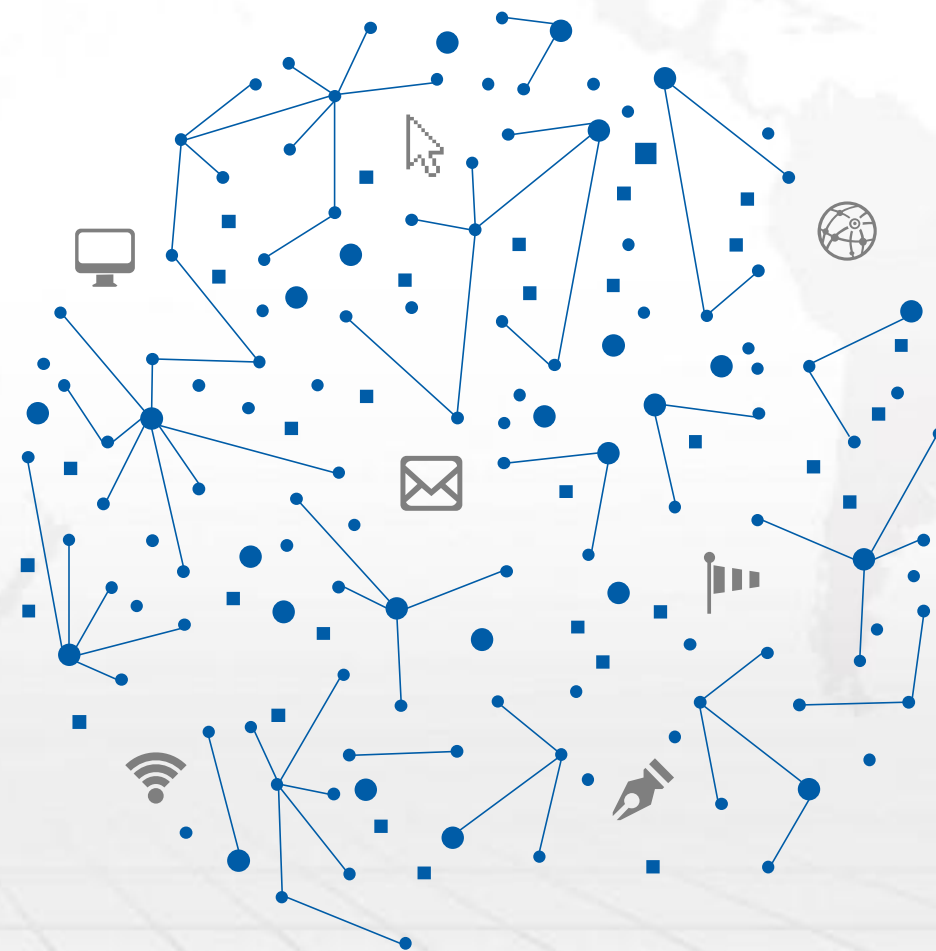


# Gitlab使用简介



CONTENT

# 目录



一

## 版本控制

Reversion Control

二

## Git简介

Git Introduction

三

## Gitlab简介

Gitlab Introduction

四

## Gitlab使用

Gitlab use



# 版本控制

Reversion control

## 版本控制

- 版本控制(Revision control)是指对软件开发过程中各种程序代码、配置文件及说明文档等文件变更的管理，是软件配置管理的核心思想之一，是一种方便查看历史更改记录，备份以便恢复以前的版本的软件工程技术。
- 简单说就是用于管理多人协同开发项目的技术。
- 如果忽视版本控制将产生诸多问题，如软件代码的一致性、软件内容的冗余、软件过程的事务性、软件开发过程中的并发性、软件源代码的安全性，以及软件的整合等问题。

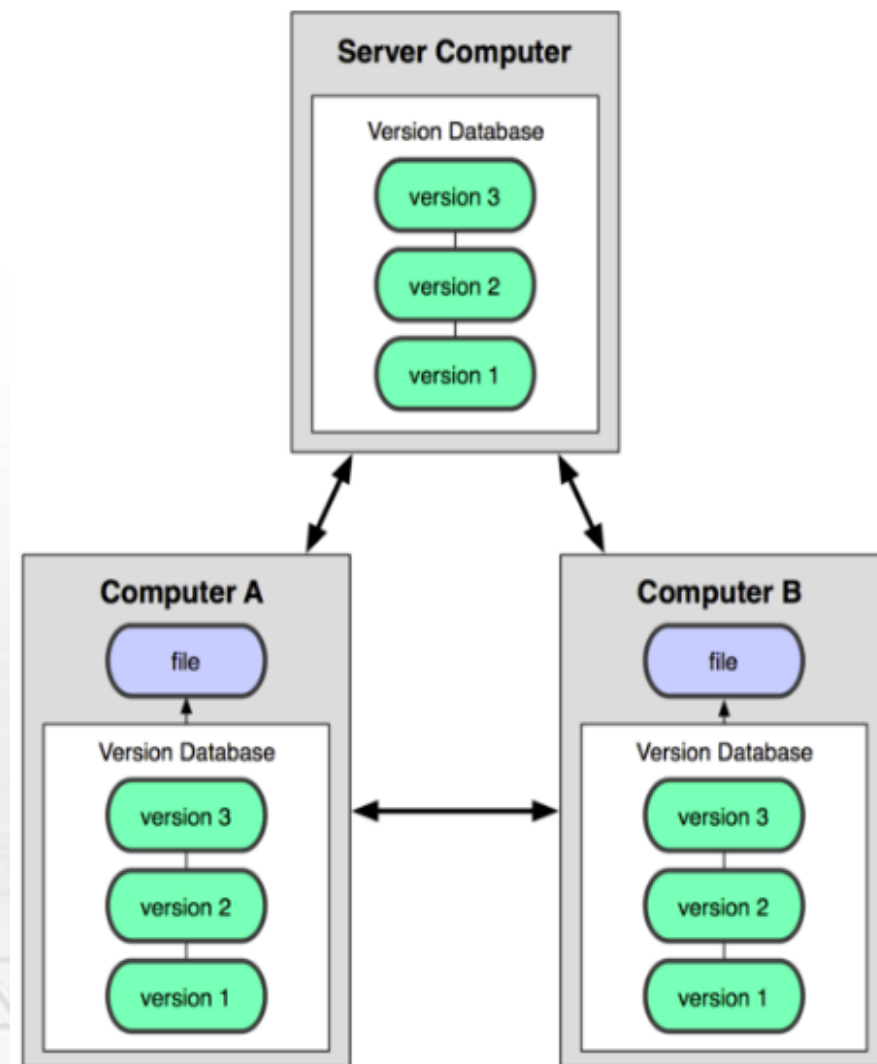
- 版本控制(Revision control)的主要功能
  - 实现跨区域多人协同开发;
  - 追踪和记载一个或者多个文件的历史记录;
  - 组织和保护你的源代码和文档;
  - 统计工作量;
  - 并行开发、提高开发效率;
  - 跟踪记录整个软件的开发过程;
  - 减轻开发人员的负担, 节省时间, 同时降低人为错误。



# Git简介

Git Introduction

- Git是一个开源的分布式版本控制系统，可以有效、高速地处理从很小到非常大的**项目版本管理**，通常有两个主要用途：代码备份和代码版本控制。
- Git 采用了**分布式版本库**的方式
  - 所有版本信息仓库全部同步到本地的每个用户，可以在本地查看所有版本历史。
  - 每个用户都存有所有的版本数据，只要有一个用户的设备正常，就可以恢复所有的数据。
- git官网: <https://git-scm.com/>
- git命令手册: <https://git-scm.com/docs>





