git教程(5) 分支管理(3)

一 多人协作
1 当你从 远程仓库 克隆【本地、远程的 master 会自动对应起来！！】时， 实际上，git会自动把本地的 master分支 和 远程的 master分支 对应起来。
且 远程的仓库的默认名称是 origin 。

git remote 会输出远程仓库信息【即 远程仓库的名字, git remote -v 去显示更详细的信息！！！】

```
$ git remote
origin
```

git remote -v

```
$ git remote -v
origin  git@github.com:michaelliao/learngit.git (fetch)
origin  git@github.com:michaelliao/learngit.git (push)
```

注意：上面显示了可以 抓取 和 推送 的origin地址。
如果 没有推送的权限， 就会看不见 push地址！！！

2 推送分支【如 git push origin dev,如果 本地、远程已经绑定，以后的 “推push、拉pull“ 可直接简写成 git push、 git pull 吧？？！】

2.1 推送分支的定义：
把该分支上 的 所有本地的修改 同送到远程仓库！！！
推送时，需要制定本地分支【本地？？？ 就是说可以在 d1 分支上执行 git push origin d2 将本地的 d2分支所做的修改 同步到远程仓库的 d2分支上？？ 不用切换分支吗？？】
，这样git就会自动的把该分支推送到 远程仓库 对应的 远程分支上 。

```
$ git push origin master
```

如果要推送其他分支，比如`dev`，就改成：

```
$ git push origin dev
```

2.2 哪些分支是需要进行推送的呢？？
master分支是 主分支，因此需要时刻与远程保持同步【如果不同步会怎么样？？ 合并产生冲突？？】
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分支只用于本地修复bug，就没必要推到远程了【？？修复好bug合并到 dev，在推送到远程？？ 还是啥原因？？】
feature分支是否推送到远程，取决于 你是否和你的小伙伴 合作、协作 开发！！！

总之，就是在 git中，分支完全可以自己藏着玩，是否推送可以取决于我们的心情！！

3 抓取分支
3.1 多人协作时，大家往往都会向 master、dev 分支推送自己的修改。

3.2 现在模拟一个你的小伙伴，可以在另一台电脑（注意要把 SSH Key添加到 Github） 或者同一台电脑的另一个目录下。

```
$ git clone git@github.com:michaelliao/learngit.git
Cloning into 'learngit'...
remote: Counting objects: 40, done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 40 (delta 14), reused 40 (delta 14), pack-reused 0
Receiving objects: 100% (40/40), done.
Resolving deltas: 100% (14/14), done.
```

3.3 你的小伙伴从远程库 clone时，默认情况下，他只能看见本地的 master 分支【为啥？？？】。

```
$ git branch
* master
```

3.4 小伙伴如果要在 dev分支上进行开发，就必须 创建远程 origin 的dev分支到本地，于是他用命令 git chckeout -b dev origin/dev【多出的 origin/dev 是进行了某个方向的绑定，如 push？？】 去创建本地的 dev分支。

```
$ git checkout -b dev origin/dev
```

3.5 现在子啊 dev开发完了，要把 本地dev所做的的修改 推送到 远程上：

```
$ git add env.txt

$ git commit -m "add env"
[dev 7a5e5dd] add env
 1 file changed, 1 insertion(+)
 create mode 100644 env.txt

$ git push origin dev
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 308 bytes | 308.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:michaelliao/learngit.git
   f52c633..7a5e5dd  dev -> dev
```

3.6 小伙伴已经成功向 远程的 origin/dev分支 提交了他的修改，碰巧你也同样对同样的文件做了修改【操作前没有 拉取小伙伴的最新修改】，并试图推送时会发生错误！！

```
$ cat env.txt
env

$ git add env.txt

$ git commit -m "add new env"
[dev 7bd91f1] add new env
 1 file changed, 1 insertion(+)
 create mode 100644 env.txt

$ git push origin dev
To github.com:michaelliao/learngit.git
 ! [rejected]        dev -> dev (non-fast-forward)
error: failed to push some refs to 'git@github.com:michaelliao/learngit.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

3.7 如上、报错，因为小伙伴的最新修改和你试图推送的提交有冲突。
git已经提示我们了，先用 git pull 把小伙伴的最新提交从 origin/dev 抓下来，然后在本地合并、解决冲突【每次工作前也是需要pull远程的master 到本地的master，才继续工作，“先在本地pull、合并分支、在推送到相应的远程分支上“？！！】。

3.8 此时 git pull也报错了【因为之前，本地的 dev分支没有设置它的“远程上游分支”，不可以直接 git pull 拉取？？】

