越大。

在版本库中，总会生成一些文件，这些文件有可能是库文件、编译生成的文件等等，对于这些文件，你并不希望git来追踪并管控，因为这会使得 .git目录无意义的增大。因此，你可以通过在版本库的根目录下创建名为 .gitignore的文件来告诉git忽略指定类型的文件的修改。

2、将当前对于版本库中文件所做的修改进行提交，需要两步，使用的命令是：

第一步： git add ./<filename>

第二步： git commit -m “<本次修改的描述信息>”

3、对于git已经追踪的某个文件，如果想要重新对文件进行命名并保持git对它的追踪，可是使用命令 git mv "old filename" "new filename". 如果想要修改对于刚刚提交的commit的描述信息，可以使用命令 git commit --amend; 如果想要修改历史中某次commit的描述信息，则需要使用 git rebase -i “commit id” 来进入交互页面重新对整个commit树进行变基重组，这里的基准commit id是想要修改的commit的父节点 id。想要修改描述信息的commit选择reword命令，保持不变的commit选择pick命令。使用这个交互页面，也可以完成将多个commit整合为一个commit，需要将被合并的commit选择squash命令。如果这多个commit不是连续的，需要在交互页面里边调整顺序使他们为连续的。

（如果你的commit已经推送到远端，不要轻易rebase）

4、使用 git status 命令查看版本库最新状态，可以显示最新状态与最近提交版本的变化情况。这个变化包括工作区和暂存区相对于上一次提交的变化。

5、为了进一步显示出版本库最新状态具体变化情况细节，可以使用 git diff <filename> 命令查看。具体说明此命令对比的两个版本对象分别是什么：git diff 将会对比当前工作台的文件版本与最近一次保存的版本之间的修改状态。所谓最近一次保存的版本是指最新的git add或者git commit。如果暂存区有此文件的一个版本还未提交，则对比工作台的文件版本与暂存区的版本之间的修改状态；如果暂存区无此文件的未提交版本，则对比工作台的文件版本与版本库的HEAD指向的版本之间的修改状态。

如果想要将最新的工作区状态与最近提交的版本对比，可以使用命令：

git diff HEAD -- <filename>。

如果想要查看到的是暂存区和HEAD的不同，可以使用的命令：

git diff --cached <filename>。

如果想要查看历史的某两个提交的版本的差异对比，可以使用命令：

git diff “commit id1” “commit id2” -- <filename>。

6、当想要查看历史提交的版本记录时，使用命令 git log 。当历史提交版本很多，我们可使用参数 --oneline 来简化显示，或者使用参数 -[数字] 来查看最近的几个提交信息。命令默认查看当前分支的提交历史，参数 --all 可以查看所有分支的提交。可以使用参数 --graph 来图形化显示版本演化的过程。Git的每个提交版本都对应着一个commit id，Git的commit id跟SVN递增的提交号不同，它是一串随机的十六进制数字。

7、在Git中，用HEAD表示当前指向的提交版本，如果HEAD指向的提交版本不是某个分支的最新提交版本，这种状态称为分离头指针状态，在这种状态下产生的提交不与任何分支挂钩，很容易被Git当作垃圾清理。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。

8、如果我们需要回退到某一个版本，需要使用命令 git reset ，

如果我们需要回退到某个特定的版本,说明各个参数的具体作用：

git reset --mixed/hard/soft <版本号> ，

--mixed 是reset的默认参数。它将重置HEAD到另外一个commit,并且清空暂存区以便和HEAD相匹配，但不会改变任何工作区的内容。

--hard 将重置HEAD到另外一个commit,并且清空暂存区以便和HEAD相匹配, 而且工作区的内容也将全部回退到reset的版本号所对应的内容，此时暂存区和工作区中所有未提交的修改都将会丢失无法找回。