# Gitlab简介

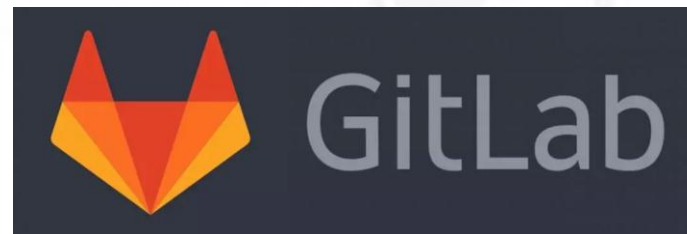
Gitlab Introduction



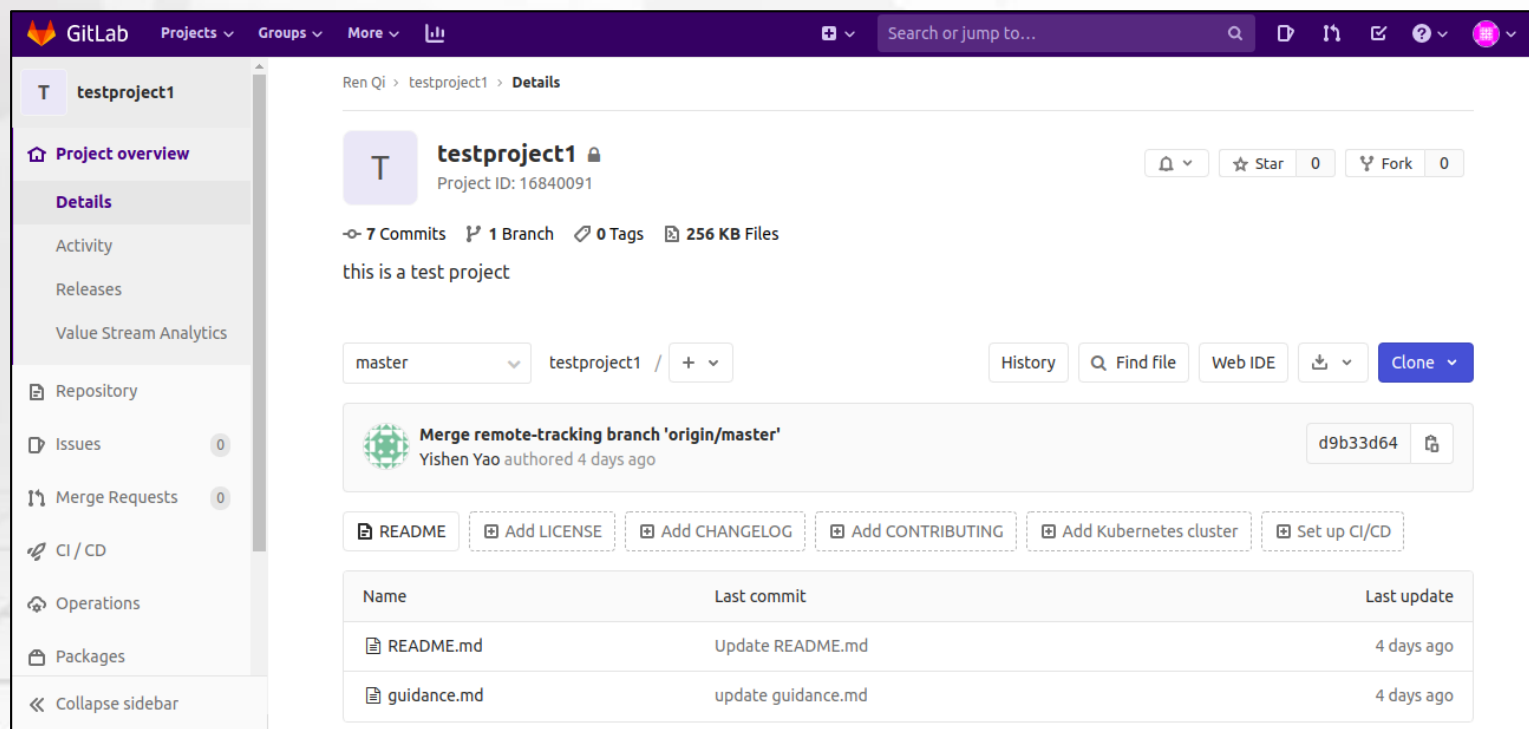
# Gitlab简介

Gitlab 是一个用于仓库管理系统的开源项目，使用Git作为代码管理工具，并在此基础上搭建起来的web服务。

Gitlab拥有与Github类似的功能，不同在于Gitlab创建private仓库不需收费。



Gitlab: <https://gitlab.com/>





# Gitlab使用

Gitlab use

# Gitlab使用

## 1、注册、登录

网址: [https://gitlab.com/users/sign\\_in](https://gitlab.com/users/sign_in)

The screenshot shows the GitLab.com sign-in page. The browser tabs are 'gitlab - 国内版 Bing' and 'Sign in · GitLab'. The address bar shows 'gitlab.com/users/sign\_in'. The page content includes the GitLab logo, a description of the service, a list of links, and a sign-in form. The form has fields for 'Username or email' and 'Password', a 'Remember me' checkbox, and a 'Forgot your password?' link. A green 'Sign in' button is present. Below the form, there is a link 'Register now' highlighted with a red box and an arrow pointing to it with the text '注册(需翻墙)'. At the bottom, there is a 'Sign in with' section with buttons for Google, GitHub, Twitter, Bitbucket, and Salesforce. The GitHub button is highlighted with a red box and an arrow pointing to it with the text '通过github账号登录(无需翻墙)'.

gitlab - 国内版 Bing x Sign in · GitLab x +

gitlab.com/users/sign\_in

应用 ubuntu 测试系统 college 版本管理

## GitLab.com

GitLab.com offers free unlimited (private) repositories and unlimited collaborators.

- [Explore projects on GitLab.com](#) (no login needed)
- [More information about GitLab.com](#)
- [GitLab.com Support Forum](#)
- [GitLab Homepage](#)

By signing up for and by signing in to this service you accept our:

- [Privacy policy](#)
- [GitLab.com Terms.](#)

Username or email

Password

☐ Remember me [Forgot your password?](#)

Sign in

Don't have an account yet? [Register now](#)

Sign in with

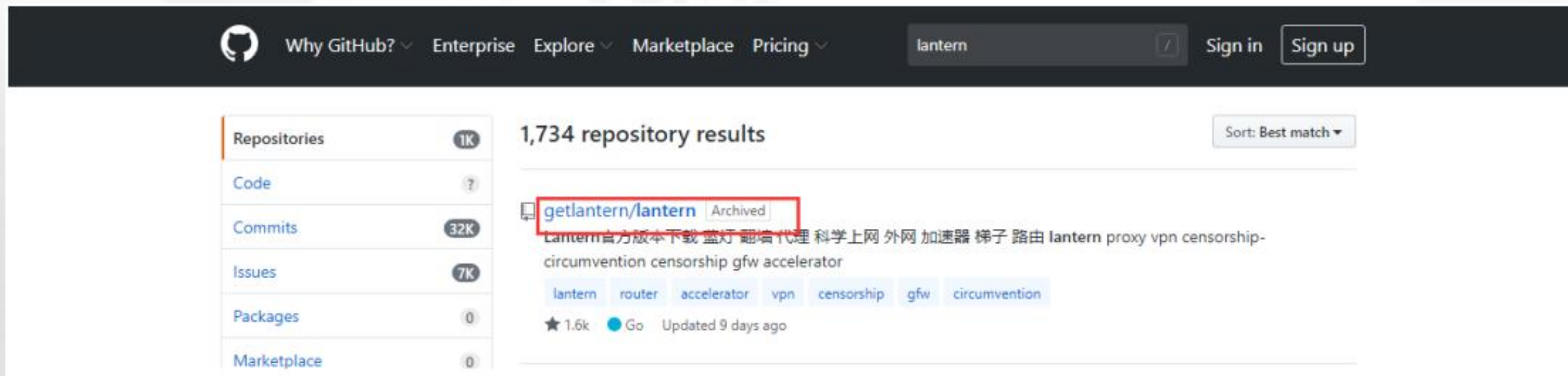
Google GitHub Twitter Bitbucket Salesforce

注册(需翻墙)

通过github账号登录(无需翻墙)

## Gitlab使用

首先我们需要一个翻墙工具，上 Github（<https://github.com/>）搜索 **lantern**，找到下面这个点进去。



## Gitlab使用

在README.md 文档中找到对应自己电脑操作系统的版本并下载、安装（**安装根据默认设置安装即可**）。

📖 README.md

### 蓝灯(Lantern)最新版本下载 版本5.8.3 Download Lantern Version 5.8.3

[Windows7及以上](#) [Windows 7+](#) [备用地址1](#) [Alternative address1](#) [备用地址2](#) [Alternative address2](#)

[安卓版\(4.1+\)](#) [Android\(4.1+\)](#) [备用地址1](#) [Alternative address1](#) [备用地址2](#) [Alternative address2](#) [Google Play](#)

[iOS版本\(12.1+\)](#) [iOS\(12.1+\)](#) [iOS版安装使用教程](#)

[苹果电脑版\(OS X 10.11 El Capitan及以上\)](#) [备用地址1](#) [Alternative address1](#) [备用地址2](#) [Alternative address2](#)

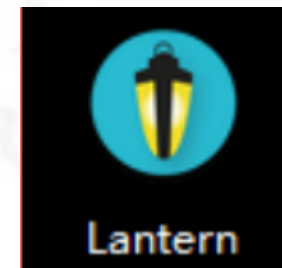
[Ubuntu 14.04及以上](#) [32位系统](#) [备用地址](#) [Alternative address](#) [64位系统](#) [备用地址](#) [Alternative address](#)

请大家收藏本页面，方便日后下载新版。 Bookmark this page to download the latest versions in the future.

