Git

1. Git下载和安装步骤

> 详细安装步骤。git工具安装完成之后，在开始菜单里面找到“Git” -> “Git Bash”，出现一个命令行窗口就说明安装成功了。 安装成功之后，在命令行部署自己的用户名和Email地址，方便之后的版本管理和远程推送。
> 部署命令
>
> - git config --global user.name "Your Name"
> - git config --global user.email "email@example.com"

2. Git基础教程

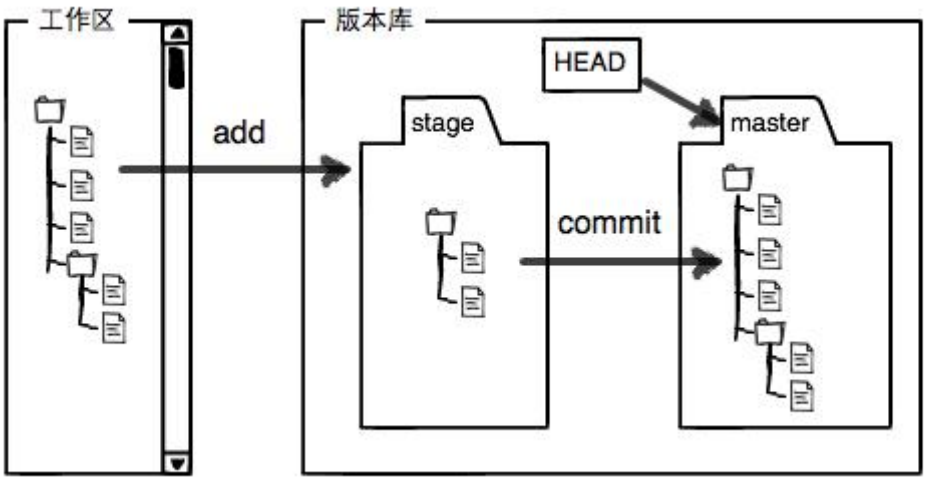
2.1 本地创建版本库

> 在本地创建一个目录，目录里的文件可以被git管理，意味着每个文件的删除、修改都能被跟踪，不同版本之间可以来回转换 [1]。
>
> #### 2.1.1 新建空目录和初始化本地仓库:
>
> - mkdir TEST
> - cd TEST
> - git init
>
> #### 2.1.2 在TEST目录下面创建一个show.txt文件，内容自定义，创建完之后添加到本地仓库并提交到本地仓库:
>
> - touch shou.txt
> - vi show.txt
> - git add show.txt
> - git commit -m "提交说明[2]"
>
> #### 2.1.3. 当提交整个项目的所有的文件时，可以用下面的命令:
>
> - git add -A
>
> 修改编辑器为vim
>
> - git config –global core.editor
> - git commit
>
> #### 2.1.4 确认修改,查看修改的地方和内容
>
> - git diff <filename>

2.2 版本回溯

2.2.1 知识点

在版本回溯之前，先简单简单介绍一下git工具中的暂存区和工作区。
工作区
我们当前操作的所在目录，这个目录下有一个隐藏的版本库，.git。git的版本库里面存有许多东西，其中最重要的就是暂存区（stage），还有git为我们自动创建的master分支，以及指向master的一个指针叫HEAD。
暂存区
之前我们往本地仓库添加文件的时候，先git add，其实际就是把文件修改添加到暂存区，后git commit，其实际就是把暂存区的所有内容提交到当前分支。如下图所示。



2.2.2 版本切换

每次我们修改某个地方，修改之后提交到版本库，每当你觉得文件修改到一定程度的时候，就可以“保存一个快照”，这个快照在Git中被称为commit。一旦你把文件改乱了，或者误删了文件，还可以从最近的一个commit恢复，然后继续工作。
git log

git log 可以让我们查看每次提交的历史，其默认输出gcommit hash, author, date, commit message等信息，这样会看起来很费劲，因此我们可以使用参数来简化或者美化git log输出。

