windows git bash常用指令：

git init 把当前目录变成git可以管理的仓库（多了一个.git隐藏目录）

git status 查看当前工作区状态

git add

把对文件的修改从工作区提交到暂存区

git add . 提交所有文件的更改到暂存区

git rm 把对文件的删除更改提交到暂存区（同时删除了此文件）

git commit -m "information" 把所有文件添加到master分支并添加附注

git log 查看提交历史

git log --oneline 把每条信息用一行来输出

git reflog 查看命令历史

git reflog --oneline 把每条信息用一行来输出

git log --graph 查看分支合并图

git reset --hard 回退到所指定id的版本

git reset --hard HEAD^ 回退到上一个版本，HEAD指向的版本是当前版本

git reset HEAD 撤销暂存区的修改

git diff 工作区和暂存区(stage)的比较

git diff --cached 暂存区(stage)和分支(master)的比较

git diff HEAD -- 工作区和分支(master)的比较

touch 新建一个文件

cat 输出文本内容

vi 调用vi编辑器实现文本内容的修改

vim 调用vim编辑器实现文本内容的修改

git checkout -- 还原工作区的更改或删除（实际上用暂存区替换工作区的版本）

git checkout HEAD -- 还原工作区的更改或删除（实际上用master分支替换工作区的版本）

git remote add origin git@server-name:path/仓库名.git 关联一个远程库（通过先本地init方式的操作）

git clone git@sever-name:path/仓库名.git 从远程库克隆一个本地仓库

git push -u origin master **第一次**推送master分支的所有内容（通过先本地init方式的操作）

git push origin master 当远程仓库master分支与本地master分支已关联时的简化操作

git checkout -b dev 创建了一个叫dev的分支，并切换到dev分支（HEAD指向dev）

加上`-b`参数表示创建并切换等同于git branch dev git checkout dev

git checkout master 切换回master分支（HEAD指向master）

git branch 查看当前所有分支，当前指向的分支前有一个`*`号

git branch -d dev 删除了dev分支（HEAD不指向dev分支时）

git merge dev 把dev分支的内容合并到当前分支(master) （合并后不要忘记删除无用的dev分支）

git stash 藏匿当前工作台，往堆栈推送一个新的储藏

git stash pop 恢复最近的一次藏匿(stash) 回到原工作台，并删除这一次stash

git stash list 打印栈中所有信息 即查看当前所有stash

git stash clear 清空git栈 即清空所有暂存的stash

git remote -v 查看远程库的详细信息

git push origin master 把该分支(master)推送到远程库对应的远程分支上

git tag 给当前分支当前版本打上标签

git tag 给当前分支的对应commit版本打上标签

git tag -a -m "information" 创建带有附注的标签 -a指定标签名 -m指定说明文字

git tag 查看所有标签

git tag -d 删除该标签

git show 查看此标签信息(commit id，author，data，附注...)

git push origin 推送某个本地标签到远程

git push origin --tags 一次性推送全部尚未推送到远程的本地标签

删除远程标签:

```
$git tag -d <tagname>       //先删除本地标签
$git push origin :refs/tags/<tagname>   //再删除远程标签
```

从本地库到远程库与远程到本地库的区别：

从本地到远程方式的remote操作，不会把远程master分支和本地master分支关联。由于远程库是空的，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。

从远程到本地的clone操作，会自动把远程master分支和本地master分支关联，所以把内容push到远程仓库时，只需*$ git push origin master*而不用 加参数-u。

从本地push到远程库的一个问题:

在（Create a new repo）创建远程仓库的时候，如果你勾选了*Initialize this repository with a README*（就是创建仓库的时候自动给你创建一个README文件），那么到了你将本地仓库内容推送到远程仓库（git push -u origin master）的时候就会报一个*failed to push some refs to git@github.com:sever-name:path/仓库名.git*。 **这是由于你新创建的那个仓库里面的README文件不在本地仓库目录中**，这时我们可以通过以下命令将内容合并： *$ git pull --rebase origin master* 这时候就不会报错了。

为什么提倡多用分支然后合并到主分支?