使用遇到问题，请阅读[蓝灯常见问题解决办法](#) When you have a problem when using Lantern, please refer to [FAQ](#)

## Gitlab使用

安装完成后在系统启动栏找到 lantern 图标，点击启动工具。  
启动工具后会打开一个网页，网页右下角会显示连接状态。



☰ 蓝灯免费版

 蓝灯与隐币是合作伙伴，购买蓝灯可免费获赠加密货币隐币！[点击购买](#)

  
**HK**  
服务器位置  
升级至更快的服务器

  
**00:37:52**  
距离今天隐币发放剩余时间  
移动鼠标了解更多细节

  
**100%**  
数据用量 - 现在升级

**蓝灯专业版**

速度更快。更稳定。无流量限制。免费加密货币隐币

[立即升级](#)

网上打折购买了专业版激活码？  
[点这里使用激活码升级，免费开抢隐币](#)

Lantern 5.8.3 (20200212.220109)

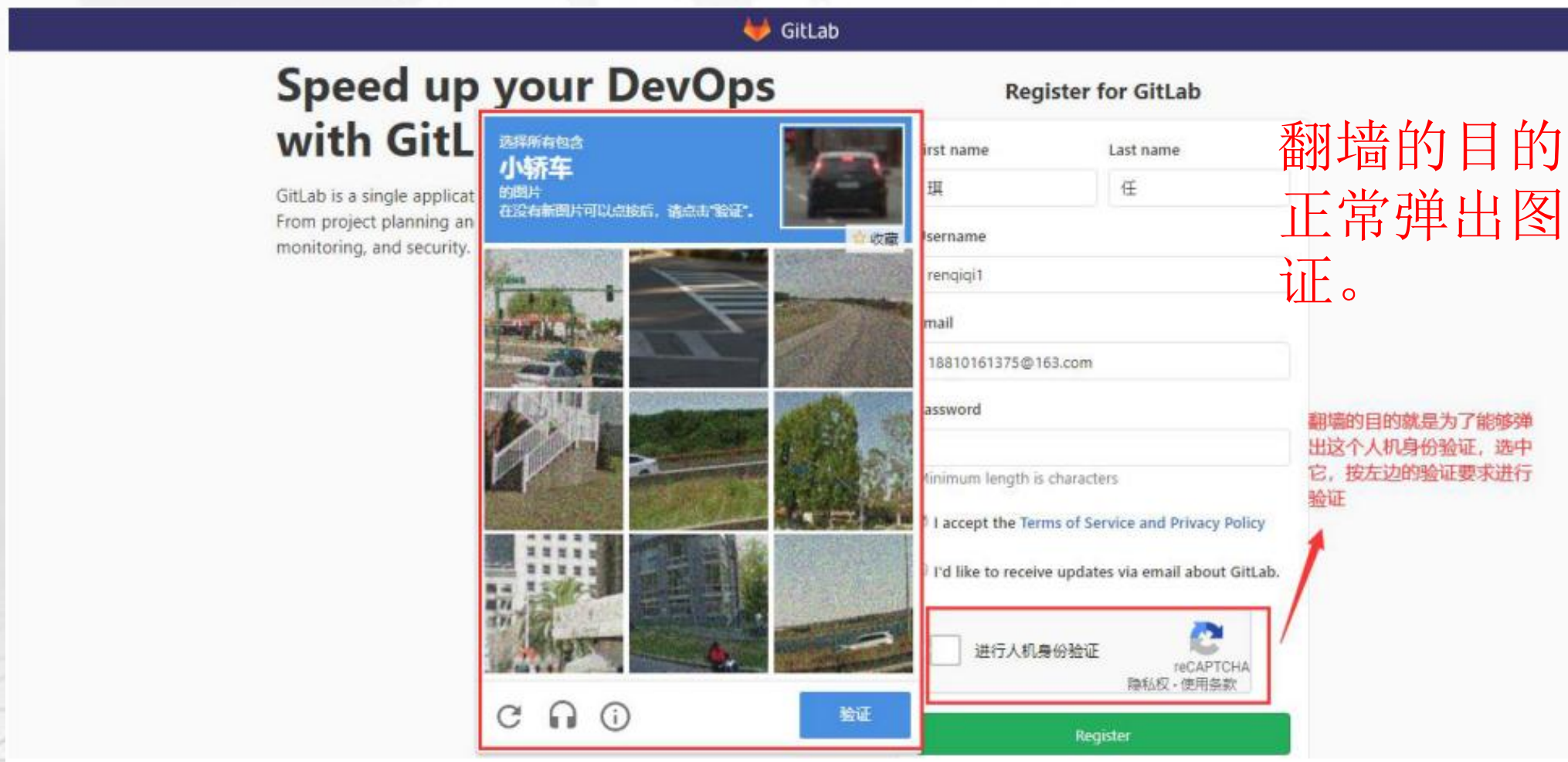
连接状态

 已连接



## Gitlab使用

然后再回到 Gitlab 注册界面，输入注册信息。



The screenshot shows the GitLab registration page. The header has the GitLab logo and the text "Speed up your DevOps with GitLab". The main heading is "Register for GitLab". The registration form includes fields for "first name" (琪), "Last name" (任), "username" (renqiqi1), "email" (18810161375@163.com), and "password". There are checkboxes for "I accept the Terms of Service and Privacy Policy" and "I'd like to receive updates via email about GitLab.". A red box highlights the reCAPTCHA section, which includes a grid of images for selection and a "验证" (Verify) button. The reCAPTCHA text says "选择所有包含小轿车的图片" (Select all images containing small cars) and "在没有新图片可以点击后，请点击'验证'。" (After no new images can be clicked, click 'Verify'). A red arrow points from the text "翻墙的目的就是为了能够弹出这个人机身份验证，选中它，按左边的验证要求进行验证" (The purpose of using a proxy is to be able to pop up this human-machine identity verification, select it, and follow the verification requirements on the left) to the reCAPTCHA section.

GitLab

### Speed up your DevOps with GitLab

GitLab is a single application for managing your project's development lifecycle, from project planning and collaboration to monitoring and security.

### Register for GitLab

first name Last name  
琪 任

username  
renqiqi1

email  
18810161375@163.com

password  
Minimum length is 8 characters

☒ I accept the [Terms of Service and Privacy Policy](#)

☐ I'd like to receive updates via email about GitLab.

☐ 进行人机身份验证

reCAPTCHA  
隐私权 · 使用条款

验证

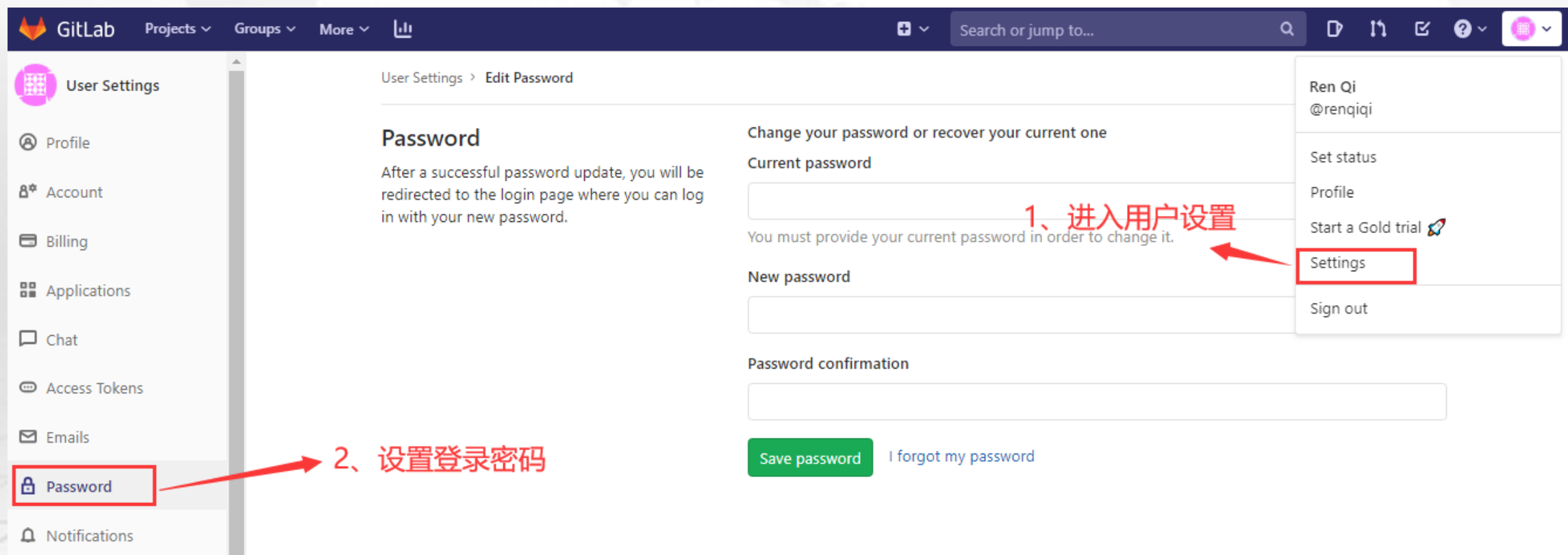
Register

翻墙的目的就是为了能正常弹出图中的身份验证。

翻墙的目的就是为了能够弹出这个人机身份验证，选中它，按左边的验证要求进行验证

# Gitlab使用

如果是用Github登录的，可以在个人设置里设置一下 Gitlab 平台的密码，这样就不用每次通过 Github 登录了，可以直接在 Gitlab 登录主页输入用户名和密码登录。



The screenshot shows the GitLab web interface. The top navigation bar includes the GitLab logo, 'Projects', 'Groups', and 'More' dropdowns, along with a search bar and user profile icon. The left sidebar contains a 'User Settings' menu with options: Profile, Account, Billing, Applications, Chat, Access Tokens, Emails, Password (highlighted with a red box and an arrow pointing to the text '2、设置登录密码'), and Notifications. The main content area is titled 'User Settings > Edit Password' and 'Password'. It contains a message: 'After a successful password update, you will be redirected to the login page where you can log in with your new password.' Below this are three input fields: 'Current password' (with a red arrow pointing to it from the text '1、进入用户设置'), 'New password', and 'Password confirmation'. At the bottom of the form is a green 'Save password' button and a blue link 'I forgot my password'. On the right side, a user profile dropdown menu is open, showing the user's name 'Ren Qi', email '@renqiqi', and options: 'Set status', 'Profile', 'Start a Gold trial', 'Settings' (highlighted with a red box and an arrow pointing to it from the text '1、进入用户设置'), and 'Sign out'.