- git log --oneline（这个命令简化git log的默认的输出，仅仅输出commit hash 前7个字符串和commit message.）
- git log --stat(输出修改的信息位置)

在别的文章看到一条很好的命令

- git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative
  该命令把文件提交的时间操作者等信息详细的显示出来，十分的便利。

git reset

git reset 命令可以用于回退版本，可以指定退回某一次提交的版本。

- git reset --hard HEAD^(回退上一个版本)
- git reset --hard HEAD^^(回退上上一个版本)
- git reset --hard 2b4ea78(回退到指定版本，版本号写几位就可以了)

git reset 命令还可以把暂存区的修改回退到工作区

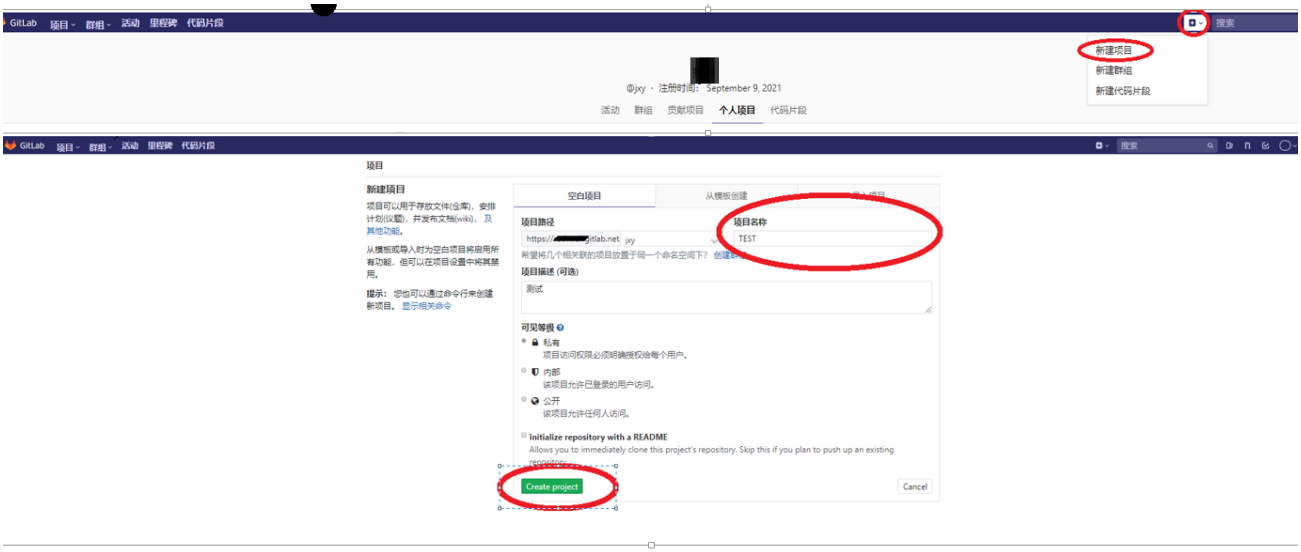
- git reset HEAD <file>(可以把暂存区的修改撤销掉)

2.3 远程仓库

此前操作都是在本地仓库操作的，如果想要跟线上创建的仓库同步操作，此时你需要把两个仓库连接起来。下面通过实际操作来讲解。（注：作者用的是GitLab，跟Github操作流程是一样的。

