**By Brother.Lee 201800507 Shanghai**

Git教程

# Git系列视频教程-1-教你正确的使用git config

## config配置

没错，在我们使用git之前，首要做的事就是使用config命令进行配置。

git的安装这里就不讲了，去官网下载对应操作系统的安装文件，基本按照默认方式安装即可。

为什么配置git是很重要的一步呢？这里列举出三点。

1.最主要的安全和隐私问题。

2.设置别名，提高工作效率。

3.设置默认选项。

λ git --version

git version 2.15.0.windows.1

λ git config --list

core.symlinks=false

core.autocrlf=true

core.fscache=true

color.diff=auto

color.status=auto

color.branch=auto

color.interactive=true

help.format=html

http.sslcainfo=G:/LeeQueen/Install/Git/mingw32/ssl/certs/ca-bundle.crt

diff.astextplain.textconv=astextplain

rebase.autosquash=true

core.symlinks=false

core.autocrlf=true

color.diff=auto

color.status=auto

color.branch=auto

color.interactive=true

pack.packsizelimit=2g

help.format=html

http.sslcainfo=/ssl/certs/ca-bundle.crt

diff.astextplain.textconv=astextplain

rebase.autosquash=true

filter.lfs.clean=git-lfs clean -- %f

filter.lfs.smudge=git-lfs smudge -- %f

filter.lfs.process=git-lfs filter-process

filter.lfs.required=true

credential.helper=manager

filter.lfs.clean=git-lfs clean -- %f

filter.lfs.smudge=git-lfs smudge -- %f

filter.lfs.process=git-lfs filter-process

filter.lfs.required=true

## 用户名和邮箱的设置:

1. 全局—global

--global表示的是全局配置，即不管是在哪个项目仓库下，这个配置都是生效的。

一般在正常情况下，我们首要设置的是用户名和邮箱，将个人使用的用户名和邮箱设置为全局的，而将公司使用的用户名和邮箱设置为内部项目下的。

这里我们看看全局的设置。

git config --global user.name ‘1iqueen’

git config --global user.email ‘1018661791@qq.com’

(二)局部--local

既然有--global全局配置，那就有局部配置。在默认的情况下如果不加--global选项就是局部配置，当然也可以额外使用--local选项。

## 如何创建本地git仓库?

## 新建一目录:



## 代码仓库的创建

λ git init

Initialized empty Git repository in C:/Users/ray/Desktop/TestForGit/.git/

进入文件夹,可以查看隐藏文件

C:\Users\ray\Desktop\TestForGit (master -> origin)

λ ls -al

total 44

drwxr-xr-x 1 ray 197121 0 May 7 15:59 ./

drwxr-xr-x 1 ray 197121 0 May 7 15:57 ../

drwxr-xr-x 1 ray 197121 0 May 7 15:59 .git/

也可以打开工程目录，同样看也看到.git文件夹；**如果我们想删除代码仓库只需把这个文件夹删掉即可！**

提交本地代码

λ git add readme.txt

λ git commit -m "Wrote a readme file"

[master (root-commit) 7bbcaed] Wrote a readme file

1 file changed, 1 insertion(+)

create mode 100644 readme.txt

## 查看Git提交记录git log

λ git log

commit 7bbcaed2dd5d0a99bd54e528e0869d10332f3e70 (HEAD -> master)

Author: likun <<E2><80><98>1018661791@qq.com<E2><80><99>>

Date: Mon May 7 16:14:01 2018 +0800

Wrote a readme file

## alias

alias主要是用于设置别名的，像git一些常用的指令都是英文单词，对于那些已经习惯别名的人来说，他们会觉得这是一件很费时的事情，因此对于alias的设置也是必须的。

1.git status是用来查看文件修改的命令，可以使用git st来代替。

2.git pull是用来远程拉取仓库代码的命令，可以使用git pl来代替。

3.git ps是用来将本地修改推送至远程仓库的命令，可以使用git ps来代替。

## git add -A和 git add .   git add -u在功能上看似很相近，但还是存在一点差别

git add . ：他会监控工作区的状态树，使用它会把工作时的**所有变化提交**到暂存区，包括文件内容修改(modified)以及新文件(new)，但不包括被删除的文件。

git add -u ：他仅监控**已经被add的文件**（即tracked file），他会将被修改的文件提交到暂存区。add -u 不会提交新文件（untracked file）。（git add --update的缩写）

git add -A ：是上面两个功能的合集（git add --all的缩写）

## git commit命令是做什么用的?

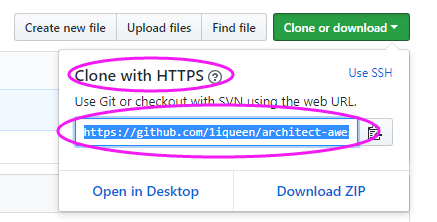
git是分为三部分，一部分是你自己的文件，另外一个是缓存区，最后一个是本地库。当你修改了自己的文件后，你会git add xx将修改保存到缓存区，然后再用commit推送修改到本地库中。