GitLab Projects Groups More

User Settings

Profile

Account

Billing

Applications

Chat

Access Tokens

Emails

**Password**

Notifications

User Settings > Edit Password

**Password**

After a successful password update, you will be redirected to the login page where you can log in with your new password.

Change your password or recover your current one

Current password

You must provide your current password in order to change it.

New password

Password confirmation

Save password I forgot my password

Ren Qi  
@renqiqi

Set status

Profile

Start a Gold trial

**Settings**

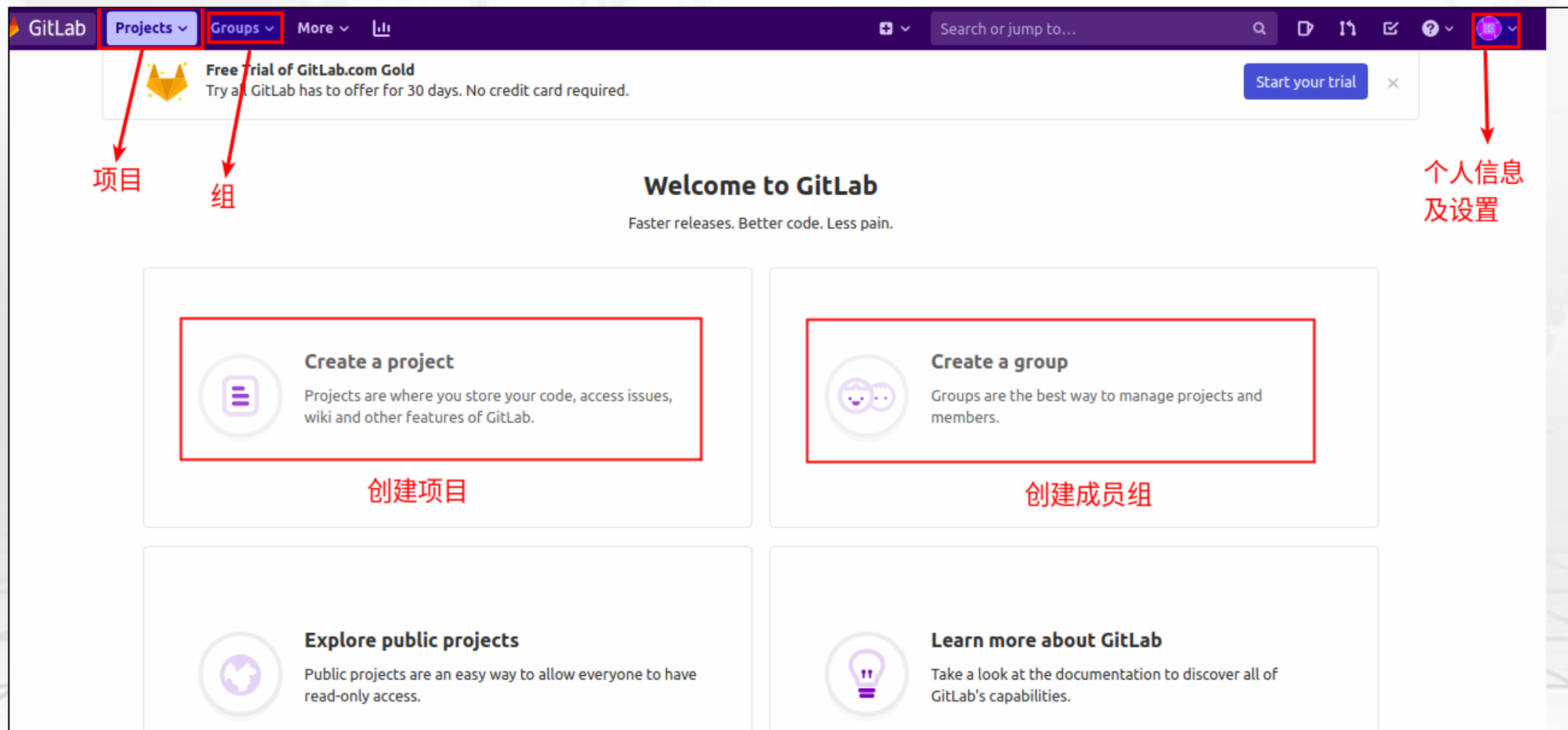
Sign out

1、进入用户设置

2、设置登录密码



## 登录后，进入Gitlab主页



## 2、创建项目

- 一个项目通常由一位项目负责人和多位开发人员组成。
- 一般由负责人创建项目并管理成员，把控项目进度。

The screenshot shows the 'Blank project' creation form in GitLab. It includes fields for 'Project name' (testproject1), 'Project URL' (https://gitlab.com/renqiqi/), 'Project slug' (testproject1), and 'Project description (optional)' (this is a test project). The 'Visibility Level' is set to 'Private'. The 'Initialize repository with a README' checkbox is checked. Red arrows point from Chinese labels to the corresponding form elements: '项目名称' to the Project name field, '项目描述' to the Project description field, '项目是否可见 (private和public都可)' to the Visibility Level section, and '创建时附带readme.md文档 (注意勾上)' to the Initialize repository with a README checkbox.

Blank project	Create from template	Import project	CI/CD for external repo
<b>Project name</b> testproject1			
<b>Project URL</b> https://gitlab.com/renqiqi/		<b>Project slug</b> testproject1	
Want to house several dependent projects under the same namespace? <a href="#">Create a group.</a>			
<b>Project description (optional)</b> this is a test project			
<b>Visibility Level</b> <input checked="" type="radio"/> Private Project access must be granted explicitly to each user. <input type="radio"/> Public The project can be accessed without any authentication.			
<input checked="" type="checkbox"/> <b>Initialize repository with a README</b> Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.			
<b>Create project</b>			<b>Cancel</b>

## 3、添加项目成员

创建好项目后，进入项目主页，在左方项目属性栏找到**settings**的子栏**members**，添加项目成员。

The screenshot shows the GitLab web interface for managing project members. The left sidebar contains navigation links: Project overview, Repository, Issues, Merge Requests, CI / CD, Operations, Packages, Wiki, Snippets, Settings (highlighted with a red box), and Members (highlighted with a red box). The main content area is titled 'Project members' and includes a sub-header 'You can invite a new member to testproject1 or invite another group.' Below this are two tabs: 'Invite member' (active) and 'Invite group'. The 'Invite member' tab contains three input fields: 'GitLab member or Email address' (with a red arrow pointing to it and the annotation '所要邀请的成员的gitlab账号名或邮箱'), 'Choose a role permission' (with a red arrow pointing to it and the annotation '成员角色: guest、reporter、developer、maintainer 一般开发给developer角色权限'), and 'Access expiration date' (with a red arrow pointing to it and the annotation '访问到期时间 (置空就行)'). At the bottom of the form are 'Invite' and 'Import' buttons.

GitLab member or Email address 所要邀请的成员的gitlab账号名或邮箱

Choose a role permission 成员角色: guest、reporter、developer、maintainer  
一般开发给developer角色权限

Access expiration date 访问到期时间 (置空就行)

## 4、分支权限设置

项目负责人要修改master分支保护状态以允许developer角色用户能够成功提交代码文件。先点击**settings**，选择子栏**repository**，在页面中选择**protected branches**项。

**Protected Branches**

Keep stable branches secure and force developers to use merge requests.

By default, protected branches are designed to:

- prevent their creation, if not already created, from everybody except Maintainers
- prevent pushes from everybody except Maintainers
- prevent **anyone** from force pushing to the branch
- prevent **anyone** from deleting the branch

Read more about [protected branches](#) and [project permissions](#).

