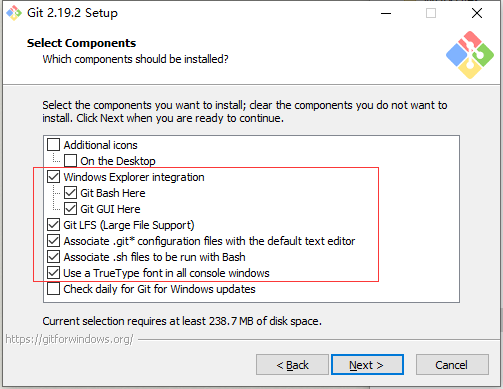
**全局配置（只需要在首次使用的机器上执行一次）**

这里仅说明windows环境下的git安装，linux及mac os安装及使用相对简单，参照官方说明即可。

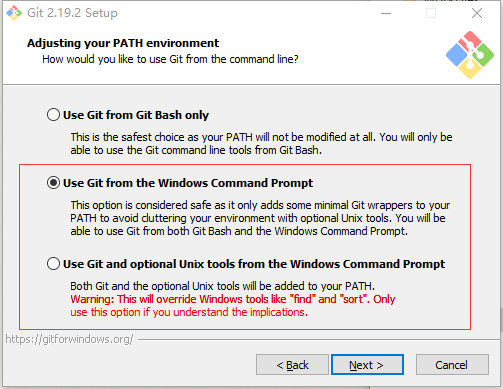
1. 安装git（已安装的可路过）

到<https://git-scm.com/downloads>下载windows版本的git安装程序，然后执行安装工作。

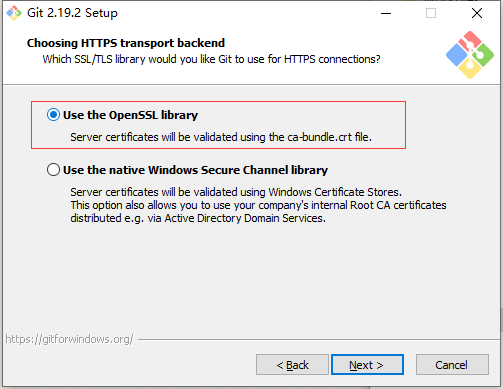
安装的组件最好设置如下（至少也要把git添加到资源管理器的右键菜单”Windows Explorer intergration”，方便随时调出使用）：



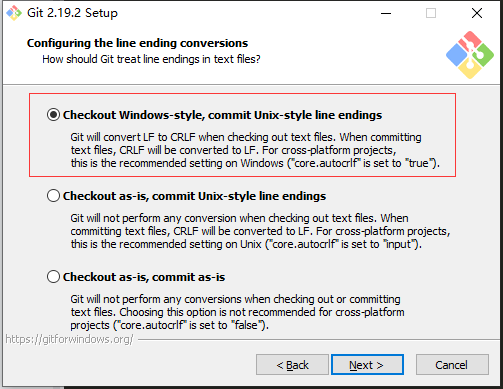
把git添加到系统环境变量中，方便在命令行模式下也能使用（红框中二选一，通常选择中间那个即可）：



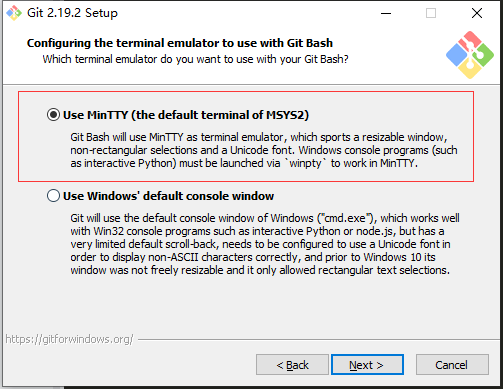
把openSSL库也一起装上，因为生成ssh证书时要用到这个库，否则还要自己再去另外装了在cmd下使用，就略麻烦了：