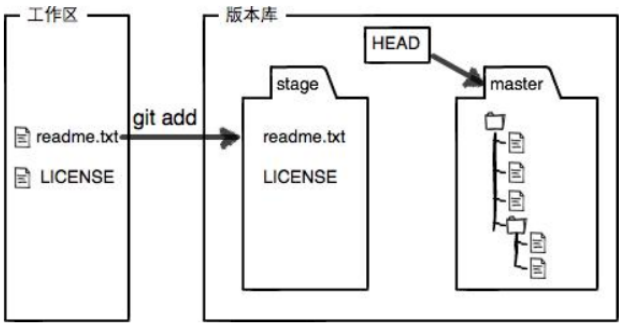
--soft 将重置HEAD到另外一个commit,并且此时暂存区保存的修改不会丢失,而且工作区的内容也不会发生任何改变。

（参考：https://www.cnblogs.com/kidsitcn/p/4513297.html）

9、假如版本提交记录A->B->C,使用git reset回退到B版本后，使用git log还能查看到C的提交记录吗？ 不能 如果回退到了某个旧版本后又想恢复到新版本，但git log无法查看到所有提交版本，如何查找你所需要的版本号： git reflog 。

10、工作区（Working Directory）是指：当前能够看到的目录和其中的文件；工作区有一个隐藏目录 .git ，是Git的版本库。Git的版本库里存了很多东西，其中最重要的就是称为 index 的 暂存区 ，还有Git为我们自动创建的第一个分支 master 。

11、我们把文件往Git版本库里添加的时候，是分了两步执行，说明这两步分别将文件修改添加到了什么地方：第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。你觉得为什么要设置暂存区，而不是直接提交？暂存区其实是用来保存暂时的修改的，如果你觉得对文件的修改还没必要构成一次提交，但又想要保存当前的修改防止丢失或者想在当前的修改上保存一个版本，以防止后边的修改失败后回退到这个断点，那么就可是使用git add将当前已有的修改保存在暂存区。commit可以一次提交很多文件，所以在提交之前可以多次add不同的文件，add相同文件多次，提交后保存的版本是最后一次add的版本。假设你在最后add之后又对文件进行了一些修改，但是并没有进行add，请问进行commit的版本中包含这些最后的修改吗？不包括。



12、如果需要丢弃工作区的修改，使用命令：git restore ./<filename>

在以下几种情况下，此命令将会使得工作区返回到哪个版本：

①文件自修改后还没有被放到暂存区 文件将会回退到HEAD指向的版本；

②文件的修改已经添加到暂存区，之后又作了修改回退到暂存区保存的版本；

③文件进行了修改后，还没有添加到暂存区，版本库进行了mixed类型的回退_文件将会回退到HEAD指向的版本_；

4.文件的修改添加到暂存区，版本库进行了soft类型的回退__文件将会回退到暂存区保存的版本_。

也就是说 restore 丢弃修改，恢复文件到最新的一次保存的版本。

注意：1. 对于情况③的说明：假设版本库是A->B->C，此时文件的修改还没有加入暂存区，进行了mixed类型的版本回退到B，此时使用restore后工作区的文件将会丢弃修改，直接恢复到版本B的状态。

2. 对于版本比较老的 git，丢弃修改使用的是checkout命令，但新版为了明确区分丢失修改和切换分支，丢弃修改已经使用了restore命令。如果你的git提示你用checkout丢弃修改，可以根据提示的checkout格式输入相应命令，但建议更新你的git。新版本仍然支持checkout相关的格式命令，这里不再介绍。

13、假如修改已经添加到了暂存区，但还没有提交。此时你需要丢弃修改，可以使用的方法是：使用命令____git restore --staged <filename>___将保存到暂存区的修改撤销掉，然后使用11题丢弃修改命令即可。这个命令只是删除了保存在暂存区中的版本断点，并不会对工作区产生影响。注意：这个命令只是将保存在暂存区中的版本断点丢弃，其中对文件的修改并不会恢复到工作区。如果你在保存版本断点后又对文件重新进行了修改，则这个命令将会导致文件无法再恢复到断点状态。

