# 10minutesToGit

## 一章 初识git

### 1.1 git 是什么

git 是一种版本控制器 .

更直白说 , 团队开发时 , 管理代码用的软件 .

面试时 , 容易被问到的一个东西 .

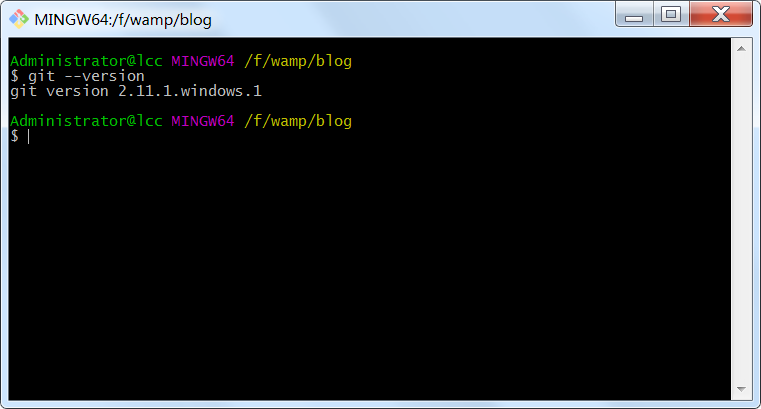
### 1.2 安装

git 在 Linux,Mac,Win 下都可以安装 .

本文是以 Win7 系统为环境编写的 .

Window 环境 :

到 https://git-for-windows.github.io/ 下载软件 , 双击 , 一路 "Next", 安装完毕 .



到开始菜单找 "git bash", 如下图

看到如下界面 :

Linux 环境安装 git:

# ubuntu,debian#

$ sudo apt-get install git

centos,redhat 系统

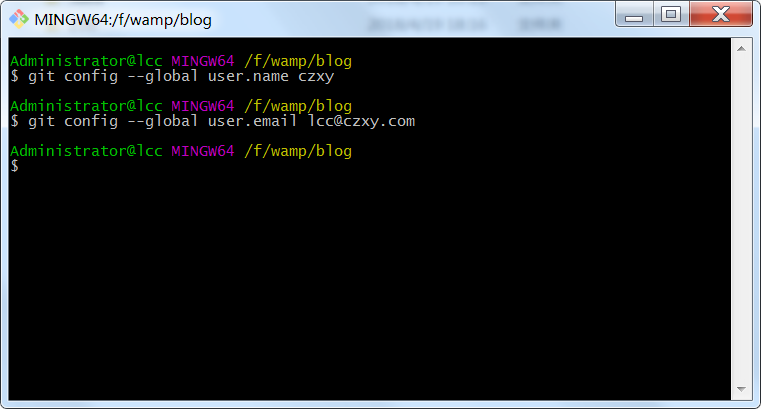
# yum install git

### 1.3 设置账号

在你用 git 之前 , 要先设置用户名, 否则代码不能提交 .