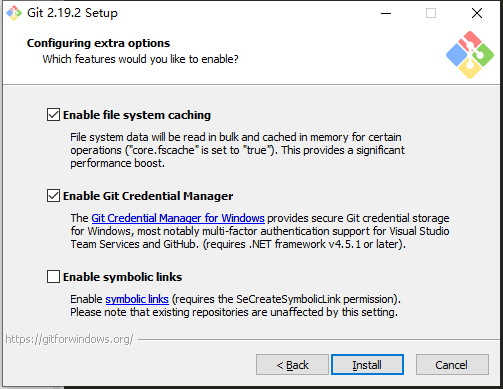
这是文件行结束符检测方式，通常使用第一个就行，windows的行结束符是“\r\n”，Unix及类Unix系统是“\n”，选择第一个，意思就是git在对文本文件进行检出时，会把\n转换为\r\n，而在提交时会将\r\n转换为\n进行提交，即保存在仓库中的始终是\n的行结束符版本，而“本地文件”使用的是\r\n，注意git的仓库也在本地，不一定就是远程，所以这里的“本地文件”加了双引号，以示区分：



Git Bash终端一般使用MinTTY，windows cmd没有颜色支持，字体粗大，默认不支持UTF8等缺点，用起来没有MinTTY舒服：

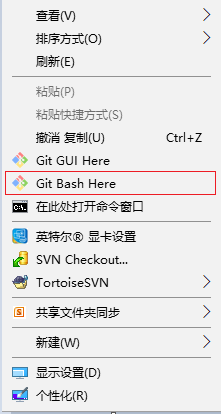


这里是扩展选项，依据需要勾上，一般三个都勾上也可以，建议至少勾前两个：



1. 创建ssh密钥

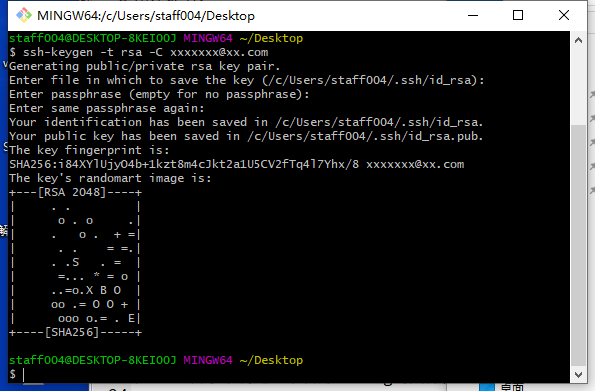
在任意目录下右击（如桌面），在右键菜单中选择”Git Bash Here”（非windows可直接在终端操作，但要先安装OpenSSL）：



弹出的终端窗口中输入以下命令：

ssh-keygen -t rsa -C xxx@xxx.com

后面的邮箱根据个人情况修改：



中间会有三个要输入的地方：

Enter file in which to save the key (/c/Users/\*\*\*/.ssh/id\_rsa): 这里输入密钥的文件名，默认值是id\_rsa，也是git默认使用的密钥（在不配置ssh的config时），所以如果只使用一个git远程域名，不用输入，直接回车即可。

接下来两个是密码，作为GIT使用，通常保持空密码即可，都直接回车即可。

生成的证书位于C:/Users/用户名/.ssh目录中，带.pub后缀的是公钥，它的内容就是要添加到gitlab中的！

1. 将ssh公钥添加到gitlab中。

打开gitlab并登录：[http://129.204.65.126:8801](http://129.204.65.126:8801/)

依次点击：右上角头像 》 settings 》 左边导航栏的SSH密钥



打开公钥文件id\_ras.pub（生成时重命名的请自行应变），全选文件内容，粘贴到gitlab的“密钥”文本域中，通常粘贴内容后，会自动填写“标题”文本框的内容，但为了好区分，可以自定义一个名称，好知道这个密钥对应的是哪台机器。

最后点击“增加密钥”按钮，就完成了密钥的添加，gitlab会向用户邮箱发送密钥添加成功的通知邮件。

**项目配置（每个项目都要做一次）**

前置相关说明

1、gitlab分支管理策略

目前暂定的分支管理策略如下：

每个项目共分为三个主要分支及多个开发分支：

master git默认主分支，也是每个里程中可执行版本的分支，主要用于测试，由管理人员从dev分支合并，开发人员通常无需关注，也不能向该分支推送。

dev 开发主分支，主要用于合并各开发分支的代码，使得各开发人员间可以按需同步代码，开发人员不能向该分支推送代码，但需要关注该分支的变更情况。

release 产品发布分支，主要用于发布产品级稳定代码，并使用tag对特定里程进行打版，一般简单的自动部署用的也是这个分支。

开发分支 根据参与开发人员的情况会存在多个开发分支，每个分支通常以开发人员的企业邮箱账号作为分支名（不含@及其后的字符，下同），开发人员推送的代码就是向自己的分支进行推送，除了特殊情况，每个开发人员只需要关心自己的分支即可。