2.3.1 添加远程仓库

在GitLab上点击左上角的“+”。创建一个新仓库，仓库名按照自己想法来命名。



远程仓库创建完成之后，会有一些命令提示配置仓库和推送的简单命令。

2.3.2 创建SSH key

此时你在本地目录下，直接git clone，git终端会提示你下载错误，其原因一般都是没有生成SSH密钥并把密钥配置到GitLab服务端。

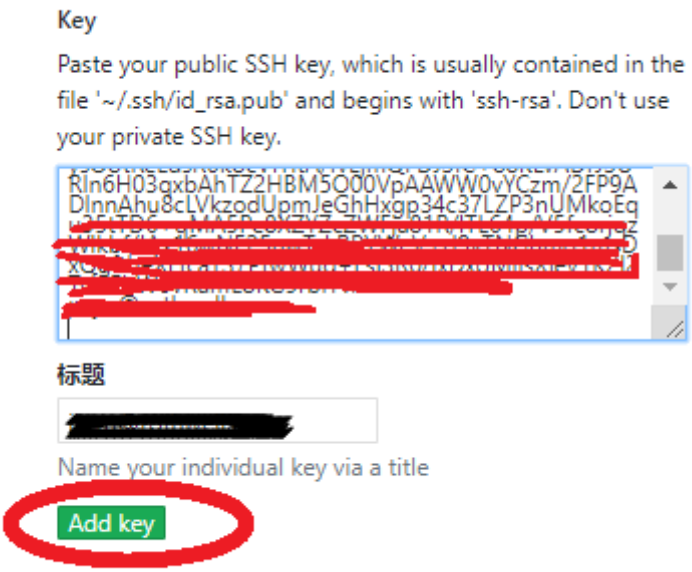
因此我们需要在本地目录下创建密钥，也就是SSH key。

- ssh-keygen -t rsa -C "youremail@example.com"
  把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可。

在本地目录生成密钥之后，可以使用 cat ~/.ssh/id_rsa.pub 查看。

2.3.3 配置GitLab

在本地目录生成SSH key 之后，我们需要把它配置到GitLab服务端，以此来访问远程仓库。



2.3.4 克隆远程仓库

前面操作无误之后，就可以在本地目录下克隆远程仓库了。

- git clone ssh://git@rothwell.gitlab.net.cn:5022/jxy/TEST.git

```
Administrator@DESKTOP-R4I32M7 MINGW64 /d/git
$ git clone ssh://git@r      l.gitlab.net.cn:5022/jxy/TEST.git
Cloning into 'TEST'...
warning: You appear to have cloned an empty repository.
```

2.3.5 推送

此前我们已经把远程仓库已经克隆到本地目录，在经过本地操作之后，内容已经有了更新，我们想把更新内容推送到远程仓库，那么推送步骤有哪几步呢?
(1)关联上远程仓库

- git remote add origin ssh://git@rothwell.gitlab.net.cn:5022/jxy/TEST.git(关联)
- git remote -v(查看)

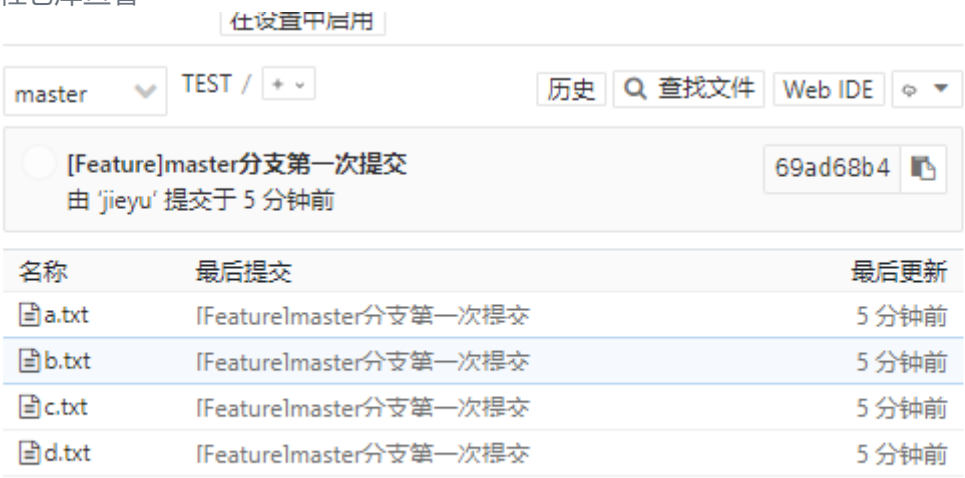
```
Administrator@DESKTOP-R4I32M7 MINGW64 /d/git/TEST (master)
$ git remote add origin ssh://git@      .gitlab.net.cn:5022/jxy/TEST.git

Administrator@DESKTOP-R4I32M7 MINGW64 /d/git/TEST (master)
$ git remote -v
origin  ssh://git@      l.gitlab.net.cn:5022/jxy/TEST.git (fetch)
origin  ssh://git@      .gitlab.net.cn:5022/jxy/TEST.git (push)
```