Protect a branch

Branch:   
Wildcards such as `*-stable` or `production/*` are supported

Allowed to merge:

Allowed to push:

Protect

Roles

- Maintainers
- ☒ Developers + Maintainers
- No one

Branch Allowed to merge

master	default	Maintainers	Developers + ...	Unprotect
--------	---------	-------------	------------------	-----------

将默认的maintainers修改为developers+maintainers

### 5、下载、安装Git

进入git官网(<https://git-scm.com/downloads>), 下载对应操作系统的git工具。  
(注意: 分清下载32bit或64bit)。

- (1) Windows10系统详细安装及配置步骤见链接:  
<https://www.cnblogs.com/xiaoliu66/p/9404963.html>
- (2) Ubuntu系统详细安装步骤:  
安装Git方法如下, 在终端中输入如下命令:  
`sudo apt-get install git`
- (3) Mac OS系统详细安装步骤见链接:  
<https://www.jianshu.com/p/7edb6b838a2e>

安装完成后, 可以通过`git --version`命令查看是否安装成功。

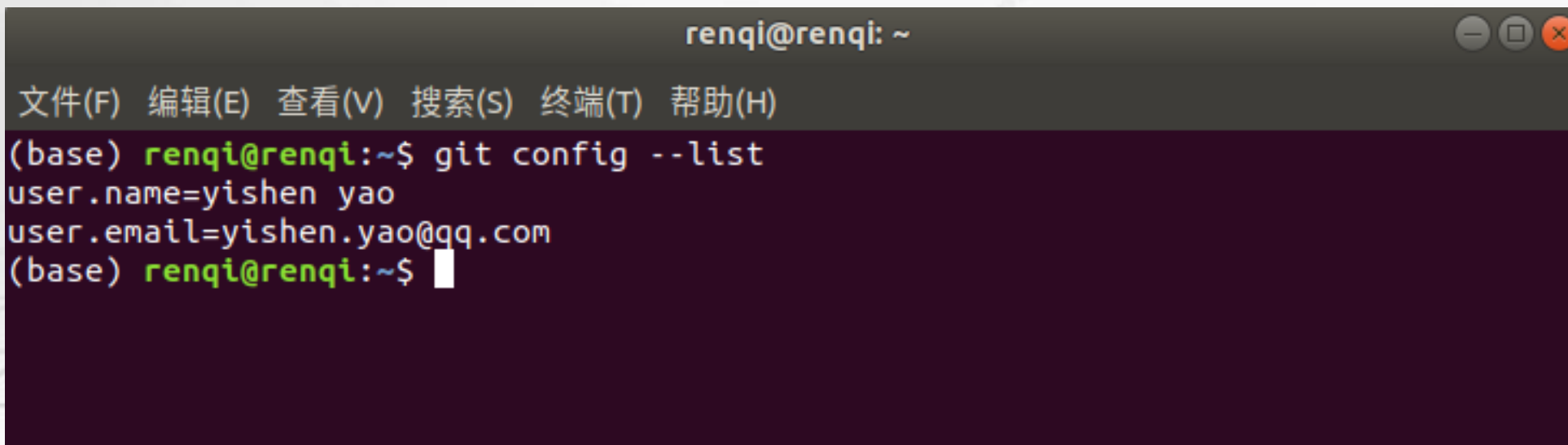
## 6、配置Git

可以保存Git用户名和邮箱，这样就不必在以后的Git命令中再次输入它们。  
在终端或git bash在命令行中配置本地仓库的账号和邮箱：

`git config --global user.name "xxxx"` (任取即可)

`git config --global user.email "xxx@xxx"` (与Gitlab注册邮箱一致)

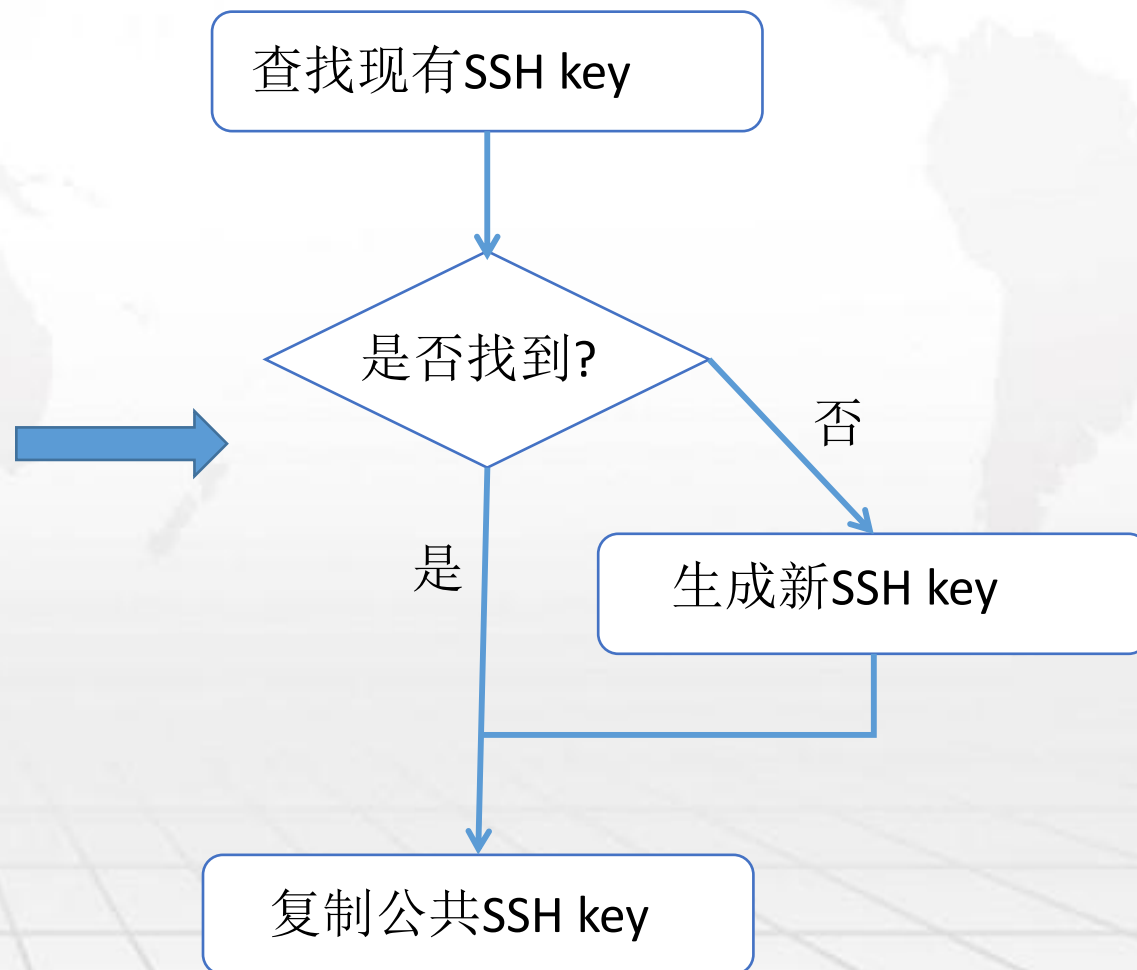
通过`git config --list`可以列出配置信息。

A terminal window titled 'renqi@renqi: ~' with standard window controls. The menu bar shows '文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)'. The command '(base) renqi@renqi:~\$ git config --list' has been executed, resulting in the output 'user.name=yishen yao' and 'user.email=yishen.yao@qq.com'. The prompt '(base) renqi@renqi:~\$' is followed by a cursor.

```
renqi@renqi: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) renqi@renqi:~$ git config --list
user.name=yishen yao
user.email=yishen.yao@qq.com
(base) renqi@renqi:~$
```

## 7、配置SSH key

Git 使用 https 协议，每次 pull、push 都要输入密码，比较麻烦。因此可以使用 SSH 密钥，这样可以省去每次都输密码。配置本地 SSH key，按照右边流程来进行。





### 8、查找现有SSH key

在生成新的SSH密钥对之前，通过Ubuntu上的终端或Windows上的命令提示并运行以下命令来检查系统在默认位置是否已经存在一个我们需要的ssh key。若已存在且生成SSH key的邮箱和Gitlab账号邮箱一致，则可使用此密钥，否则生成新的密钥。