$ git config --global user.name # 你是谁

$ git config --global user.email # 怎么联系你



### 1.4 代码管理

#### 1.4.1 创建版本库

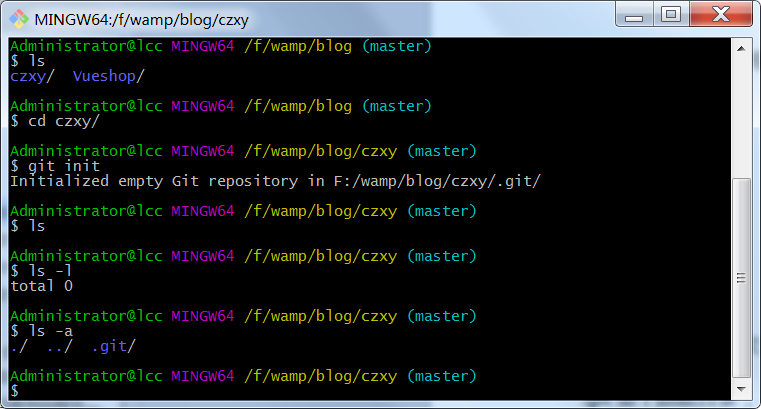
$ mkdir czxy

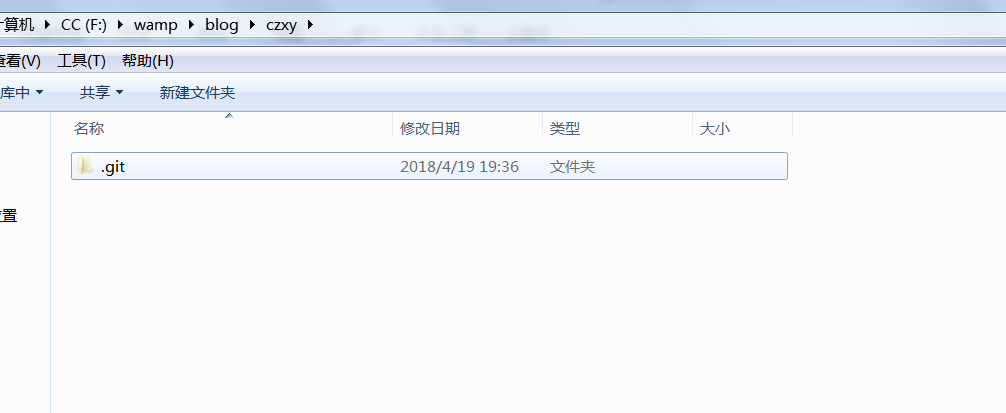
$ git init

注意 :

不要把仓库建在中文目录下 , 可能出问题 .

.git是个隐藏目录 , 不要乱碰 .( 你的每一次代码修改它都帮你记录着呢 )





#### 1.4.2 添加文件

开发你的程序 . 比如 , index.php

echo 'hello git';

编辑 PHP 文件后 , # git status , 查看仓库状态

实例如下 :

$ git status

可见 , 此时 git 发现有一个新文件 , 但并没有把此文件纳入管理 .

我们需要两步 , 让 git 仓库管理 index.php

git add index.php

把index.php提交到 暂存区

git commit -m " 新建 index.php"

把index.php提交到 版本库

实例如下 :

$ git add index.php

$ git commit -m " 新建 index.php"

#### 1.4.3 修改文件

如果修改了文件 , 也不要忘记提交到版本库

这个过程 和添加文件是一样的

一样是需要两步 , 让 git 仓库记录此次改变

git add index.php

把index.php提交到 暂存区

git commit -m " 改了第 2 行 "

把index.php提交到 版本库

实例 :

$ git add index.php

$ git commit -m " 改了第 2 行 "

#### 1.4.4 删除文件

用 rm 命令删除文件 , 并直接 commit, 提交到版本库

例 : 先创建一个 foo.php, 供练习删除用

实例如下 :

$ touch foo.php # 创建 foo.php

$ git add foo.php

$ git commit -m " 练习删除用 "

$ ls

foo.php index.php

# 开始删除

$ git rm foo.php

rm 'foo.php'

$ git commit -m " 删除 foo.php"

[master e4dc37c] 删除 foo.php

1 file changed, 0 insertions(+), 0 deletions(-)

delete mode 100644 foo.php

$ ls

index.php

### 1.5 远程仓库

经过前面的练习 , 你在本地的仓库里管理代码已经比较熟练了 .

但如果是团队开发 , 如何配合起来呢 ?

我们可以把版本仓库放在互联网上 .

开发者把自己最新的版本推到线上仓库 ,

同时 , 把线上仓库的最新代码 , 拉到自己本地 .

这样 , 就可以配合工作了 .

#### 1.5.1 注册 git 在线仓库的账号

国外 : http://www.github.com

国内 : http://git.oschina.net

github.com 也是目前程序员的装逼利器 , 在 github 上挂个项目 , 逼格瞬间提升两三档 .

不过由于是国外网站 , 速度不咋样 .

出于学习的方便 , 我在 oschina.net 给大家演示 .

请立即打开 http://git.oschina.net 注册一个账户 .

#### 1.5.2 创建项目

在 oschina 注册后 ," 新建项目 ", 我们先建一个测试项目 , 叫 test.

如下 :

oschina 为此项目提供的仓库地址有 2 个 .