因为主分支通常需要进行多次改进，在开发过程中没人希望主分支上有各种各样的测试代码，所以就需要再创建一个分支，在新分支上开发最后与主分支进行合并。

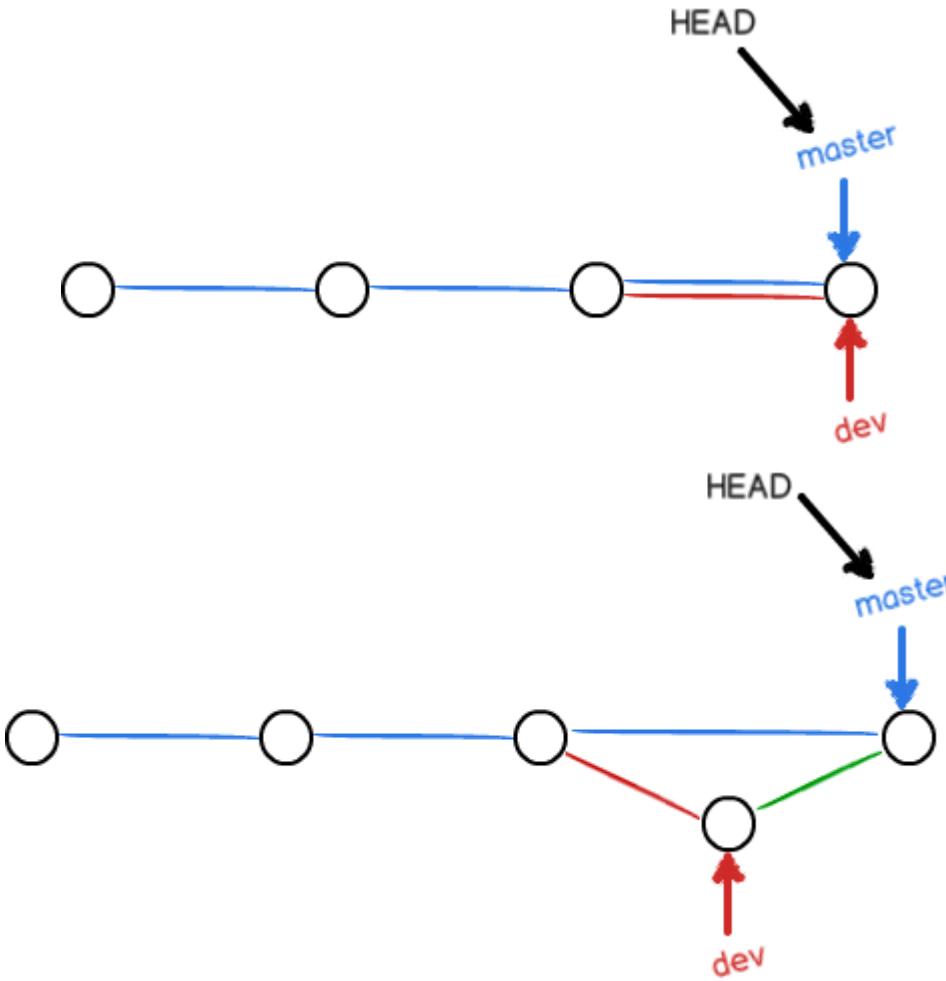
为什么可以用HEAD实现对master分支的操作?

此时HEAD类似于指向`master`的指针一开始的时候，`master`分支是一条线，Git用`master`指向最新的提交，再用`HEAD`指向`master`，就能确定当前分支，以及当前分支的提交点

当前master分支比远程的master分支要超前1个提交的含义:

当前master分支: A->B->C->D 远程master分支: A->B->C

Fast forward合并分支和不用fast forward方式合并的区别:



为什么需要git stash来暂存区的修改藏匿起来?

git stash可以把修改过的被追踪的文件和暂存的变更保存到一个未完结变更的堆栈中，

使用情景：①想pull最新的代码，又不想增加commit记录。

②需要修复紧急bug，又不想在当前分支commit还未完成的版本，如果强制转换到master分支，会提示 `Please, commit your changes or stash them before you can switch branches.` 通俗来说就是暂存区是公用的，每个分支都是使用相同的工作台。

git stash不针对特定的分支，切换分支后，stash内容不变，所以弹出时要小心!

值得一提的是，如果有已跟踪并修改的文件但未add时可以stash，但会自动把未add的更改add到stage

中文显示不正常的解决方法:

因为git中的中文用八进制来显示，要显示中文的话，在bash命令行内输入:

```
git config --global core.quotepath false
```

core.quotepath设为false的话，就不会对0×80以上的字符进行quote。中文显示正常。

git中出现.swp不可见文件的解决办法：

为什么会生成.swp文件?

