# Git配置

Git 提供了一个叫做 git config 的工具，专门用来配置或读取相应的工作环境变量。

这些环境变量，决定了 Git 在各个环节的具体工作方式和行为。这些变量可以存放在以下三个不同的地方：

（1）/etc/gitconfig 文件：系统中对所有用户都普遍适用的配置。若使用 git config 时用 --system 选项，读写的就是这个文件。

（2）~/.gitconfig 文件：用户目录下的配置文件只适用于该用户。若使用 git config 时用 --global 选项，读写的就是这个文件。

（3）当前项目的 Git 目录中的配置文件（也就是工作目录中的 .git/config 文件）：这里的配置仅仅针对当前项目有效。每一个级别的配置都会覆盖上层的相同配置，所以 .git/config 里的配置会覆盖 /etc/gitconfig 中的同名变量。

# 用户信息

git config --global user.name "runoob"

git config --global user.email test@runoob.com

# 差异分析工具

git config --global merge.tool vimdiff

# 查看配置信息

git config --list

# Git 工作流程



# Git 工作区、暂存区和版本库

1. 工作区：就是你在电脑里能看到的目录。
2. 暂存区：英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。
3. 版本库：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

# Git 创建仓库

初始化（当前目录）：

git init

初始化（指定目录）：

git init newrepo

初始化后会出现一个名为.git的目录，所有的Git需要的数据和资源都放到这个目录当中。

如果当前目录下有几个文件想要纳入版本控制，需要先用git add命令告诉git开始对这些文件进行跟踪然后提交。

$ git add \*.c

$ git add README

$ git commit -m '初始化项目版本'

以上命令将目录下以 .c 结尾及 README 文件提交到仓库中。

克隆仓库的命令格式为：

git clone <repo>

如果我们需要克隆到指定的目录，可以使用以下命令格式：

git clone <repo> <directory>

其中repo指的是Git仓库。

git clone 时，可以所用不同的协议，包括 ssh, git, https 等，其中最常用的是 ssh，因为速度较快，还可以配置公钥免输入密码。各种写法如下：

git clone git@github.com:fsliurujie/test.git --SSH协议

git clone git://github.com/fsliurujie/test.git --GIT协议

git clone https://github.com/fsliurujie/test.git --HTTPS协议

# git add

（git add 命令可将该文件添加到缓存）

新项目中，添加所有文件很普遍，我们可以使用 git add . 命令来添加当前项目的所有文件。

# git status

显示你上次提交更新后的更改或者写入缓存的改动， 而 git diff 一行一行地显示这些改动具体是啥。

# git diff

git diff (filename)

git diff 告诉我们当前文件（filename）与上一次commit的差异

git diff HEAD -- readme.txt

用git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别：

# git commit

使用 git add 命令将想要快照的内容写入缓存区， 而执行 git commit 将缓存区内容添加到仓库中。

git reset HEAD

git reset HEAD 命令用于取消已缓存的内容。

# 分支管理

## 创建分支命令：

git branch (branchname)

## 切换分支命令:

git checkout (branchname)

git checkout -- readme.txt可以丢弃工作区的修改

git checkout -- readme.txt

## 列出分支基本命令：

git branch

## 删除分支：

1.删除本地分支

git branch -d (branchname)

2.删除远程分支

git push origin –delete (branchname)

## 分支合并

一旦某分支有了独立内容，你终究会希望将它合并回到你的主分支。 你可以使用以下命令将任何分支合并到当前分支中去：

git merge

# Git 查看提交历史

$ git log

查看精简信息

$ git log --pretty=oneline

# git push origin与git push -u origin master的区别

$ git push origin

上面命令表示，将当前分支推送到origin主机的对应分支。

如果当前分支只有一个追踪分支，那么主机名都可以省略。

$ git push

如果当前分支与多个主机存在追踪关系，那么这个时候-u选项会指定一个默认主机，这样后面就可以不加任何参数使用git push。

$ git push -u origin master

上面命令将本地的master分支推送到origin主机，同时指定origin为默认主机，后面就可以不加任何参数使用git push了。

不带任何参数的git push，默认只推送当前分支，这叫做simple方式。此外，还有一种matching方式，会推送所有有对应的远程分支的本地分支。Git 2.0版本之前，默认采用matching方法，现在改为默认采用simple方式。

# Git master branch has no upstream branch的解决

## 具体原因：

出现这种情况主要是由于远程仓库太多，且分支较多。在默认情况下，git push时一般会上传到origin下的master分支上，然而当repository和branch过多，而又没有设置关联时，git就会产生疑问，因为它无法判断你的push目标。

## Master和origin

Git 的 “master” 分支并不是一个特殊分支。 它就跟其它分支完全没有区别。 之所以几乎每一个仓库都有 master 分支，是因为git init命令默认创建它，并且大多数人都懒得去改动它。

远程仓库名字 “origin” 与分支名字 “master” 一样，在 Git 中并没有任何特别的含义一样。origin” 是当你运行git clone时默认的远程仓库名字。 如果你运行 git clone -o booyah，那么你默认的远程分支名字将会是 booyah/master。

## 解决办法

1：$ git push --set-upstream origin new\_branch。其中的origin是你在clone远程代码时，git为你创建的指向这个远程代码库的标签，它指向repository。为了能清楚了解你要指向的repository，可以用命令git remote -v进行查看。master是你远程的branch，可以用git branch -a查看所有分支，远程分支是红色的部分。然后确定好这两个值后，将值换掉即可。

2：git push -u origin master。同样根据自己的需要，替换origin和master

# 记Git报错-refusing to merge unrelated histories

$git pull origin master –allow-unrelated-histories

# Git push到相应的远程分支

$ git push <远程主机名> <本地分支名>:<远程分支名>

# 基本知识点

## 1. git commit、git push、git pull、 git fetch、git merge 的含义与区别

git commit：是将本地修改过的文件提交到本地库中；

git push：是将本地库中的最新信息发送给远程库；

git pull：是从远程获取最新版本到本地，并自动merge；

git fetch：是从远程获取最新版本到本地，不会自动merge；

git merge：是用于从指定的commit(s)合并到当前分支，用来合并两个分支；

git pull 相当于 git fetch + git merge。

## 2.版本回退

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

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。

## 3.gitcheck

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”。