Git及GitLab相关

关于 Git 你应该知道的东西

Git 是一个分布式版本控制系统。分布式的意思是，每个人电脑上都是一份完整的代码库，包含了所有的代码提交历史。由于 Git 分布式的特点，在没有网络的情况下，依然可以自由地将代码提交的本地的代码库中，等网络恢复后再推送到服务器，开发更加灵活和自由。

重要概念：本地一个代码库，对本地文件的所有操作，最后都是提交到这个代码库中。同时可以设置多个远程（remote，默认的remote通常用origin表示），当你要将代码更新到服务器上时（称作push），就通过设置的remote，更新到指定的服务器。

> 关于多个remote：例如需要同时将代码同步到多个代码管理仓库，例如 Github 和自己公司中内网的 Gitlab。那么可以将默认remote 设置为公司自己的源码管理服务器，同时设置另外一个remote为 Github。这样即可以将代码提交到自己公司的 Gitlab，同时也可以提交到 Github。

一些术语

- Fetch（获取），从远程代码库更新数据到本地代码库。注意：Fetch 只是将代码更新到本地代码库，你需要检出（check out）或与当前工作分支合并（merge）才能在你的工作目录中看到代码的改变。
- Pull（拉取），从远程代码库更新数据到本地代码库，并与当前工作分支合并，等同于 Fetch + Merge。
- Push（推送），将本地代码库中已提交（commit）的数据推送到指定的 remote，没有 commit 的数据，不会push
- HEAD，指向你正在工作中的本地分支的指针
- Master 分支：主分支，所有提供给用户使用的正式版本，都在这个主分支上发布。
- Tags（标签）：用来记录重要的版本历史，例如里程碑版本
- Origin：默认的 remote的名称
- Git clone（克隆版本库）：从服务端将项目的版本库克隆下来
- Git init（在本地初始化版本库）：在本地创建版本库的时候使用

工作流程

1. 对代码进行修改
2. 完成了某项功能，提交（commit，只是提交到本地代码库），1-2可以反复进行，直到觉得可以推送到服务器上时，执行3
3. 拉取（pull，或者用获取 fetch 然后再手动合并 merge）
4. 如果存在冲突，解决冲突
5. 推送（push），将数据提交到服务器上的代码库

Gitlab 可以做什么

Gitlab 是 Git 服务端的集成管理平台，提供了：

1. 代码托管服务
2. 访问权限控制
3. 问题跟踪，bug的记录、跟踪和讨论
4. Wiki，项目中一些相关的说明和文档

5. 代码审查，可以查看、评论代码

安装与配置

需要安装以下工具：

- Git（Git 主程序） http://git-scm.com/

初次运行 **Git** 前的配置

一般在新的系统上，我们都需要先配置下自己的 Git 工作环境。配置工作只需一次，以后升级时还会沿用现在的配置。当然，如果需要，你随时可以用相同的命令修改已有的配置。

Git 提供了一个叫做 `git config` 的工具（译注：实际是 `git-config` 命令，只不过可以通过 `git` 加一个名字来呼叫此命令。），专门用来配置或读取相应的工作环境变量。而正是由这些环境变量，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：

- `/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。
- `~/.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。
- 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。

在 Windows 系统上，Git 会找寻用户主目录下的 `.gitconfig` 文件。主目录即 `$HOME` 变量指定的目录，一般都是 `C:\Documents and Settings\$USER`。此外，Git 还会尝试找寻 `/etc/gitconfig` 文件，只不过看当初 Git 装在什么目录，就以此作为根目录来定位。

用户信息

第一个要配置的是你个人的用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，所以会随更新内容一起被永久纳入历史记录：

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

如果用了 `--global` 选项，那么更改的配置文件就是位于你用户主目录下的那个，以后你所有的项目都会默认使用这里配置的用户信息。如果要在某个特定的项目中使用其他名字或者电邮，只要去掉 `--global`选项重新配置即可，新的设定保存在当前项目的 `.git/config` 文件里。

查看配置信息

要检查已有的配置信息，可以使用 `git config --list` 命令：

```
$ git config --list
user.name=Scott Chacon
user.email=schacon@gmail.com
color.status=auto
```

```
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

有时候会看到重复的变量名，那就说明它们来自不同的配置文件（比如 `/etc/gitconfig` 和 `~/.gitconfig`），不过最终 Git 实际采用的是最后一个。

也可以直接查阅某个环境变量的设定，只要把特定的名字跟在后面即可，像这样：

```
$ git config user.name
Scott Chacon
```

##获取帮助

想了解 Git 的各式工具该怎么用，可以阅读它们的使用帮助，方法有三：

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

比如，要学习 config 命令可以怎么用，运行：

```
$ git help config
```

我们随时都可以浏览这些帮助信息而无需连网。 不过，要是你觉得还不够，可以到 Freenode IRC 服务器（irc.freenode.net）上的 `git` 或 `github` 频道寻求他人帮助。这两个频道上总有着上百号人，大多都有着丰富的 Git 知识，并且乐于助人。

常见问题

1.

- 问题：Gitlab的develop角色的人没有权限无法提交的问题解决方案.建好仓库，将其他几位同事添加进来，角色分配为Develop。之后提交初始代码到master分支后，他们用sourceTree拉取代码后进行开发，之后再提交时发现无法提交，提示没有权限
- 解决方案： 在项目的【Setting】中的【Protected branches】可以设置哪些分支是被保护的，默认情况下【master】分支是处于被保护状态下的，develop角色的人是无法提交到master分支的，在下面的【Developers can push】打上钩就可以了