正常情况：当你打开一个文件时，vi会自动生成一个.swp文件，文件名为[filename.**].swp，如果你正常退出，.swp文件会自动删除。

非正常情况：1、当用多个程序编辑同一个文件

2、非常规退出(如：当你强行关闭vi时，比如电源突然断掉或者使用了Ctrl+Z，vi便会自动生成一个.swp文件，当你下次编辑时就会出现一些提示)

解决方法:

```
$vim -r <filename>          用于恢复
```

```
$rm .{filename}.swp         然后把.swp文件删除
```

git中vi和vim编辑文件的区别及常用操作：

它们都是多模式编辑器，不同的是vim 是vi的升级版本，它不仅兼容vi的所有指令，而且还有一些新的特性在里面。

1、多级撤消 我们知道在vi里，按 u只能撤消上次命令，而在vim里可以无限制的撤消。 2、易用性 vi只能运行于unix中，而vim不仅可以运行于unix,windows ,mac等多操作平台。 3、语法加亮 vim可以用不同的颜色来加亮你的代码。 4、可视化操作 就是说vim不仅可以在终端运行，也可以运行于x window、 mac os、 windows。 5、对vi的完全兼容 某些情况下，你可以把vim当成vi来使用。

常用操作:

:w 保存编辑后的文件内容，但不退出vim编辑器

:w! 强制写文件，即强制覆盖原有文件。

:wq 保存文件内容后退出vim编辑器

:wq! 强制保存文件内容后退出vim编辑器

ZZ 使用ZZ命令时，如果文件已经做过编辑处理，则把内存缓冲区中的数据写到启动vim时指定的文件中，然后退出vim编辑器。否则只是退出vim而已

:q 在未做任何编辑处理而准备退出vim时，可以使用此命令。如果已做过编辑处理，则vim不允许用户使用":q"命令退出，同时还会输出下列警告信息：No write since last change (:quit! overrides)

:q! 强制退出vim编辑器，放弃编辑处理的结果

多人协作的工作方式：

1. 首先，可以试图用`git push origin branch-name`推送自己的修改；
2. 如果推送失败，则因为远程分支比你的本地更新，需要先用`git pull`试图合并；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用`git push origin branch-name`推送就能成功！

   如果`git pull`提示"no tracking information"，则说明本地分支和远程分支的链接关系没有创建，用命令`git branch --set-upstream branch-name origin/branch-name`。

如果想查看某个标签状态下的文件：

方法一：git reflog回退版本

方法二：1.git tag 查看当前分支下的标签

2.git checkout v0.21 此时会指向打v0.21标签时的代码状态，（但现在处于一个空的分支上）tag相当于一个快照，不能在这个tag的基础上修改文件

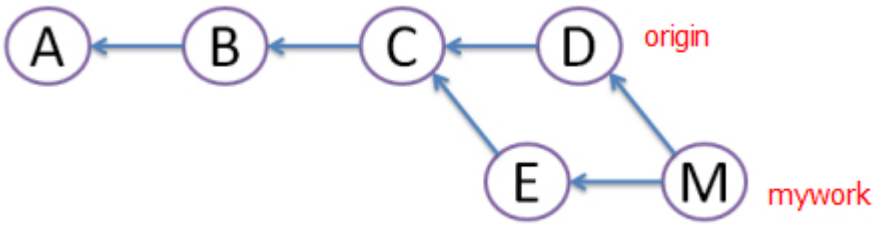
3.cat test.txt 查看某个文件

如果要做修改的话可以从该tag创建一个分支：`git checkout -b branch_name tag_name`

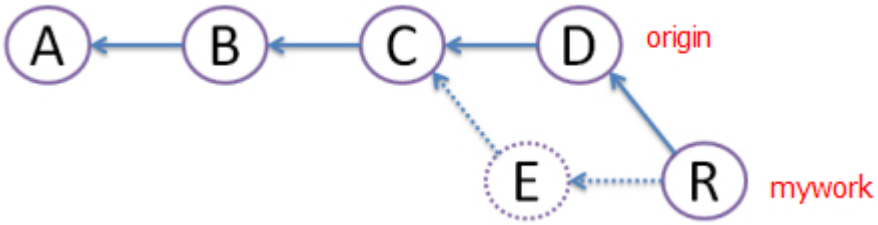
merge和rebase的区别(摘自cnblogs)：

```
git pull = git fetch + git merge
git pull --rebase = git fetch + git rebase
```

1、git merge 用git pull命令把"origin"分支上的修改pull下来与本地提交合并（merge）成版本M，但这样会形成图中的菱形，让人很困惑。



2、git rebase 创建一个新的提交R，R的文件内容和上面M的一样，但我们将E提交废除，当它不存在（图中用虚线表示）。由于这种删除，小李不应该push其他的repository.rebase的好处是避免了菱形的产生，保持提交曲线为直线，让大家易于理解。



在rebase的过程中，有时也会有conflict，这时Git会停止rebase并让用户去解决冲突，解决完冲突后，用git add命令去更新这些内容，然后不用执行git-commit,直接执行git rebase --continue,这样git会继续apply余下的补丁。 在任何时候，都可以用git rebase --abort参数来终止rebase的行动，并且mywork分支会回到rebase开始前的状态。