2、开发人员本地分支管理建议

通常情况下，对于远程仓库而言，每个开发人员通常只需要关注自己分支及dev分支即可，所以从远程创建克隆代码、pull或push代码时，都主要是dev或自己的分支。

除此之外，git可以在本地开设很多其他的各种用途的分支，比如某个功能代码测试、破坏测试等，通这些分支都只是在本地创建及提交，在完成某个本地开发后也会合并到自己的开发分支后进行删除，所以本地可以开设任意的分支结构，但除dev及自己的开发分支外，其他分支不要提交到gitlab上。

对于自己的开发分支，在gitlab中要求是与自己的企业邮箱账号相同，但本地可以使用自己的名称，比如更短的my作为分支名进行开发，但提交代码到gitlab时，必须将代码提交到gitlab的对应开发分支上，比如gitlab上的开发分支名称是lixunhua，本地开发分支是my，那么提交代码到gitlab时，就是把my分支提交到远程的lixunhua分支上。

通常一个项目会由管理员创建好并设定好分支后，会把项目的gitlab仓库地址、项目的特征化文件（如配置文件）及其他说明，使用邮箱附件发送给每个开发人员，通常会有以下几个信息：

1. 仓库地址，[一般是git@129.204.65.126:\*\*\*\*\*.git](mailto:一般是git@129.204.65.126:*****.git)这种形式
2. 特征化文件及其他说明

由于项目在仓库中的初始信息是由管理员创建并配置好后提交到gitlab，再由各开发人员通过仓库地址去获取的，所以项目的配置需要进行一些操作进行适当的修改

通常的一个gitlab的地址至少有以下成份：

[git@host:namespace/project\_name.git](mailto:git@host:namespace/project_name.git)

host 就是gitlab服务器的域名

namespace 是项目所在的空间（通常是用户名或组名）

project\_name 是项目的名称（或称路径）

操作步骤（以仓库地址[git@129.204.65.126:group/project.git](mailto:git@129.204.65.126:group/project.git)及克隆存放目录D:\www来说明）：

1. 克隆代码

在要克隆代码的目录下右击，并打开git bash终端。（非windows系统就是将当前目录切换到要存放克隆代码的目录）

这里需要注意，克隆的结果是生成一个目录来装载的，git功能需要在克隆完成后切换到这个克隆下来的目录中才能操作！比如远程仓库地址是：[git@129.204.65.126:group/project.git](mailto:git@129.204.65.126:group/project.git)，如果是在D:\www目录中执行克隆操作，则克隆完成后会得到一个目录D:\www\project，而不是项目的代码会放置在D:\www这个目录下，要执行这个项目的git操作，还要再把git终端的当前目录cd到D:\www\project中才可以。

一般情况下，开发人员要克隆的分支是dev，因为该分支是开发人员需要关注的分支之一：

git clone –b dev [git@129.204.65.126:group/project.git](mailto:git@129.204.65.126:group/project.git)

1. 克隆完成后会得到D:\www\project目录，转到该项目目录

cd project

1. 配置项目的用户名及邮箱

git config user.name “你的用户名”