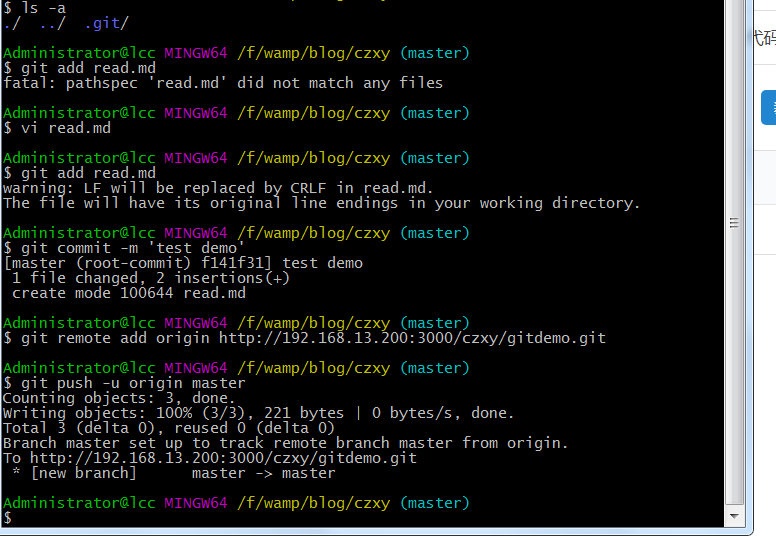
http 地址 : https://git.oschina.net/lianshou/test.git

ssh 地址 : git@git.oschina.net:lianshou/test.git

#### 1.5.3 把代码推到远程仓库

推 : push

为本地库添加远程库

意思是 : 添加 1 个远程库 , 代号是 origin, 地址是 https://....test.git

push 推代码

push origin master

意思是 , 把本地的版本 ( 默认是 master), 推到代号为 origin 的远程库去 .

这个过程会让你输入用户名 / 密码 , 即你注册时的账户密码 .

## 二章 git 的特点及诞生

### 2.1 分布式版本控制器

何为分布式 ? 与集中式相比有何特点 ?

以 SVN 为例 :

中心的 svn 服务器中 , 存储着代码版本的变迁 , 及日志 .

你想查看改动日志 , 请联网 SVN 服务器 .

你想退回上个版本 , 请联网 SVN 服务器 .

你想创建新的分支 , 请联网 SVN 服务器 .

联网不说 , 万一 SVN 服务器要是坏了 ??? 后果你说呢 .

而 git 是这样的 :

每个开发者的电脑上 , 都有完整的版本 , 日志 , 及分支信息 .

但开发者不依赖于服务器 , 可以查看日志 , 回退版本 , 创建分支 .

当然 , 世界各地的开发需要交换最新的版本信息 ,

因此 ,git 往往也需要服务器 .

但是 , 本质的区别在于 :

git 服务器是供开发者 " 交换 " 代码 , 服务器数据丢了没关系 , 分分钟再建一台 .

svn 的服务器 , 不仅交换代码 , 还控制着日志 , 版本 , 分支 . 服务器数据丢了就完了 .

### 2.2 发展历史

Linux 之父 Linus Torvalds 在 1991 年创建了 linux 开源项目 , 并把项目放在互联网上 , 引来世界大量的黑客 , 大神为项目贡献代码 .

问题是 , 这么多的人同时贡献代码 , 如何管理代码成了一件头疼的事 .

随着 linux 内核的管理工作越来越吃力 ,linus 选择了一款商业版本控制器 -BitKeeper.

BitKeeper 是 BitMover 公司旗下的产品 .

公司的老大 Larry 也希望借机扩大产品的影响力 , 因此授权 Linux 社区免费使用 BitKeeper.

这件事 , 在开源圈引起了不小的骚动 .

因为 ,BitKeeper 只是 free( 免费 ), 而非 free( 自由 ).

开源教主 RMS 为这事儿还说过 linus.

2002 年 2 月 ,Linus 开始用它来管理 Linux 内核代码主线 ,Linus 对 BitKeeper 的评价是 the best tool for the job.

确实 , 自从 Linus 使用 BitKeeper 之后 ,Linux 的开发步伐加快了两倍 .