(2)推送到远程仓库

- git push -u origin master

```
Administrator@DESKTOP-R4I32M7 MINGW64 /d/git/TEST (master)
$ git push -u origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 371 bytes | 371.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To ssh://rothwell.gitlab.net.cn:5022/jxy/TEST.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

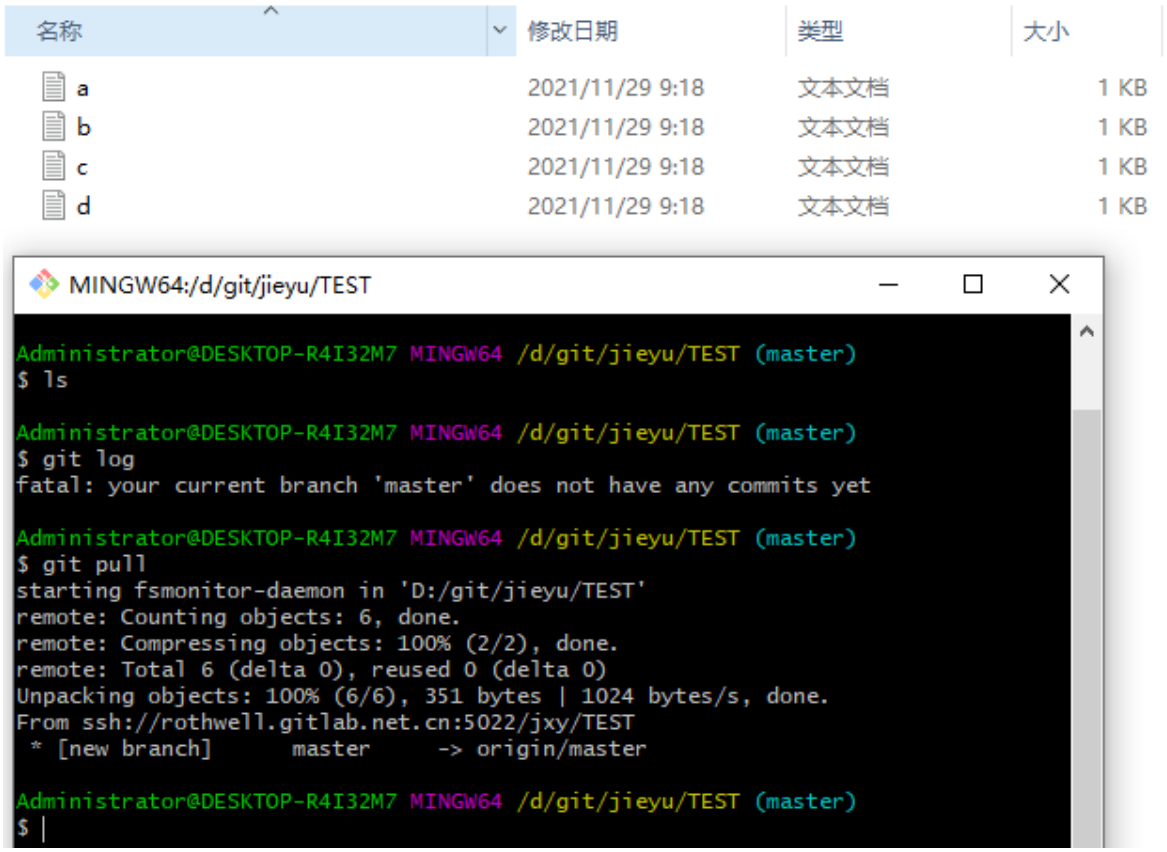
在远程仓库查看



(3)更新到本地仓库
在上一步推送中，此时是用户A推送了更新内容，而用户B却还是上次内容，此时用户B打算获取最新内

容,此时用户B需要在克隆仓库的本地目录下，运行下面的命令

- git pull



2.4 分支管理

在版本回退中，git把每次提交串成一条时间线，这条时间线就是一个分支，截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。每次提交，master分支就会往前移动一步。分支之间修改文件是互不影响的，在合并分支之前，用户A在分支dev修改的内容，用户B在master分支是看不到的，但是并不影响俩者的工作情况。

2.4.1创建分支dev并且切换到dev分支上

- git checkout -b dev

相当于下面两行命令

- git branch dev
- git checkout dev

2.4.2 合并分支

git merge命令用于合并指定分支到当前分支

- git merge dev

2.4.3 删除分支

- git branch -d dev

创建分支的命令，在最新版本的git中，可以使用git switch来创建和切换分支

- git switch -b dev

直接切换到已有分支

- git switch master

2.5 变基

除了可以使用 git merge合并分支，还可以用git rebase变基操作，所谓变基，就是变更基线。当我们在分支操作，合并到master分支，使用下面命令查看分支提交历史。

- git log --graph --pretty=oneline --abbrev-commit(git merge)