```
$ git pull
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details.

    git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

    git branch --set-upstream-to=origin/<branch> dev
```

根据最后一行的提示去解决即可【git pull 失败。因为没指定 本地 dev分支 与 远程 origin/dev分支 的链接！！ “即没有将 dev 的本地、远程建立关联、映射吧？！！“】：
git branch --set-upstream-to=origin/dev dev

```
$ git branch --set-upstream-to=origin/dev dev
Branch 'dev' set up to track remote branch 'dev' from 'origin'.
```

3.9 最后进行 git pull、解决冲突、add、commit 即可

```
$ git pull
Auto-merging env.txt
CONFLICT (add/add): Merge conflict in env.txt
Automatic merge failed; fix conflicts and then commit the result.
```

这回`git pull`成功，但是合并有冲突，需要手动解决，解决的方法和分支管理中的解决冲突完全一样。解决后，提交，再push：

```
$ git commit -m "fix env conflict"
[dev 57c53ab] fix env conflict

$ git push origin dev
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 621 bytes | 621.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To github.com:michaelliao/learngit.git
   7a5e5dd..57c53ab  dev -> dev
```

4 因此，多人协作的工作模式通常是这样：
4.1 首先可以是图用 git push origin branch_name【如果建立 链接、关联映射，就可以简写成 git push 吧？！！】推送自己的修改

4.2 若推送失败，则可能远程分支比你的本地更 新 。需要先使用 pull 拉取最新的内容。

4.3 pull完了、若有合并冲突，手动去解决冲突、并在本地进行提交。

4.4 没有冲突或者解决掉冲突后，再用 git push origin branch_name 推送即可！！

注意：

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-to <branch-name> origin/<branch-name>`。

5 小结
5.1 查看详细的远程库信息【git remote -v】。
查看远程仓库名字【git remote】

5.2 本地新建的分支如果不推送到 远程相应的分支上，别人是看不见的！！

5.3 从本地向远程推送分支，git push origin branch_name，若失败、应先用 git pull 抓取远程的新提交！！！

5.4 在本地创建 和 远程分支对应的分支【尽量2者的名字保持一致！！】 git checkout -b branch_name origin/branch_name 【？？这个与 git branch --set-upstream-to branch_name origin/branch_name 有啥区别？？】

5.5 建立本地分支 和 远程分支的 关联、映射。git branch --set-upstream-to brnach_name

origin/branch_name。

5.6 若pull最新内容下来有冲突，先手动解决冲突，再 add、commit、push 到远程！！！

二 rebase【变基操作】
1 上一节中，我们可以看到，多个人在同一个分支上协作时，很容易产生冲突。
即使无冲突，后 push的同学 需要先 pull，在本地合并，然后再能 push成功！！
每次合并再 push，分支就变成这样【显得很“杂乱丛生”】：
下面的图怎么看、技巧，优先找 * 号所在的分支？？

```
$ git log --graph --pretty=oneline --abbrev-commit
* d1be385 (HEAD -> master, origin/master) init hello
*   e5e69f1 Merge branch 'dev'
|\
| *   57c53ab (origin/dev, dev) fix env conflict
| |\
| | * 7a5e5dd add env
| * | 7bd91f1 add new env
| |/
* |   12a631b merged bug fix 101
|\ \
| * | 4c805e2 fix bug 101
|/ /
* |   e1e9c68 merge with no-ff
|\ \
| |/
| * f52c633 add merge
|/
*   cf810e4 conflict fixed
```

注意：总之，现在分支看起来很杂乱，为啥git的提交历史就不能是一条干净、笔直 的直线呢？？ ---> 可以， rebase 变基操作【把分叉的提交变成直线！！】

2 变基的实验操作
2.1 和远程分支进行同步后，我们对 hello.py 这个文件进行了2次提交。
用 git log 查看提交历史情况：