14、在Git中，删除也是一个修改操作。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，此时，可以使用命令 git restore <filename> 进行恢复，恢复后的文件的版本是 最近一次保存的版本(add/commit) 。如果确定不需要恢复，要删除此文件并且在版本库中将它删除，可以使用命令___git rm <filename>___。如果使用了在版本库中删除的命令，使用git restore <filename>是否还可以进行恢复？_不能_，此时如何恢复这个文件？git rm <filename> 这个命令其实进行了两个动 作，即删除并添加到暂存区。此时暂存区记录到这个文件的修改版本是删除，这时用git restore <filename> 恢复的版本即为暂存区的删除版本（最新保存的修改点），所以无法找回文件。正确的方法是首先使用 git restore --staged <filename> 撤销暂存区的版本，然后使用git restore <filename> 恢复文件。此时文件只能恢复到HEAD指向的版本，自从HEAD之后的修改将会丢失无法找回，请慎重使用。 此后使用命令 git reset [版本号] 回退到拥有此文件的某个版本，回退后工作区中是否会出现这个文件？为什么？不会，因为reset默认是mixed参数，不会改变工作区的文件。此时使用git status查看会发现文件的删除状态，再使用git restore即可恢复工作区这个文件到HEAD版本。

15、你已经在本地创建了一个Git仓库后，又想在GitHub创建一个Git仓库，并且让这两个仓库进行远程同步，这样，GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作：

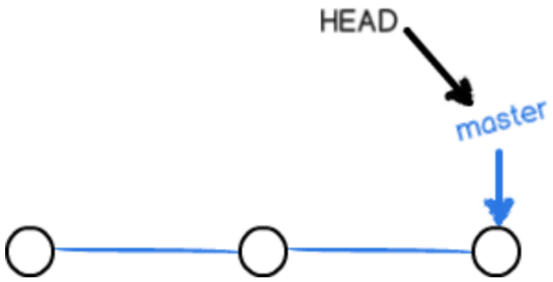
首先，你需要在GitHub上创建一个仓库。

接着，使用命令 git remote add “远程仓库命名” “远程链接(SSH/HTTP)” 将远程库和本地库关联起来。之后可以使用 git remote -v 查看你都关联了哪些远程库。

最后，使用命令 git push “远程仓库名” “远程分支名称” 将分支推送到远程。

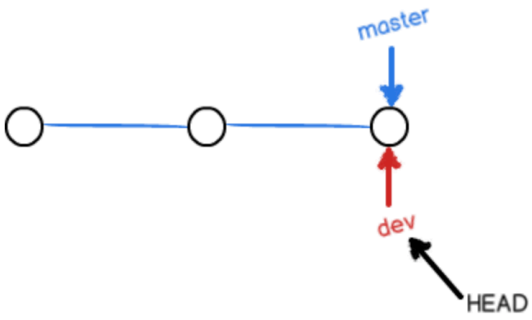
16、我们可以使用命令 git clone "远程仓库链接(SSH/HTTP)" 将一个远程库克隆到本地，git支持SSH协议和HTTP协议。使用 git pull 可以将远程分支的最新修改拉取到本地，并与本地的修改合并，可以使用命令 git branch --set-upstream-to <branch-name> origin/<branch-name> 创建本地分支和远程分支的链接关系。

17、在Git里，初始化后默认所在的分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。每次在master分支上的提交都会使得master分支向前移动一步，这样，随着你不断提交，master分支的线也越来越长。



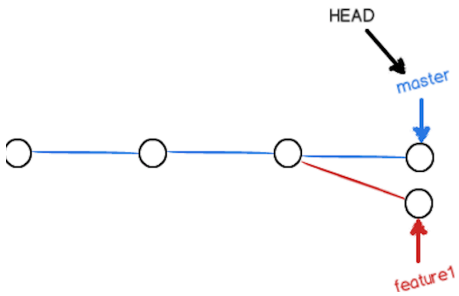
18、当你想要开发一个新功能时，在功能开发完之前并不想影响项目的内容和功能，此时就应该重启一个分支，这段时间所有的修改的提交都应该在这个分支上进行。我们可以通过命令 git branch "branch_name" "commit id" 来创建一个新的基于所给commit id的分支，并通过 git switch “branch_name” 来切换到创建的分支上。我们可以使用命令 git switch -c "branch_name" 来合并以上两步操作，完成创建并转换。完成开发后，如果测试功能正常，可以在master分支上，使用命令 git merge “branch_name” 来将某个分支上的内容合并到主分支上。合并完成后，此时如果想要删除已经无用的分支，可以使用命令 git branch -d "branch_name"。-d参数只能删除完全合并过的分支，如果想要删除的分支上存在并未被合并过的修改，则会删除失败，这时，如果你确定要放