git config user.email 你的邮箱

注:用户名及邮箱如果中间无空格，就不需要双引号，如果有空格，就需要双引号包起来，有中文的话通常也需要双引号包起来，防止与ASCII字符不兼容

1. 从自己在远程仓库的开发分支中创建出自己的本地开发分支

git checkout –b my\_branch origin/you\_branch

注：my\_branch就是本地的开发分支名，you\_branch就是自己在远程仓库中的开发分支名（该分支名通常由管理员创建），建议二者保持相同。

例：git checkout –b lixunhua origin/lixunhua

创建完后就会自动切换到刚刚创建的分支了。

1. 执行项目其他说明的工作。

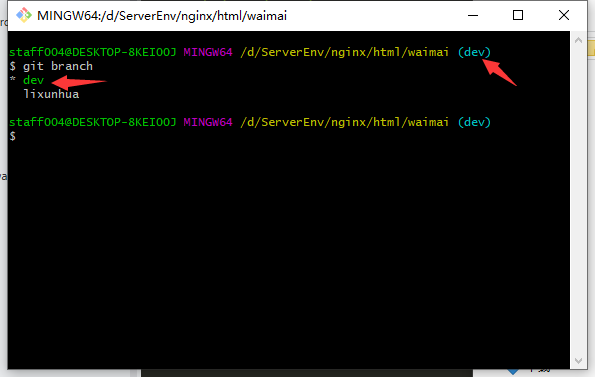
**git日常开发使用**

日常工作中，通常需要在每个工作日开始时（或需要同步各开发人员的代码时）从gitlab中拉取dev分支的代码，然后切换到自己的本地开发分支，并进行对dev分支的合并操作（注意：合并操作都是“进”操作，比如现在处于my\_branch分支，对dev进行合并操作时，是将dev的更改合并进当前分支，而不是将当前分支的更新输“出”到dev分支！）

以下列出的，只是最为常用的命令，更多的可以上网查阅相关资料。

**查看当前项目本地分支情况（有多少个分支，各是什么分支：**

git branch



注意箭头所指，右边括号中的就是当前分支名，而在git branch输出时，前边有个星号、标识为绿色的是当前分支。

同时，有这个括号才表示当前是git项目目录，没有的话就表示当前不是git项目目录。

**切换分支（或创建分支）**

git checkout my\_branch

切换到my\_branch，如果分支不存在，则提示错误

git checkout –b my\_branch

切换到my\_branch，如果分支不存在，则以当前分支为基版，创建这个新分支后再切换到该分支

**添加、提交**

这两个放在一起是因为它们总是相依，添加是为了提交，而提交的前提是添加。

添加是让git“知道”哪些文件有改动了，提交是将已“知道”的改动记录到版本库信息中。

git add 文件路径（相对git根目录）

将路径指定的文件添加到临时区，实际上并不常用

git add .

将所有改动添加到临时区，较为常用

git add –A

与git add .相同，但比git add .作用更多一些，不深究的话无大的差别，-A是参数--all的简写方式。

git commit –m “备注信息”

备注信息以双引号包起来时，只能单行，如果使用多行，可以用单引号，但只能在git bash有用，cmd下无效。

备注信息通常只简单明的介绍了做了何种改动，提交日志之类的相当简单的记录，而详细的开发记录，最好还是使用自己创建开发日志文件的方式进行记录。

**push（推送）**

git push origin you\_branch

将本地分支you\_branch推送到远程仓库，本地仓库的分支名与远程仓库的分支名相同。

其中origin一般是指远程仓库的名称，实际上不单止能有一个远程仓库，而且名称也可以随意取，怎么操作可自行查阅相关资料。

git push origin local\_branch:remote\_branch

将本地分支local\_branch推送到远程分支remote\_branch，当本地分支和远程分支不同名时就是使用这个方法进行推送。

**pull（同步）**

git pull origin you\_branch

将远程分支you\_branch拉取到本地并与本地的当前分支合并

git pull origin remote\_branch:local\_branch

将远程分支remote\_branch拉取到本地并与本地的local\_branch分支合并

外卖项目，用下面的命令，强制使用远程覆盖本地：

git checkout dev

git pull origin dev --force

git checkout 你的分支名

git pull origin 你的分支名 --force

**查看当前有哪些远程仓库**

git remote –v

**添加一个远程仓库**

git remote add 远程仓库名 仓库地址

如git remote add origin [git@\*\*\*\*:\*\*\*\*.git](mailto:git@****:****.git)

**版本回滚（移动）**

git reset --hard HEAD~1

回滚到最近一次commit之前的版本

特别说明：HEAD~1可以使用HEAD^表示，而HEAD~2也就是HEAD^^，依此类推。

HEAD表示的是当前的版本，而HEAD~1就是前一个版本，HEAD~2就是再前一个版本，依此类推。

git reset –hard commit\_id

commit\_id表示的是历史提交号，使用这个命令，可以将当前版本在各历史提交之间来回移动（回滚或撤消回滚。

git log

查看详细历史commit记录，会有commit\_id，但此ID为长ID

git log --pretty=oneline

查看历史commit记录，每个commit使用一行显示，commit\_id为长ID

git log –oneline

同上，但ID为短ID

git reflog

查看历史commit记录，commit\_id为短ID，只显示ID及提交备注

注：在查看commit记录时，如果一面显示不完，会进入按键继续显示的方式，显示完后会有<END>的标识，此时可按Q键退出列表显示回到命令模式。

关于--hard参数，这个参数是指“硬”版本移动，意思就是当指定版本到某个commit时，文件的内容就会变为这个commit时的那个版本的内容，假设有一个文件a.txt，各个commit与其内容如下：