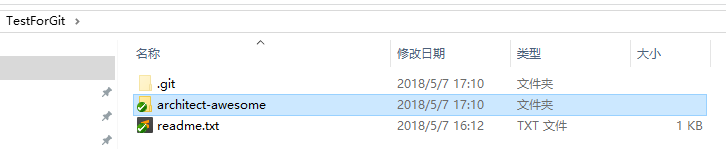
## config配置的级别

git读取时:优先从local> global> system级别

# Git系列视频教程-2-git clone和git tag的重点内容

## 从github上clone代码





λ git clone https://github.com/1iqueen/architect-awesome.git

Cloning into 'architect-awesome'...

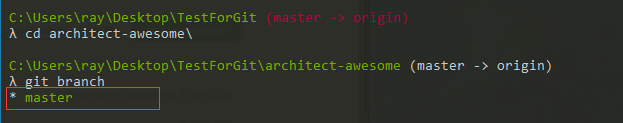
remote: Counting objects: 418, done.

remote: Total 418 (delta 0), reused 0 (delta 0), pack-reused 418

Receiving objects: 100% (418/418), 162.52 KiB | 76.00 KiB/s, done.

Resolving deltas: 100% (137/137), done.

## git branch显示分支



## git tag显示标签(版本号)



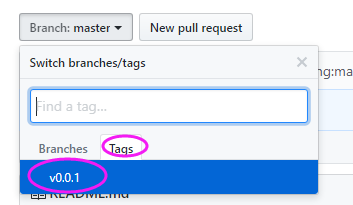
打标签

git tag –a v0.1 –m ‘v0.1’

并推送到仓库

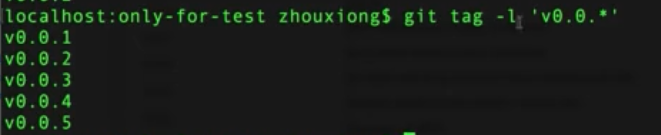
git push –-tag





## git –tag查找过滤

git tag –l ‘v.0.0.\*’



## 补充标签

$ git log --pretty=oneline

0b7434e add a function

4682c3a add a file

166ae0c write support

9fceb02 updated rakefile

964f16d commit the todo

8a5cbc4 updated readme

假如我们需要在964f16d提交记录上打一次标签。

可以通过运行以下这条命令。

$ git tag -a v1.2 964f16d

# Git系列视频教程-3-文件状态之git status与git diff

## 文件状态

在一个本地仓库中，所有文件都不外乎两种状态：已追踪和未追踪。

已追踪文件

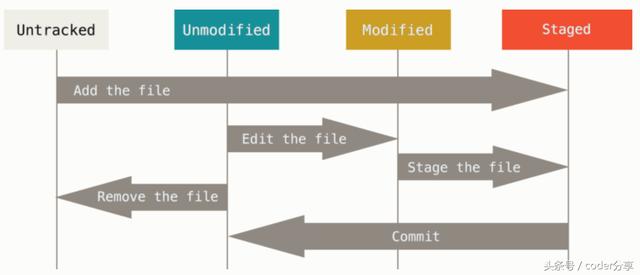
已追踪文件表示的是在仓库之前的版本快照中包含了文件的记录，在用户工作一段时间后，这些文件同样能够被追踪到，要么是修改文件，要么是删除文件，又或者是没做任何更改。

未追踪文件

除了已追踪文件外，其他文件都是未追踪文件，它们并不在上一次版本快照中，也不在本地暂存区内，这种类型的文件一般都是新增的文件。

有一点需要记住的是：**刚clone完的项目，所有文件都是已追踪状态，并且都未修改。**

在本地仓库中，文件状态的变更如下图所示。



## git status

通过git status命令显示出来的文件总共有三种状态,分别是

1:已添加至暂存区,待提交(git add后)