```
Administrator@DESKTOP-R4I32M7 MINGW64 /e/git clone/git (master)
$  git log --graph --pretty=oneline --abbrev-commit
*   ba0d855 (HEAD -> master) Merge branch 'merge'
|\
| * ab7df99 (merge) [Feature]777777
* | a1df706 (dev) [Feature]666666
|/
* bb57346 [Feature]555555
* 89dec0a [Feature]333333
* 4a9b1d7 [Feature]2222222
* 8e74c58 [Feautre]111111
```

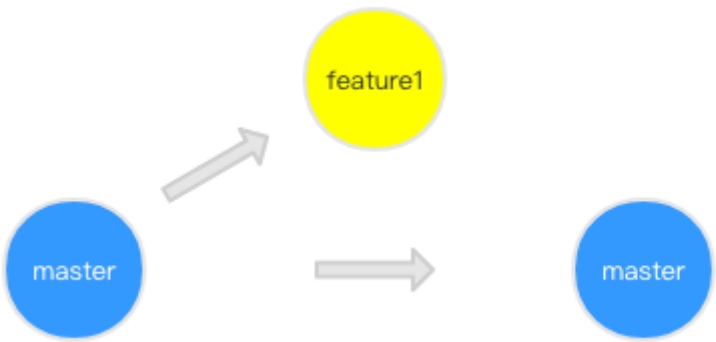
- git log --graph --pretty=oneline --abbrev-commit(git rebase)