commit短ID a.txt的内容

6ec5a12 555 <------ head

9ca5221 444

f52cc10 333

3312fba 222

d30ee9b 111

当前版本在6ec5a12的位置，当使用git reset --hard f52cc10时，a.txt的内容就会变为333。

与--hard相对应的一个参数是--soft意思就是“软”移动，它的作用是仅移动版本指向，不改变文件内容，所以与目标版本不同的文件都被记录到工作暂存区。

比如上面的例子，当使用命令git reset --soft f52cc10时，a.txt的内容还是555，但当前版本却是位于f52cc10的位置，而这时git所看到的a.txt最原始的内容是333，但被改成了555。

**--soft参数最有用的是在误commit进行回滚一版本，然后确认无误再次提交！**

git reflog会详细记录版本HEAD指针的移动情况，比如上面例子中，HEAD从6ec5a12移动到f52cc10后，git reflog会看到详细的移动记录，而不会使得6ec5a12和9ca5221消失，而git log则只能显示到f52cc10，6ec5a12和9ca5221是看不到了（消失了），但依然可以通过短ID 6ec5a12和9ca5221将版本再次回遡到指定的版本。

**合并分支**

git merge dev

将当前分支与dev分支进行全并，注意，合并都是数据“进”操作，即目标分支（这里是dev分支）的数据合并进当前分支，而目标分支并不会受当前分支的影响！

如果全并成功，则目标分支的改动会应用到当前分支，如果合并有冲突，则合并操作失败，命令会在有冲突的文件上对冲突的块进行标记，这时需要手动解决冲突，或撤消合并操作，冲突块在文件中的表现一般为如下形式：

<<<<<<< HEAD

22222222222【当前分支版本的内容】

=======

aaaaaaaaaa【目标分支版本的内容】

>>>>>>> dev

注意看，HEAD指的就是当前分支的版本中的内容，而后面的dev就是指目标分支中的版本的内容，因为这两个内容有冲突，所以无法合并，需要手动解决这些冲突，如果使用的pull从远程接取，则这里的dev可能会是一串长ID。解决冲突的办法当然是将冲突的内容修正后，再次add和commit即可，如果冲突文件还存在以上格式的字符串，则表示冲突未解决，git是不允许commit的。

git reset merge

将版本退回到merge前最近一次commit的版本，也就是在误合并后进行撤消合并。

**git日常开发用例**

1. 开始上班（或需要同步代码），将dev分支从远程仓库拉取下来（会自动与当前分支合并）。

git checkout my\_branch

git pull dev

注：如果当前分支已在my\_branch，则可以不用再执行这个切换分支的命令。

1. 解决冲突（如果有）
2. 开始开发工作
3. 下班前提交本地仓库（或需要将修改推送到远程仓库）

git add -A

git commit –m "... 20190328 other desctipt”

注：备注中的“...”表示一种约定，意思是告诉管理员，因某些原因，本次提交的代码还不应该合并到dev中，否则可能会引起其他开发人员合并代码后引起某些问题，当然这里只是示例，具体约定需要另外商定。

1. 将自己的开发分支推送到远程仓库

git push origin my\_branch

1. 如果是管理员，则还需要根据开发人员提交的约定，将可以合并到dev的开发分支合并到dev上，并初步解决冲突，然后把dev推送到远程仓库上，以使得在下一工作日开始时，每一个开发人员可以将所有其他开发人员的改动应用到自己的开发分支上。

2019-04-16

git远程仓库地址已改，请知悉：  
网页登录地址：[http://gitlab.nnmtl.com](qq://txfile/)  
现有gitlab更换远程仓库操作（以外卖项目为例）  
$ git remote -v 查看远程仓库名，一般只有origin  
$ git remote remove origin 删除现有远程仓库  
$git remote add origin [git@gitlab.nnmtl.com:waimai/waimai.git](mailto:git@gitlab.nnmtl.com:waimai/waimai.git) 使用新的远程仓库地址  
附：（可选）如果配置有多个ssh请书设置，记得修改~/.ssh/config文件中对应的配置域名！本地只有一个ssh默认证书时则不需要进行这个可选操作。