可惜的是 , 就像黑帮电影中 , 老大蒸蒸日上的事业 , 往往坏在一个不懂事的小弟手中 .

这帮视 free( 自由 ) 如信仰的牛人中 , 一个叫 Andrew 的 , 试图破解 BitKeeper 的协议 , 且被 BitMover 公司警告几次后仍不停手 .

最终 , 出事了 !

Linus 在 Andrew 和 Larry 两人间费力调停 , 但没有成功 .

既如此 ,Linus 说 :" 我的兄弟只是做错事不是做坏事 . 我扛 !"

于是 ,10 天后 ,git 诞生了 !

## 三章 代码管理

### 3.1 工作区和版本库

如果你想更清晰的学习 git, 你必须要了解 3 个重要区域 .

工作区 , 即开发者的工作目录 .

暂存区 , 修改已被记录 , 但尚未录入版本库的区域 .

版本库 , 存储变化日志及版本信息 .

做个实验 : 创建并提交一个文件的全过程

touch readme.txt 创建文件

$ git status 查看状态 , 如下 :

# On branch master

# Untracked files:

# (use "git add <file>..." to include in what will be committed)

#

# readme.txt

nothing added to commit but untracked files present (use "git add" to track)

意思是说 : 有未被跟踪的文件 readme.txt.

( 文件在你的工作区 , 还没告诉 git 去管理它 , 当然没被跟踪了 .)

你可以用git add <file> , 把它加入待提交列表

git add reame.txt 添加到暂存区

再次查看状态 , 如下 :

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: readme.txt

#

意思是说 :

代码的变化 , 可以提交 (commit) 了 .

也可以 git reset HEAD <file 把它从暂存区拿出来 .

git commit <file> -m < 注释 > 提交代码

$ git commit readme.txt -m "new readme.txt"

再次查看状态 , 如下 :

# On branch master

nothing to commit, working directory clean

可以看到 , 所有的变动 , 已录入版本库 . 目前工作区没有新内容 , 是干净的 .

### 3.2 文件操作

添加多个文件

git add <file1> <file2> # 添加 file1,file2

git add \*.txt # 添加当前目录下的 .txt 文档

git add . # 添加当前目录的所有变化

删除文件

git rm <file>

移动或改名

git mv 源文件 新文件

例

移动 : git mv config.php ./inc/config.php

改名 : git mv config.php config.inc.php

### 3.3 改动日志

每个文件 / 目录发生的版本变化 , 我们都可以追溯 .

命令为 :"git log "

常用格式 :

git log 查看项目的日志

git log <file> 查看某文件的日志

git log . 查看本目录的日志

例 :

### 3.4 版本切换

我们针对 ver.txt, 连续修改 4 次 , 形成 4 个版本 , 练习版本切换 .

one line

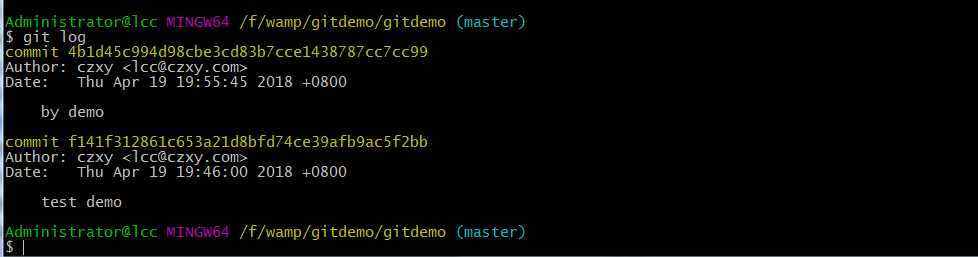
second line

third line

four line

git reflog 查看版本变化

$ git reflog

HEAD指向当前版本5d5df86 ,

切换为head的前 1 版本 , git reset --hard HEAD^

切换为head的前 2 版本 , git reset --hard HEAD^^

切换为head的前 100 版本 , git reset --hard HEAD~100

实例 :