Windows Command Prompt:

```
type %userprofile%\\.ssh\id_rsa.pub
```

Git Bash on Windows/GNU/Linux/macOS/PowerShell:

```
cat ~/.ssh/id_rsa.pub
```



## 9、生成新的SSH key密钥对

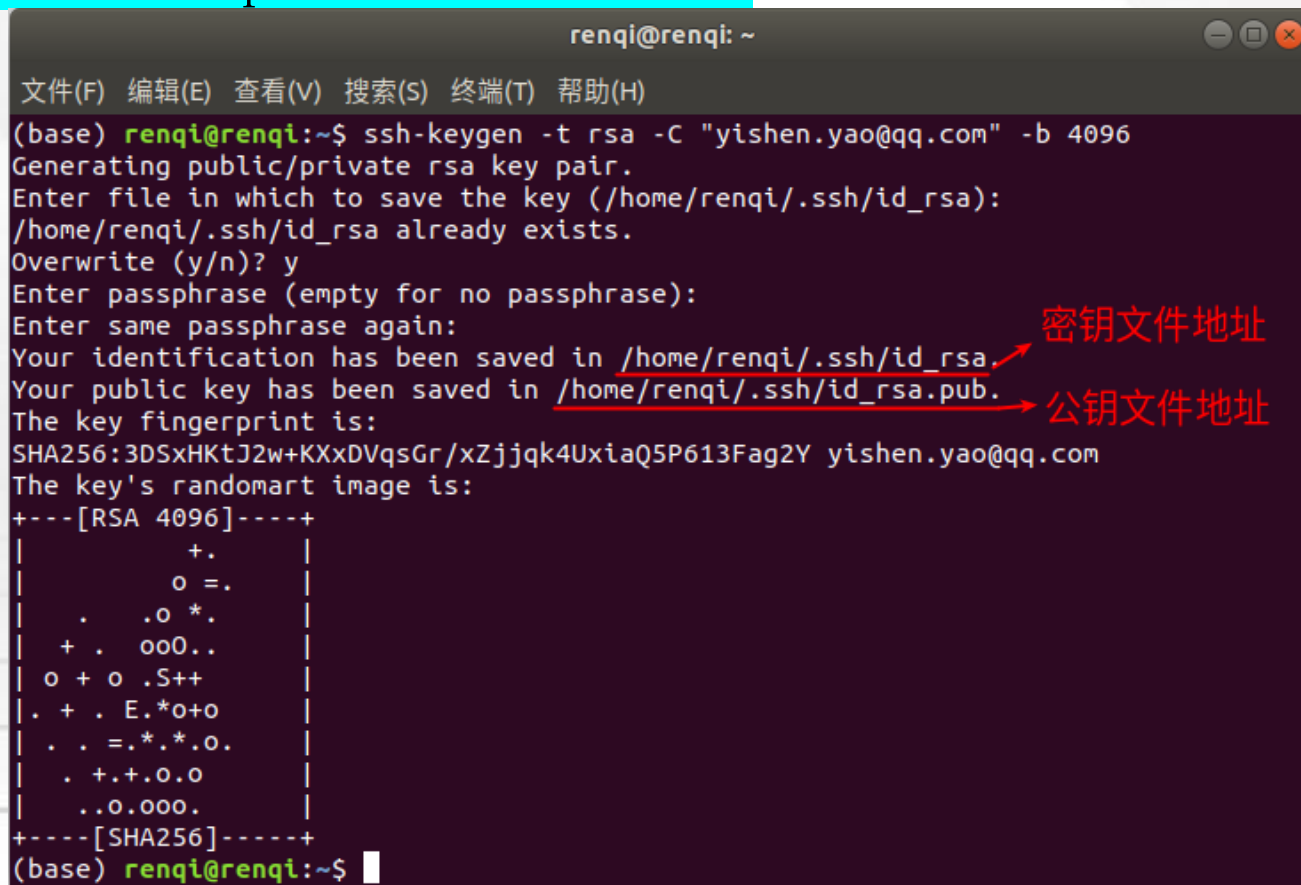
要生成新的SSH密钥对，使用以下命令：

Git Bash on Windows/GNU/Linux/macOS:

```
ssh-keygen -t rsa -C "your.email@example.com" -b 4096
```

(引号内换成你的邮箱地址)

出现右边界面表示公钥和密钥生成成功，信息中包含密钥文件地址和公钥文件地址，我们所需要的就是公钥文件。



```
renqi@renqi: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
(base) renqi@renqi:~$ ssh-keygen -t rsa -C "yishen.yao@qq.com" -b 4096  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/renqi/.ssh/id_rsa):  
/home/renqi/.ssh/id_rsa already exists.  
Overwrite (y/n)? y  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/renqi/.ssh/id_rsa. 密钥文件地址  
Your public key has been saved in /home/renqi/.ssh/id_rsa.pub. 公钥文件地址  
The key fingerprint is:  
SHA256:3DSxHKtJ2w+KXXdVqsGr/xZjjqk4UxiaQ5P613Fag2Y yishen.yao@qq.com  
The key's randomart image is:  
+----[RSA 4096]-----+  
|          +.          |  
|          o =.        |  
|         .o *.        |  
|        +. oo0..       |  
|       o + o .S++      |  
|      . + . E.*o+o      |  
|     . . =.*.*.o.       |  
|    . +.+ .o.o         |  
|   ..o.ooo.            |  
+----[SHA256]-----+  
(base) renqi@renqi:~$
```

### 10、复制公共SSH密钥

将公共SSH密钥复制到剪贴板，复制所用命令如下：

macOS: `pbcopy < ~/.ssh/id_rsa.pub`

GNU/Linux (requires the xclip package): `xclip -sel clip < ~/.ssh/id_rsa.pub`

Windows Command Line: `type %userprofile%\ssh\id_rsa.pub | clip`

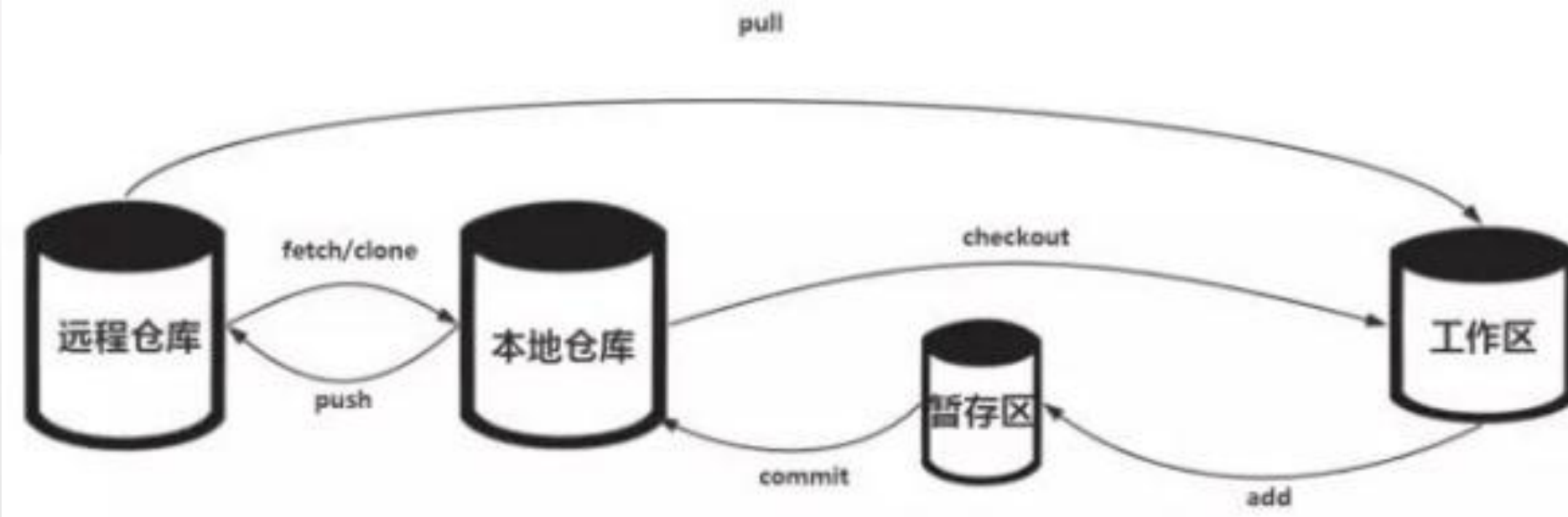
Git Bash on Windows / Windows PowerShell: `cat ~/.ssh/id_rsa.pub | clip`

若使用以上命令无法复制公钥，可根据公钥文件地址找到公钥文件手动拷贝。

You won't be able to pull or push project code via SSH until you **add an SSH key** to your profile

点击黄条中的add an SSH key，进入ssh key配置界面，将剪贴板中的内容添加到gitlab的SSH key里面。

# Gitlab使用



**工作区Workspace:** 程序员进行开发(改动)的地方, 是当前看到的。

**说明:** 任何对象都是在工作区中诞生和被修改

**暂存区Index / Stage:** .git目录下的index文件, 暂存区会记录git add添加文件的相关信息(文件名、大小、timestamp...), 不保存文件实体, 通过id指向每个文件实体。可以使用git status查看暂存区的状态。

**说明:** 任何修改都是从进入暂存区才开始被版本控制;

## Gitlab使用

本地仓库**Repository**: 保存了对象被提交过的各个版本，比起工作区和暂存区的内容，它要更旧一些。`git commit` 后同步index的目录树到本地仓库，自动初始化为本地仓库，同时它会新建`.git`目录方便从下一步通过`git push` 同步本地仓库与远程仓库。

