Git简易使用

安装Git(Windows)

- 详细流程参照AnyShare://AnyShare产品线/Dorado/pShare/应用小组/2.技术方案/svn迁移git使用.pdf。安装完成后，在开始菜单里找到“Git”->“Git Bash”，打开一个类似命令行窗口的东西，就说明Git安装成功。
- Git是分布式版本管理系统，使用用户名、邮箱作为唯一标识，安装完成后，还需要最后一步设置，在命令行输入：git config –global 参数，有了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然你也可以对某个仓库指定的不同的用户名和邮箱。

```
`$ git config --global user.name "Your Name"`

`$ git config --global user.email "email@example.com"`
```

搭通自己的电脑与远程服务器的传输通道

- 1.创建SSH Key

  a.在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id_rsa和id_rsa.pub这两个文件，如果已经有了，可直接进行第3步。

  b.如果没有，打开Shell（Window下打开Git Bash），创建SSH Key：输入ssh-keygen命令一路默认执行，执行成功后便会在用户主目录下的.ssh目录中生成公钥文件(id_rsa.pub)和私钥文件(id_rsa)

  c.成功：可在用户主目录里找到.ssh目录，里面有id_rsa和id_rsa.pub两个文件，这两个就是 SSH Key 的密钥对，id_rsa是私钥，不能泄露出去，id_rsa.pub是公钥，可以放心地告诉任何人。

- 2.在服务器上放入你的SSH Key（http://192.168.4.2/）

  使用域账号进行登录，打开Settings -> SSH Key设置 -> SSH Public Keys，点Add Key，将生成的id_dsa.pub文件中的内容复制到key 文本框里，点击add即可完成gerrit 中 ssh key的添加。

- 3.Git支持SSH协议，所以，Git服务器只要知道了你的公钥，就可以确认只有你自己才能推送。

从远程库克隆到本地

- 选择Projects -> ShareWeb，选择ssh协议，Clone with commit-msg hook，点击copy复制命令，命令行进入本地目录，粘贴执行：

```
`$ git clone ssh://quan.tiantian@192.168.4.2:29418/ShareWeb && scp -p -P 29418 quan.tiantian@192.168.4.2:hooks/commit-msg ShareWeb/.git/hooks/`
```

工作区 vs 暂存区

- 1.工作区：当前电脑上看到的目录，以及目录下的文件,它持有实际文件（.git隐藏目录版本库除外）；
- 2.暂存区（stage）：它像个缓存区域，临时保存你的改动；

- 3.版本库(Repository):

  工作区有一个隐藏目录.git，属于版本库，这就代表它是一个Git可以管理的仓库。版本库存了很多东西，最重要的是暂存区(stage)，Git为我们自动创建了第一个分支master,以及指向master的一个指针HEAD。

- 4.Git提交文件到版本库有两步:

  a. 使用 git add 把文件添加进去，实际上就是把文件添加到暂存区。

  b. 使用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支上。

添加与提交

- 1.切换到自己的开发分支
- 2.在本地版本库目录下，新建一个文件readme.txt，将文件添加到暂存区

```
`$ git add readme.txt`
```

- 3.diff文档

```
`$ git diff readme.txt`
```

- 4.查看提交状态

```
`$ git status`
```

- 5.提交文件到仓库，引号中内容是提交的注释

```
`$ git commit -m "add readme.txt"`
```

- 现在，你的改动已经提交到了 HEAD(指向你最近一次提交后的结果)，但是还没到你的远端仓库。

推送改动

- 你的改动现在已经在本地仓库的 HEAD 中了。执行如下命令以将这些改动提交到远端仓库的 HEAD:

```
`$ git push origin HEAD:refs/for/develop`
```

- 待审核通过后，你的改动才真正放到了远程仓库中。

分支

- 分支是用来将特性开发绝缘开来的。在你创建仓库的时候，master 是版本发布线(默认)。在其他分支上进行开发，完成后再将它们合并到 develop 分支(研发线)上。
- 1.在 develop 拉取一个你自己的分支(按照一定的分支命名规则)，叫做"dev"，并切换过去:

```
`$ git checkout -b dev`
```