$ git reset --hard HEAD^^

HEAD is now at 70110b9 two

此时 , 查看ver.txt的内容变为 :

one line

two line

也可以利用版本号来切换 , 例

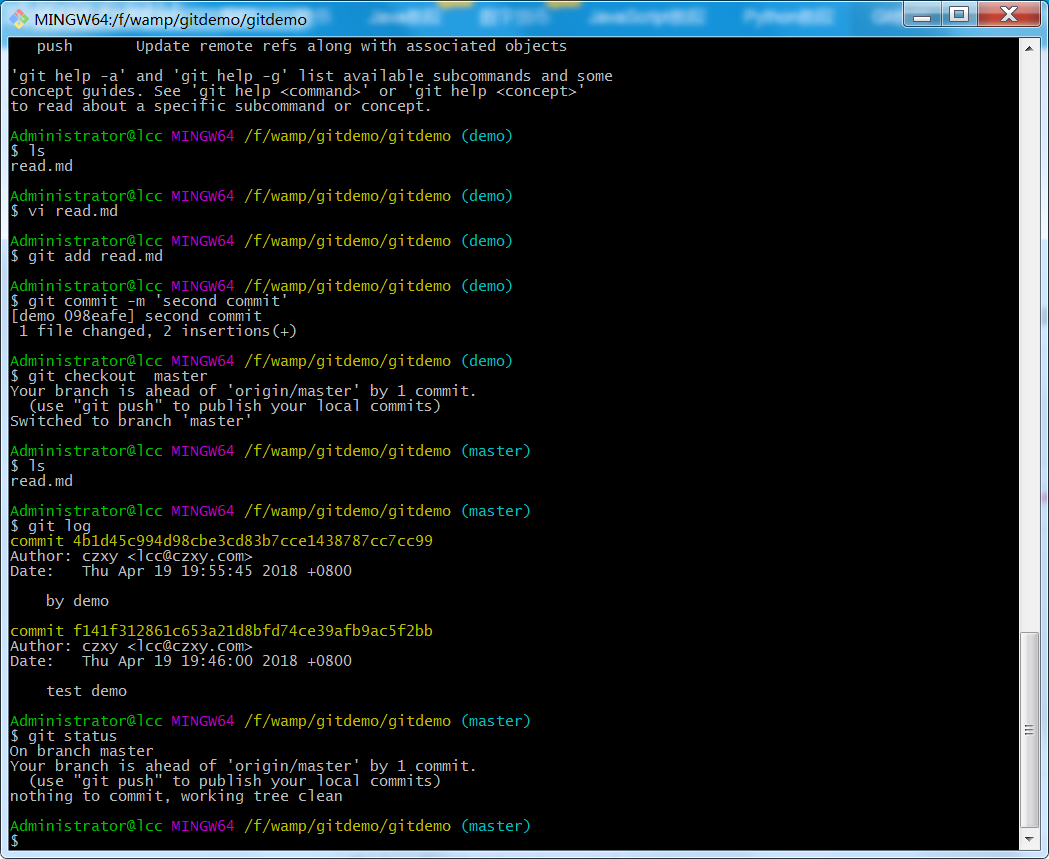
$ git reset --hard 6207e59

HEAD is now at 6207e59 three

注意 : 版本号不用写那么长 , 能要能保证不与其他版本号重复就行 .

例git reset --hard 6207

### 3.5版本冲突



### 3.6版本回退

git re**set** --hard 3628164

## 四章 分支管理1

### 4.1 分支有什么用 ?

在开发中 , 遇到这样的情况怎么办 ?

网站已有支付宝在线支付功能 , 要添加 " 微信支付 ".

修改了 3 个文件 , wechat.php , pay.php

刚做到一半 , 突然有个紧急 bug: 支付宝支付后不能修改订单状态 .

你需要立即马上修改这个 bug, 需要修改的文件是 , ali.php , pay.php .

问题是 : pay.php , 已经被你修改过 , 而且尚未完成 .

直接在此基础上改 , 肯定有问题 .

把pay.php倒回去 ? 那我之前的工作白费了 .

此时你肯定会想 : 在做 " 微信支付 " 时 , 能否把仓库复制一份 , 在此副本上修改 , 不影响原仓库的内容 . 修改完毕后 , 再把副本上的修改合并过去 .

好的 , 这时你已经有了分支的思想 .