**说明1**: 只有把修改提交到本地仓库，该修改才能在仓库中留下痕迹；

**说明2**: 可以在任何地方新建本地仓库，只需要在目标目录下执行 “`git init`” 指令，就会将此目录自动初始化为本地仓库，同时它会新建 “`.git`” 目录。

远程仓库**Remote**: 内容可能被分布在多个地点的处于协作关系的本地仓库修改，因此它可能与本地仓库同步，也可能不同步，但是它的内容是最旧的。

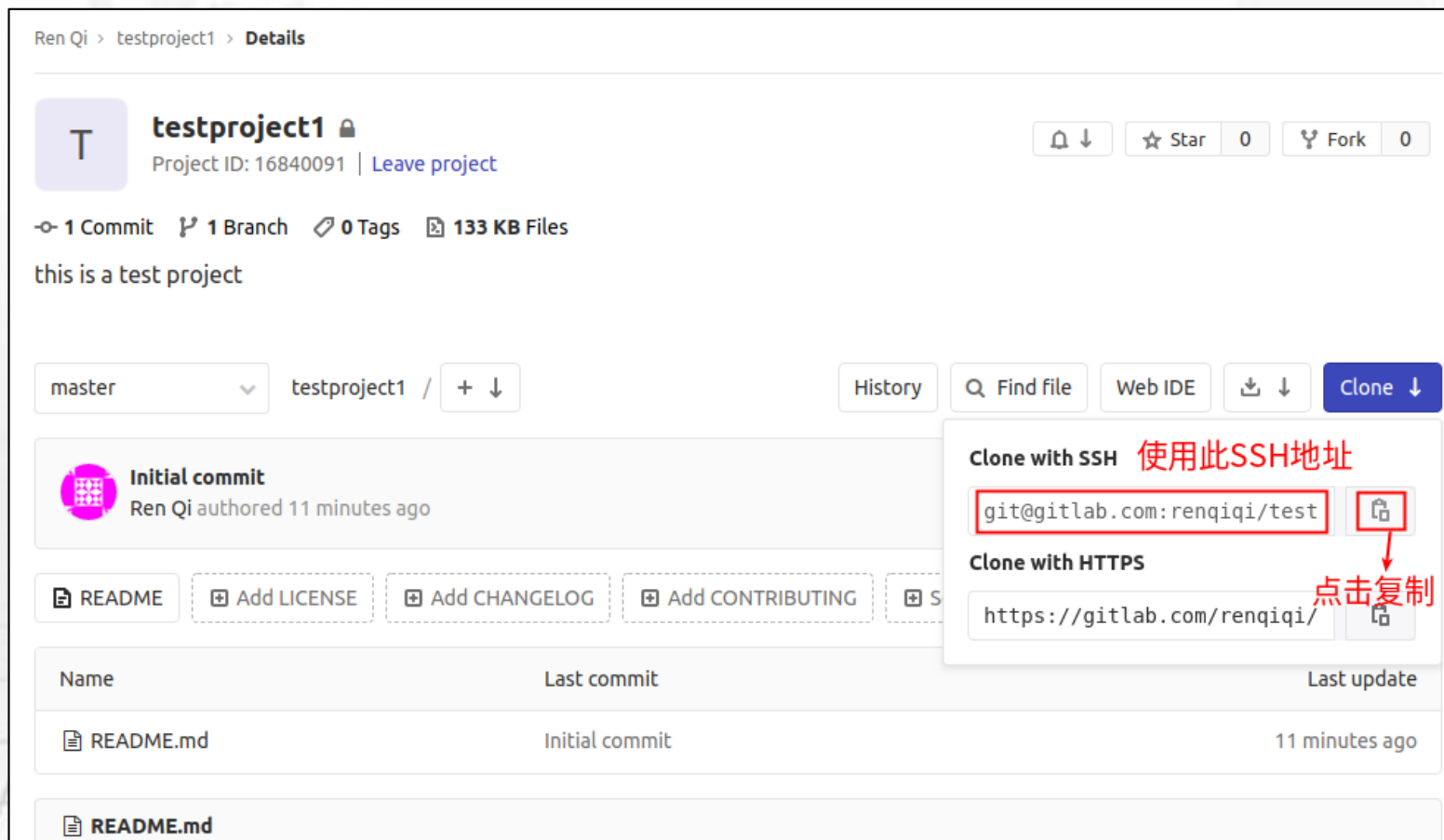
**说明**: 与协作者分享本地的修改，可以把它们push到远程仓库来共享。

## 11、克隆项目到本地

在developer角色用户第一次向项目中提交代码和文档前，需要将项目克隆至本地。


`git clone +地址`




地址使用右图的SSH地址，  
执行完后会在本地得到一个以项目命名的文件夹(工作区)。







The screenshot shows the GitLab interface for a project named 'testproject1'. The project details include the name, ID (16840091), and a 'Leave project' link. It shows 1 commit, 1 branch, 0 tags, and 133 KB of files. The project description is 'this is a test project'. The 'Clone' button is highlighted, and a dropdown menu is open showing two options: 'Clone with SSH' and 'Clone with HTTPS'. The SSH address is 'git@gitlab.com:renqiqi/test' and the HTTPS address is 'https://gitlab.com/renqiqi/'. A red box highlights the SSH address, and a red arrow points to the 'Copy' icon next to it with the text '点击复制' (Click to copy). The text '使用此SSH地址' (Use this SSH address) is also present.

Ren Qi > testproject1 > Details

**testproject1**   
Project ID: 16840091 | [Leave project](#)


  Star 0  Fork 0






 1 Commit  1 Branch  0 Tags  133 KB Files


this is a test project


master testproject1 / + ↓


History Find file Web IDE Clone ↓


 **Initial commit**  
Ren Qi authored 11 minutes ago

 README  Add LICENSE  Add CHANGELOG  Add CONTRIBUTING  Add SECURITY

Name	Last commit	Last update
 README.md	Initial commit	11 minutes ago

 README.md

Clone with SSH 使用此SSH地址  
git@gitlab.com:renqiqi/test 

Clone with HTTPS  
https://gitlab.com/renqiqi/ 

点击复制



## 12、提交代码、文档

将需要提交的代码或文档复制到上一步得到的仓库中，如下图的guidance.md文档。



然后在代码文件所在的目录打开终端，输入如下命令提交代码：

- 1、`git add filename` (逐个添加文件)  
`git add -A` (添加当前目录中的所有文件)  
`git add .` (添加当前目录中的所有文件更改)
- 2、`git commit -m "提交信息"`
- 3、`git push -u origin master` (向远处仓库推送代码)

## 13、拉取远程仓库

- 多人开发项目中，所有developer都可以向gitlab仓库提交代码或文档，所以不同的developer本地仓库的文件进度不同会引发冲突，导致无法向gitlab远程仓库提交更新。
- 这时，先从远程仓库fetch到更新并和本地仓库合并，再进行git push操作。
- 执行的两句命令如下：

git fetch origin

git merge origin/master

```
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git fetch origin
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
展开对象中: 100% (4/4), 完成.
来自 gitlab.com:renqi/testproject1
    68050ed..0df422c  master    -> origin/master
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git merge origin/master
Merge made by the 'recursive' strategy.
 README.md | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$
```

## 14、本地修改文件后提交到远程仓库

`git status` 查看git是否有修改内容需要提交  
`git add` 指向需要提交的内容文件  
`git commit` 提交到本地库  
`git push origin master` 提交到远程仓库

```
renqi@renqi: ~/文档/gitlab使用指南/test/testproject1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

无文件要提交，干净的工作区
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

尚未暂存以备提交的变更：
  (使用 "git add <文件>..." 更新要提交的内容)
  (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

修改:   guidance.md

修改尚未加入提交 (使用 "git add" 和/或 "git commit -a")
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$
```

修改本地代码后，使用 `git status` 查看本地仓库状态，此时我们修改了文档 `guidance.md` 的内容，能够看到提示修改了的文档，需要将这修改后的文档重新提交到gitlab平台上。



# Gitlab使用

`git add path/guidance.md`

即 `git add` 文件路径

```
renqi@renqi: ~/文档/gitlab使用指南/test/testproject1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

无文件要提交，干净的工作区
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

尚未暂存以备提交的变更:
  (使用 "git add <文件>..." 更新要提交的内容)
  (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

        修改:        guidance.md



修改尚未加入提交 (使用 "git add" 和/或 "git commit -a")
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git add guidance.md
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$
```

## Gitlab使用

### git commit -m "提交信息"

```
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git add guidance.md
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$ git commit -m "update guidance.md"
[master 68050ed] update guidance.md
 1 file changed, 10 insertions(+), 1 deletion(-)
(base) renqi@renqi:~/文档/gitlab使用指南/test/testproject1$
```

**git push origin master** 推送到gitlab平台，修改更新成功，此时在gitlab平台上就能看到guidance.md文档在3分钟前提交了更新。

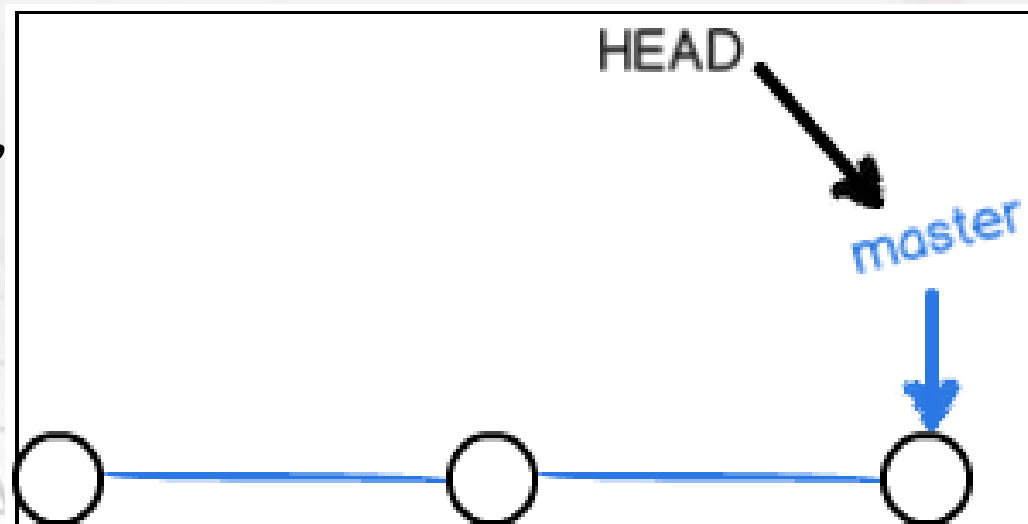
Name	Last commit	Last update
 README.md	Initial commit	13 hours ago
 guidance.md	update guidance.md	3 minutes ago