```
Administrator@DESKTOP-R4I32M7 MINGW64 /d/git/test1 (master)
$  git log --graph --pretty=oneline --abbrev-commit
* 4601e64 (HEAD -> master, feature2) [Feature]第一次修改chh.txt文件
* 5c147a3 (feature1) [Feature]master分支第一次修改
* de8ad2c [Feature]测试
```

不难看出，虽然分支合并没有发生冲突，但是git rebase 更加简洁，修改记录看起来更为清晰。两者之间操作有何区别呢？此时我们新建一个分支feature1，

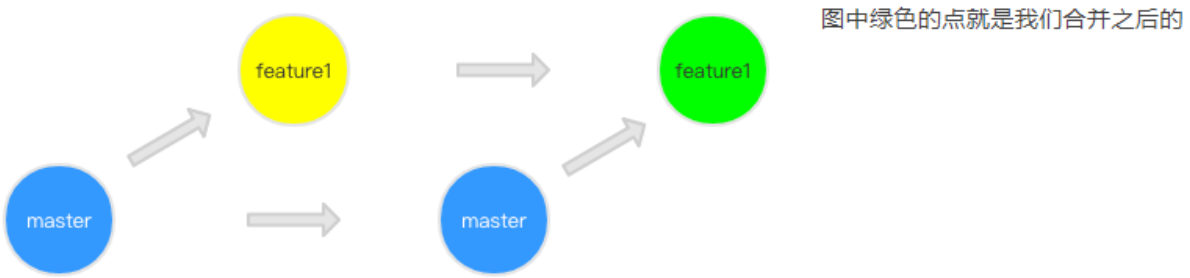
- git branch feature1

此时我们在分支feature1上正在修改某个文件，而另外一个分支的操作员把最新的更改已经推送到



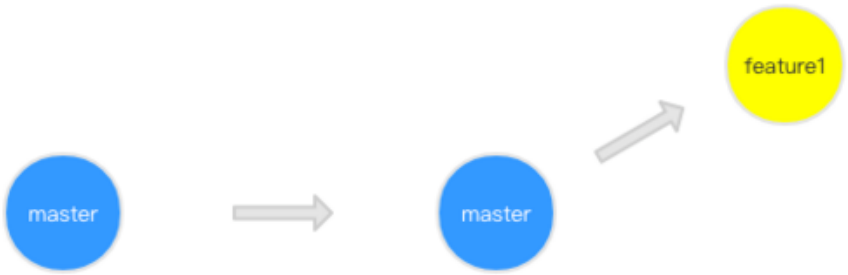
master分支上了，如下所示。
此时我们需要把master分支的改动同步，当使用命令 git merge时，此时合并信息如下

- git merge master



此时分支合并开起来似乎很整洁，但是分支多了之后，就会显得杂乱无章，这个时候git rebase 作用就十分明显了。

- git rebase master 此时git rebase操作过程：
  (1)首先， git 会把 feature1 分支里面的每个 commit 取消掉；
  (2)其次，把上面的操作临时保存成 patch 文件，存在 .git/rebase 目录下；
  (3)然后，把 feature1 分支更新到最新的 master 分支；
  (4)最后，把上面保存的 patch 文件应用到 feature1 分支上；



2.6 解决合并冲突

git 合并中产生冲突具体有两种情况:
(1)两个分支中修改了同一个文件（不管什么地方）
(2)两个分支中修改了同一个文件的名称 例如，在本地库中两个不同的分支，修改了同一个文件的同一块代码，两分支先后合并到master分支上，master分支在合并第二个分支时候，报错，合并冲突。如下图所示，

```
Administrator@DESKTOP-R4I32M7 MINGW64 /d/git/test1 (master)
$ git rebase feature2
error: could not apply 2f06399... [Feature]feature1修改chh.txt文件
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 2f06399... [Feature]feature1修改chh.txt文件
Auto-merging chh.txt
CONFLICT (content): Merge conflict in chh.txt
```

解决方法:
在当前分支上:

- git add
- git rebase --continue

2.7 git常用命令

> mkdir： XX (创建一个空目录 XX指目录名)
> pwd： 显示当前目录的路径
> git init 把当前的目录变成可以管理的git仓库，生成隐藏.git文件
> git add XX 把xx文件添加到暂存区去。
> git commit –m “XX” 提交文件 –m 后面的是注释
> git status 查看仓库状态
> git diff XX 查看XX文件修改了那些内容
> git log 查看历史记录
> git reset --hard HEAD^ 或者 git reset --hard HEAD~ 回退到上一个版本(如果想回退到100个版本，使用git reset –hard HEAD~100)
> cat XX 查看XX文件内容
> git reflog 查看历史记录的版本号id
> git checkout -- XX 把XX文件在工作区的修改全部撤销
> 点击Git查看更过详细命令。

总结

> Git是一个非常强大的版本管理工具，还有很多的地方去学习，有后续自己会继续更新。
> 借鉴文档 借鉴文档 借鉴文档

---

1. 前提是当前操作窗口未关闭 ↩
2. 本次提交的说明，记录本次提交的改动 ↩