```
$ git log --graph --pretty=oneline --abbrev-commit
* 582d922 (HEAD -> master) add author
* 8875536 add comment
* d1be385 (origin/master) init hello
*   e5e69f1 Merge branch 'dev'
|\
| *   57c53ab (origin/dev, dev) fix env conflict
| |\
| | * 7a5e5dd add env
| * | 7bd91f1 add new env
...
```

注意：git用 HEAD->master 和 origin/master 表示当前分支的HEAD 和 远程的origin 的位置分别为 582d922 add author 、 d1be385 init hello。
本地分支 比 远程 快了2个提交！！！

2.2 我们尝试 推送本地分支【会失败，因为此前有人 比我们先推送了远程分支！！】

```
$ git push origin master
To github.com:michaelliao/learngit.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:michaelliao/learngit.git'
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.
```

2.3 我们先 pull 让本地同步一下远程的，再用 git status 查看当前状态：

```
$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:michaelliao/learngit
   d1be385..f005ed4  master     -> origin/master
 * [new tag]         v1.0       -> v1.0
Auto-merging hello.py
Merge made by the 'recursive' strategy.
 hello.py | 1 +
 1 file changed, 1 insertion(+)
```

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

注意：第3行的 Your branch is ahead of 'origin/master' by 3 commit【本地该分支 比 远程多了 3个提交！！】。
说明了，加上刚才合并的提交，现在我们本地分支 比 远程分支超前了3个提交！！

2.4 git log会发现 提交历史分叉【如果现在把本地分支 push到远程，会显得不好看，此时我们就需要 rebase变基 操作了！】了！！

```
$ git log --graph --pretty=oneline --abbrev-commit
*   e0ea545 (HEAD -> master) Merge branch 'master' of github.com:michaelliao/learngit
|\
| * f005ed4 (origin/master) set exit=1
* | 582d922 add author
* | 8875536 add comment
|/
* d1be385 init hello
...
```

2.5 输入 git rebase ， git 会自动执行一系列操作、帮我们完成变基的更改！

```
$ git rebase
First, rewinding head to replay your work on top of it...
Applying: add comment
Using index info to reconstruct a base tree...
M   hello.py
Falling back to patching base and 3-way merge...
Auto-merging hello.py
Applying: add author
Using index info to reconstruct a base tree...
M   hello.py
Falling back to patching base and 3-way merge...
Auto-merging hello.py
```

2.6 git rebase之后，用 git log【如 git log --graph --pretty=oneline --abbrev-commit】 查看当前的历史提交

```
$ git log --graph --pretty=oneline --abbrev-commit
* 7e61ed4 (HEAD -> master) add author
* 3611cfe add comment
* f005ed4 (origin/master) set exit=1
* d1be385 init hello
...
```

rebase变基原理：我们发现 git 把我们本地的提交“挪动了”位置，放到了 f005ed4(origin/master) set exit=1 之后。
这样，整个提交就是一条直线了。rebase操作前后，最终的提交内容是一致的，有相同【不管有没有变基，只要提交内容一致！！】的 commit_id。
但是我们本地的 commit 修改内容已经发生了变化。他们不再基于 d1be385 init hello，而是基于 f005ed4(origin/master) set exit=1 【为啥一定要基于这个？？】，但是最后提交 7e61ed4 内容是一样的！！

2.7 将最后变好基的 本地分支 推送到 远程：

```
Mac:~/learngit michael$ git push origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 576 bytes | 576.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:michaelliao/learngit.git
   f005ed4..7e61ed4  master -> master
```

git log 查看：

```
$ git log --graph --pretty=oneline --abbrev-commit
* 7e61ed4 (HEAD -> master, origin/master) add author
* 3611cfe add comment
* f005ed4 set exit=1
* d1be385 init hello
...
```

会发现：
远程分支的提交分支已经是一条直线了，但是注意 本地的分叉提交 已经被修改过了【不好“理清背锅人？？”】

3 小结
3.1 rebase【操作对象是 本地未push的分叉提交！！】操作可以把本地未push的分叉提交整理成直线。

3.2 rebase的目的是使得我们在查看历史提交的变化时更容易【一条直线比较直观、易懂】，因为分叉的提交需要三方对比【啥叫三方对比？？！】。

完