2:已修改,未暂存

3:未追踪状态的文件

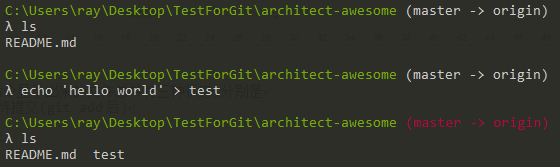
λ git status

On branch master

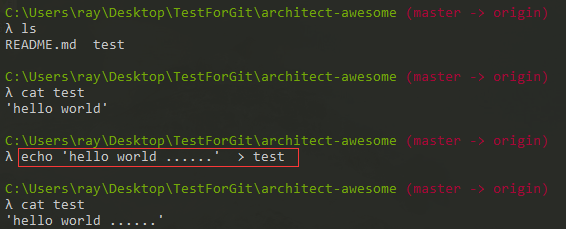
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

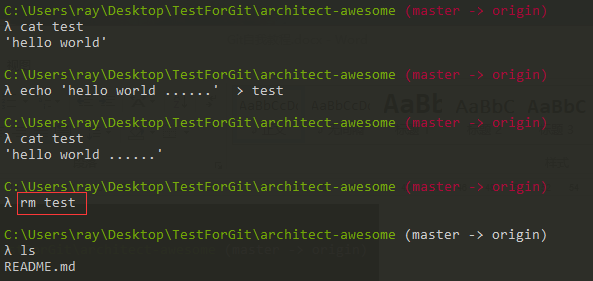
新增一个文件:



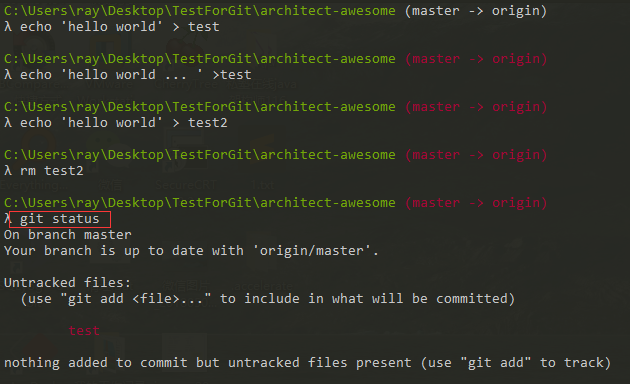
修改一个文件:



删除一个文件:



查看状态:



## 忽略文件

在一个仓库中，有很多的文件其实是不用提交至远程仓库中的，像前端项目中npm安装的node\_modules文件夹，还有Java项目的Maven Dependencies和.class文件。