弃在当前删除分支上的修改并删除分支，可以使用参数 -D 。我们还可以使用命令 git push origin -d "branch_name" 来删除远程库上的某个分支。



19、分支之间是独立的，但是工作台和暂存区却是公用的。假如现在我们在分支1上创建了文件test1并修改了内容，修改完成后添加到暂存区并提交；我们又在分支2上创建了文件test2并修改了内容，修改完成后添加到暂存区并提交。请问当我们再次切换到分支1上时是否能看到文件test2？不能。 如果我们在分支2上创建了文件2之后，对其添加或未添加到了暂存区，但并没有对test2进行提交，请问当我们再次切换到分支1时使用git status能够在看到test2的修改变化吗？能。假如我们分支1和分支2共享一个文件test，请问在分支2上对文件做了相应的修改，提交后切换回分支1，能够看见在分支2上的修改？不能。如果做了修改后不提交呢？能。因此可以得出结论：工作区和暂存区是公用的，如果工作区和暂存区上的修改没有被某个分支提交，则其不属于任何分支，所有分支均可见，也可提交。

20、如果两个不同的分支分别对同一个文件做了修改并进行了提交，则此时git状态如下：



此时如果在某个分支上合并另一个分支是否会产生冲突？解决的办法是什么？会产生冲突，冲突的文件中将会在冲突的地方保存两个分支上均存在的修改，此时需要手动修改冲突后再提交。

21、在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master 分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；干活都在 dev 分支上，也就是说，dev 分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。团队合作的分支看起来就像这样：

22、版本库又名仓库，英文名repository，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

23、如果在开发过程中，你必须清空你的工作台和暂存区来处理一个紧急修复bug的任务，然而你当前的工作还并不想构成一个提交，此时可以使用命令：

git stash。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。可以使用命令：git stash list 来查看你当前储藏起来的各个工作区。如果我们需要恢复工作区到某个stash状态，有两种方式：

一种是用 git stash apply stash@{序号} 恢复，恢复后stash内容并不删除，你需要用 git stash drop stash@{序号} 来删除；

另一种方式是用 git stash pop stash@{序号} 恢复的同时把stash内容也删了。

git stash在恢复的过程中会恢复暂存区内容吗? git stash命令会将工作区和暂存区的修改合并后保存，同时清空暂存区状态，如果你暂存区有已经添加的未提交修改，git stash会导致这些修改丢失，git stash pop恢复现场后不会恢复暂存区状态。

24、在master分支上修复了bug后，dev分支是早期从master分支分出来的，所以，这个bug其实在当前dev分支上也存在。因此，我们可以通过命令

git cherry-pick “提交版本号” 来将某个提交所做的修改“复制”到当前分支。这个命令只会复制这个提交所做的修改，并不是把整个master分支merge过来。

25、在版本库中打标签tag的意义在于：tag就是一个让人容易记住的有意义的名字，它跟某个commit绑在一起。当我们需要寻找到某个版本时容易查找。在git中对某个提交打标签的方法是：切换到需要打标签的分支，使用 git tag “标签名” <版本号>；版本号为可选参数，默认为打在当前分支最新提交的commit上的。git tag 的可选参数 -a 用来指定标签名， -m 用来添加标签的描述信息。可以使用 git tag 来查看所有标签。可以使用 git show “标签名” 来查看某个标签的详细信息。如果标签打错了，可是使用 git tag -d “标签名”删除某个标签。命令 git push origin <tagname> 可以推送一个本地标签到远程库；命令 git push origin --tags 可以推送全部未推送过的本地标签；命令git push origin :refs/tags/<tagname>可以删除一个远程标签。