前面见过的master , 即是代码的主干分支 ,

事实上 , 在实际的开发中 , 往往不会直接修改和提交到master分支上 .

而是创建一个dev分支 , 在dev分支上 , 修改测试 , 没问题了 , 再把dev分支合并到master上 .

如果有了分支 , 刚才的难题就好解决了 , 如下图 :

在做 " 微信支付 " 时 , 我们创建一个wechat分支 .

把wechat分支commit , 此时 , master分支内容不会变 , 因为分支不同 .

当遇到紧急 bug 时 , 创建一个AliBug分支 .

修复 bug 后 , 把AliBug分支合并到master分支上 .

再次从容切换到wechat分支上 , 接着开发 " 微信支付 " 功能 , 开发完毕后 ,

把wechat分支合并到master分支上 .

### 4.2 查看分支

查看所有分支 git branch

例

git branch

\* master # 说明只有 master 分支 , 且处于 master 分支 .

### 4.3 创建分支

创建 dev 分支 git branch dev

git branch dev # 创建 dev 分支

git branch # 查看分支

dev

\* master # dev 分支创建成功 , 但仍处于 master 分支

### 4.4 切换分支

切换到 dev 分支 git checkout dev

再次查看

$ git branch

\* dev

master # 已切换到 dev 分支上

### 4.5 合并分支

当我们在 dev 上开发某功能 , 并测试通过后 , 可以把dev的内容合并到master分支 .

例 :

当前的 readme.txt 内容为 "so so", 在dev分支下 , 添加一行 "from dev"

并提交

git add readme.txt

git commit -m "mod in dev"

再次切换到master , 查看 readme.txt 的内容 , 仍为 'so so'

合并dev分支 , git merge dev , 如下 :

$ git merge dev

Updating c5364fe..412926b

Fast-forward

readme.txt | 1 +

1 file changed, 1 insertion(+)

再次查看 readme.txt 的内容 , 已变为 "soso from dev";

### 4.6 删除分支

git branch -d dev

Deleted branch dev (was 412926b).

### 4.7 快速创建和切换分支

git checkout -b dev # 创建 dev 分支并立即切换到 dev 分支

即起到git branch dev和git checkout dev的共同作用 .

## 五章 远程仓库

查看远程仓库

查看远程仓库 : git remote

查看仓库地址 : git remote -v

例 :

git remote -v

origin https://git.oschina.net/lianshou/test.git (fetch)

origin https://git.oschina.net/lianshou/test.git (push)

删除远程库

命令 : git remote remove < 远程库名 >

示例 : git remote remove origin

### 5.1 添加远程库

命令 : git remote add < 远程库名 > < 远程库地址 >

示例 :

git remote add origin https://git.oschina.net/lianshou/test.git

注 : 远程库名一般叫origin , 但并非强制 , 你可以自己起名 .

例 :

git remote add online https://git.oschina.net/lianshou/test.git

### 5.2 修改远程库名称

git remote rename < 旧名称 > < 新名称 >

例 :

git remote rename online oschina

公钥登陆

我们 push 本地仓库到远程时 , 总要输入用户名 / 密码 , 这很不方便 .

配置公钥 , 可以避免频繁输用户名 / 密码的麻烦 .

#### 1. 配置 ssh 格式的远程仓库地址

git remote add 远程仓库名 远程仓库地址

例 :

git remote add gitaddr git@git.oschina.net:lianshou/test.git

#### 2. 创建 ssh key

ssh-keygen -t rsa -C "youremail@example.com" , 把邮件地址换成你自己的邮件地址 , 一直回车 , 不用输入密码 . 完成后 , 可以在用户主目录里找

到 .ssh 目录 , 内有id\_rsa和id\_rsa.pub两个文件 . id\_rsa是私钥 , id\_rsa.pub是公钥 .

这两把钥匙是成对的 , 可以让分别持有私钥和公钥的双方相互认识 .

#### 3. 把公钥放在服务器

用记事本打开id\_rsa.pub , 复制公钥内容 .

登陆 git.oschina.net, 如下图 , 填入公钥并保存 .

4. push 本地仓库到远程 , 发现不用填密码了

git push gitaddr master