这个时候.gitignore文件就派上用场了，在文件中列出需要忽略的文件模式。

.gitignore文件格式如下：

# 表示注释会被忽略。

空行直接被忽略。

可以用正则匹配。

除去匹配的文件，在表达式前面加上取反符号(!)。

下面我们看一个标准的.gitignore文件

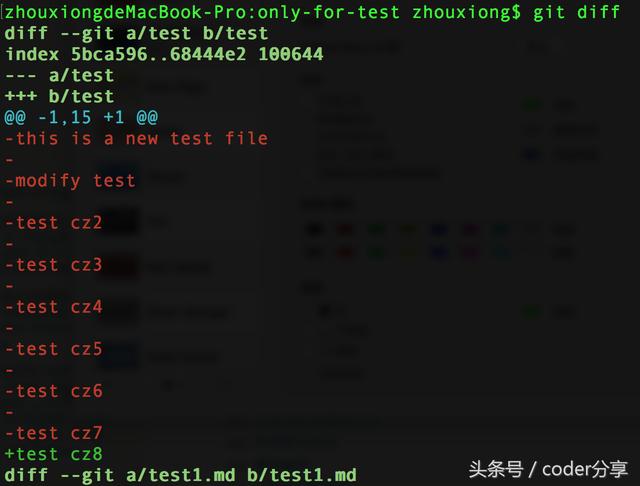


## 查看修改git diff

当我们需要查看文件具体修改的地方时，可以运行git diff命令。

查看尚未暂存文件的修改

在默认情况下，我们运行git diff命令查看的是尚未暂存的文件所有修改的地方。

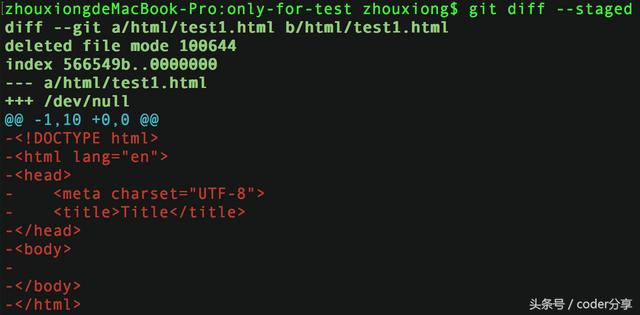


需要记住的一点是：**git diff显示的是修改后未暂存起来的文件内容，如果将所有修改的文件已经添加至暂存区再运行git diff命令，这时会发现什么内容都没有。**

查看暂存区修改

如果需要查看的是暂存区的修改，需要添加--staged选项。

从下面的图可以看出，我们删除的文件html/test1.html内容显示了出来



## 提交记录git commit

在通过git add命令将修改的文件添加至暂存区后，就该使用git commit提交修改了。

一般都会通过-m参数补充本次提交的内容描述，这个是很重要的，尤其是多人协同开发时，可以让别的开发人员大致了解你提交的目的。



关于-m后的信息描述是一件需要重点说明的事，我会在下一篇文章中讲述

# Git系列视频教程-4-如何写出一个完美的commit message？

## 什么是commit message?

git commit –m ‘每次提交时在这个地方编辑的内容’

## commit message的好处

提供更多可查询的信息,用于排查问题

过滤重要的内容

生成changelog

## commit message

标准的commit message包括Header,Body,Fotter三个部分

只使用Header部分

## Header

type(scope):subject

type有以下几种常用的类型:

feat:新功能

fix:修复bug

style:格式

refactor:代码重构

chore:项目构建

## git cz

全局安装commitizen

设置支持angular格式的Commit message

commitizen init cz-conventional-changelog --save-exact

全局安装cz(在linux系统中)

npm install –g commitizen

## changelog

changelog是项目迭代过程中一系列的变更记录

全局安装changelog

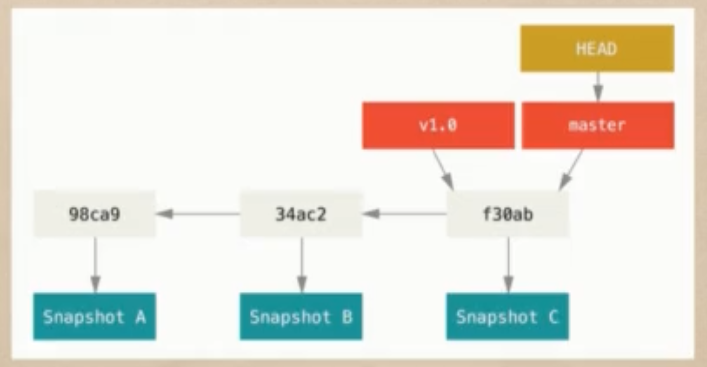
npm install –g conventional-changelog

生成所有的changelog

conventional-changelog –p angular –i CHANGELOG.md –s –r O

# Git系列视频教程-5-git分支管理

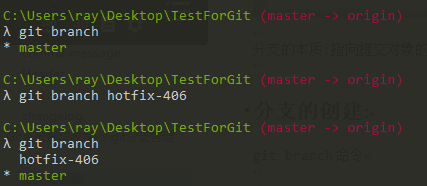
## 分支



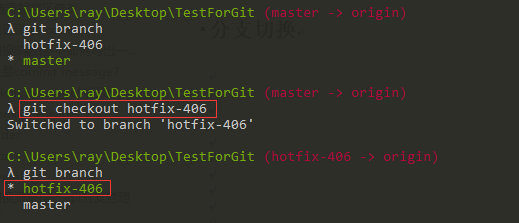
分支的本质:指向提交对象的一个可变指针.