- 2.切换回主分支：

```
`git checkout develop`
```

- 3.合并分支
- a.git merge命令用于合并指定分支到当前分支上。先切换 develop 分支，再合并dev分支到 develop 分支

```
`$ git checkout develop`

`$ git merge dev`
```

- b.解决冲突：如果develop、dev同时修改同一文件，则会出现冲突

  1.Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，其中<<<HEAD是指主分支修改的内容，>>>>>dev 是指dev上修改的内容；

  2.修改冲突的部分，重新提交

- 4.再把新建的分支删掉（可根据实际情况决定什么时候删掉）：

```
`git branch -d dev`
```

- 5.将develop分支推送到远端仓库：

```
`git push origin HEAD:refs/for/develop`
```

- 6.总结创建与合并分支命令如下：

```
查看分支：`git branch`

创建分支：`git branch name`

切换分支：`git checkout name`

创建+切换分支：`git checkout -b name`

合并某分支到当前分支：`git merge name`

删除分支：`git branch -d name`
```

版本回退

- 1.查看历史记录

```
`$ git log`

显示精简信息：`$ git log --pretty=oneline`
```

- 2.回退版本
- 这里有几个参数

  a.--soft：缓存区（stage）工作目录中的内容不作任何改变，仅仅把HEAD指向commit。这个模式的效果是，执行完毕后，自从commit以来的所有改变都会显示在git status的"Changes to be committed"中。

  b.--hard：重设缓存区（stage）和工作目录，自从commit以来在工作目录中的任何改变都被丢弃，并把HEAD指向commit。

  c.--mixed：仅重置缓存区（stage），但是不重置工作目录。这个模式是默认模式，即当不显示告知git reset模式时，会使用mixed模式。这个模式的效果是，工作目录中文件的修改都会被保留，不会丢弃，也不会被标记成"Changes to be committed"，但是会打出什么还未被更新的报告。

  d.根据实际需要选用不同的参数

```
回退到上一个版本 `$ git reset --hard HEAD^`

回退到上上一个版本 `$ git reset --hard HEAD^^`

回退到前100个版本 `$ git reset --hard HEAD~100`

按版本号进行回退 `$ git reset --hard 版本号`
```

- 3.查询版本号

```
`$ git reflog`
```

撤销修改

- 1.如果你知道要删掉哪些内容的话，直接手动更改去掉那些需要的文件，然后add添加到暂存区，最后commit。
- 2.可以按以前的方法直接恢复到上一个版本。使用 git reset –hard HEAD^
- 3.未commit前 git checkout —- readme.txt

```
`$ git checkout -- readme.txt`
```

- a. readme.txt自动修改后，还没有放到暂存区，使用撤销修改就回到和版本库一模一样的状态。

- b. readme.txt已经放入暂存区了，接着又作了修改，撤销修改就回到添加暂存区后的状态。

删除文件

- 1.删除文件

```
`$ rm readme.txt`
```

- 2.提交修改

```
`$ git commit -m “delete readme.txt”`
```

- 3.未commit前，可使用checkout命令恢复

```
`$ git checkout –- readme.txt`
```

更新

- Git中从远程的分支获取最新的版本到本地有这样2个命令：
- 1.git fetch 相当于是从远程获取最新版本到本地，不会自动merge

```
`$ git fetch origin <name>`
```

- 2.git pull 相当于是从远程获取最新版本并merge到本地

```
`$ git pull origin <name>`
```

- 3.在实际使用中，git fetch更安全一些，因为在merge前，我们可以查看更新情况，然后再决定是否合并

多人协作

- 当你从远程库克隆时候，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且远程库的默认名称是origin。要查看远程库的信息使用 $ git remote；要查看远程库的详细信息使用 $ git remote –v
- 1.从master拉取 develop 分支（研发线）并切换过去：

```
`$ git checkout -b develop`
```

- 2.从远程获取最新版本并 merge 到本地的 develop 分支

```
`$ git pull origin develop`
```

- 3.开发工作完成后

添加某个文件:`$ git add 文件路径`

或添加所有修改过的文件: `$ git add .`

`$ git commit -m “注释”`

- 4.推送到远程仓库 develop 分支

```
`$ git push origin HEAD:refs/for/develop`
```

- 5.测试通过后合并到 master 分支

```
`$ git checkout master`

`$ git merge develop`
```