## 分支的创建:

git branch命令

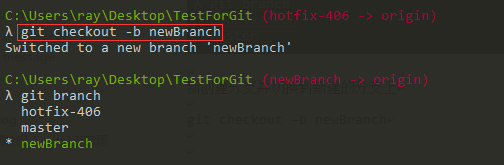


## 分支切换



## 新创建分支并切换到新建的分支上

git checkout –b newBranch



## 分支删除

git brach –d test



## 分支合并

git merge

(当前在self分支上)

1:修改代码

2:添加代码到暂存区git add -A

3:提交代码git commit

4:切换(到dev)分支 git checkout dev

5:合并分支git merge self

6:将dev的代码push到远程(github)仓库git push origin dev

## 解决冲突

merge的时候会产生冲突,解决完后再通过git add,commit等操作;

# Git系列视频教程-6-git remote远程仓库

## 远程分支

远程分支类似于本地分支,是指向远程仓库中文件的指针

实际开发中,每一个本地分支都应该存在一个域远程分支的对应关系;

## 开发的基本步骤:

1:git pull origin dev

2:修改文件

3:git add –A

4:git commit

5:git push origin dev

## 拉取远程分支的内容:

git fetch origin dev

## 远程抓取

git fetch不会对本地仓库内容进行更新

git merge真正的更新本地仓库的内容

git pull = git fetch + git merge

## 远程推送

git push

git push分支,推送内容至仓库

git push –d,删除分支

git push --tag 推送tag

## 打标签并推送到远程

git tag –a v0.0.1 0 –m ‘v0.0.7’

git push --tag

## git remote

git remote –v

git remote add

git remote renmove

# Git系列视频教程-7-git merge与git rebase

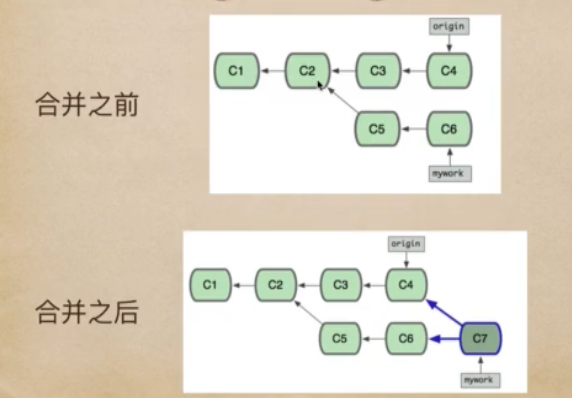
## 分支合并

git merge和git rebase都可以进行分支合并

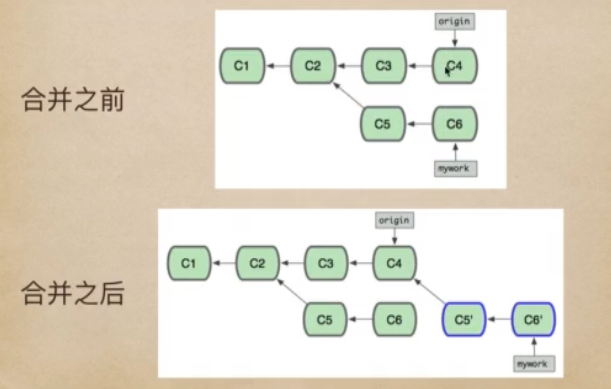
git merge合并后保留两个分支的记录

git rebase合并后会展示成一个分支的记录,另一个分支的提交实际生成了一个副本.

## git merge



## git rebase



## 冲突解决

git merge在解决冲突后需要手动通过git add,git commit生成一次提交;

git rebase在解决冲突后,会自动生成一次提交.

git merge一次性会解决之前提交记录中的所有冲突

git rebase一次只会解决一次提交上的冲突,如果多次提交都有冲突,需要执行多次rebase操作;

# Git系列视频教程-8-git撤销和回滚操作

## git撤销

未添加进暂存区的撤销

添加进暂存区的撤销

已经提交了,想要撤销某次提交

git revert

git checkout --filename

将没有添加至暂存区的文件回溯到你所在的分支最近的一次提交

git reset HEAD filename

已经添加至暂存区文件的撤销

## git回滚

回退到某次提交,该提交以后的提交都回退

git reset –hard commitId

# Git系列视频教程-9-git常用命令实战

## 本地项目创建

1:创建项目

2:git配置

3:初始化成git项目

4:添加.gitignore文件

5:添加ReadME文件

## 正常开发联调

1:建立个人分支

2:文件新增,编辑,删除

3:状态查看git status

4:添加git add

5:提交git commit

6:推送git push

## hotfix修复问题

切换新的分支git checkout –b

问题修复

代码提交

代码合并

重新打tag

## 多人协作开发

远端会出现多个不同的开发分支

拉取远程公共分支

合并到本地分支

解决冲突

提交

推送