## 15、分支

Gitlab存储库的`master`分支应始终包含有效且稳定的代码，同时还希望备份一些当前正在处理的代码，但这些代码并不完全稳定。

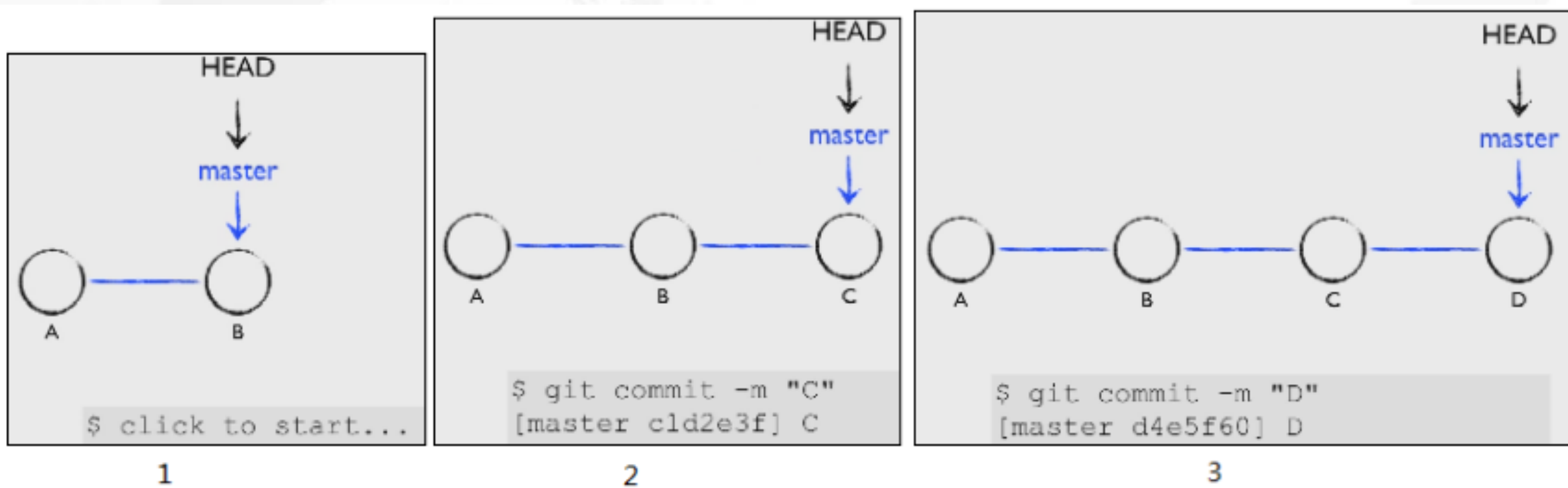
分支是可以在不影响`master`分支的情况下处理代码的单独副本。首次创建分支时，将以新名称创建`master`分支的完整克隆。然后，可以独立地在此新分支中修改代码，包括提交文件等。一旦你的新功能已完全集成并且代码稳定，就可以将其合并到`master`分支中！

一开始的时候，`master`分支是一条线，Git用`master`指向最新的提交，再用`HEAD`指向`master`，就能确定当前分支，以及当前分支的提交点：



# Gitlab使用

每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长：



此时master分支指向B，  
当前分支也为B。

提交C，master分支指向  
最新的一次提交，当前分  
支为C。

提交D，master分支指向  
最新的一次提交，当前分  
支为D。

# Gitlab使用

当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：

## git checkout -b 分支名

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

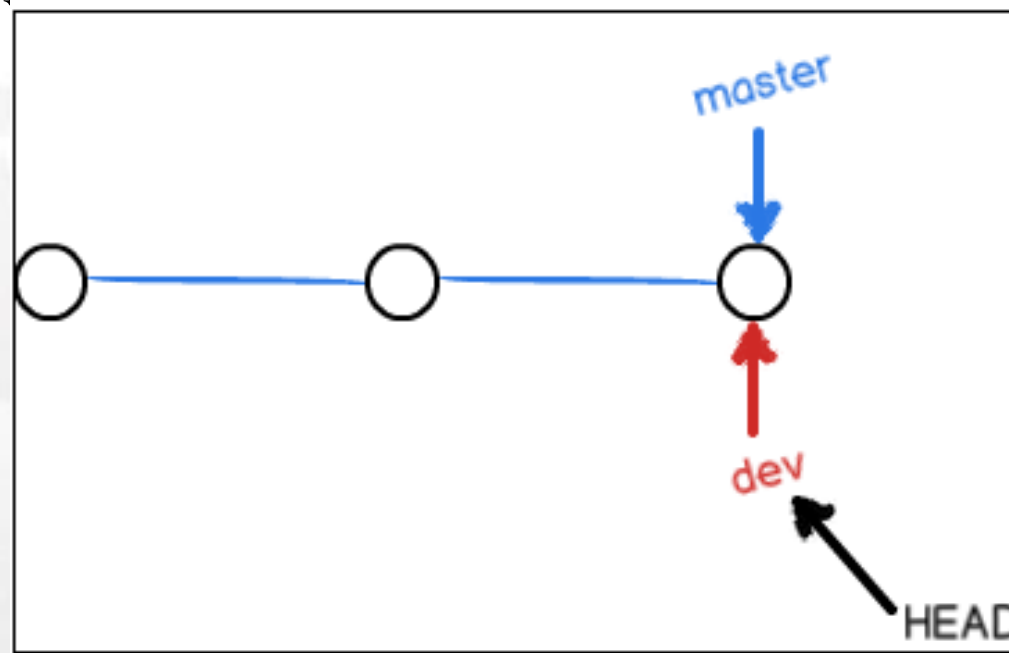
git branch 分支名 (创建分支)

git checkout 分支名 (切换分支)

用git branch命令查看当前分支：

## git branch

git branch命令会列出所有分支，当前分支前面会标一个\*号。



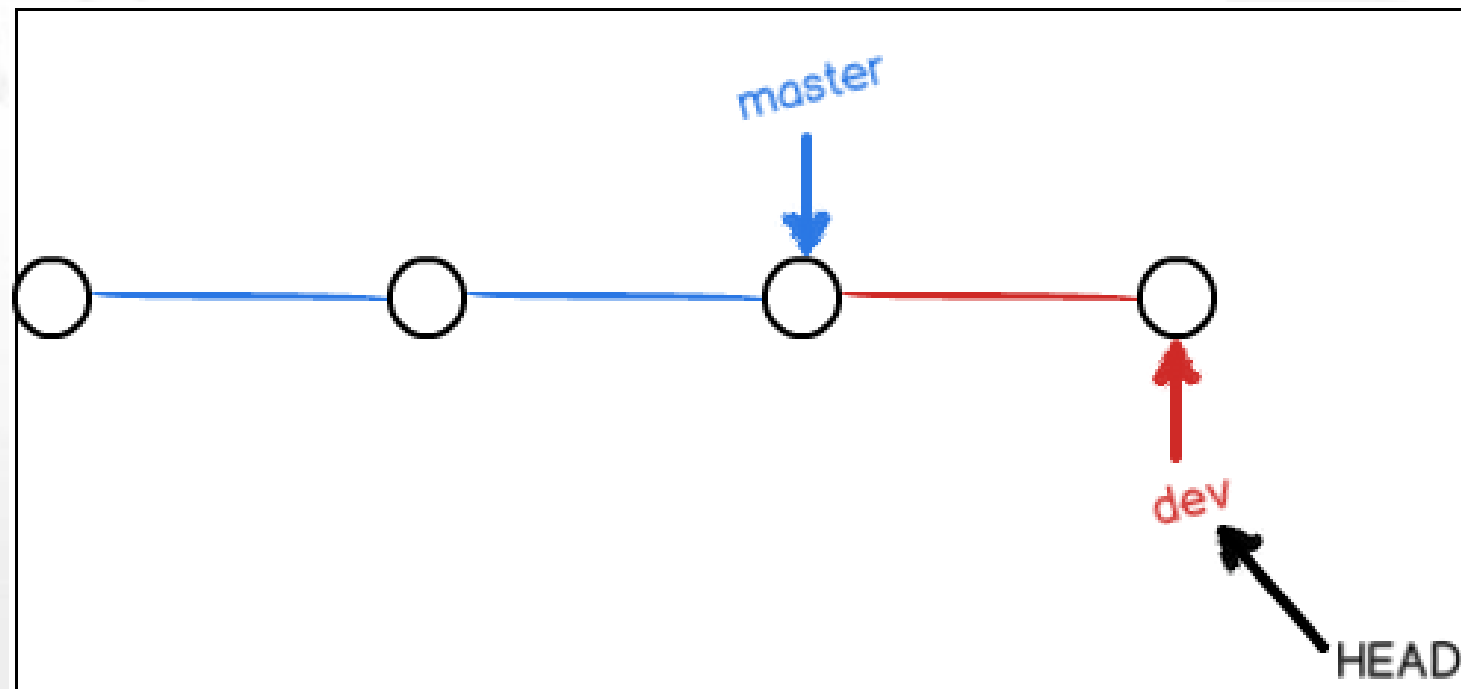
```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git checkout -b dev
Switched to a new branch 'dev'

18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (dev)
$ git branch
* dev
  master
```

## Gitlab使用

从现在开始，对工作区的修改和提交就是针对dev分支了，比如在工作区新建一个dev.txt文档后，dev指针往前移动一步，而master指针不变：

```
git add dev.txt  
git commit -m "create dev.txt"
```



在dev分支下，能够看到工作区中有dev.txt文档。

名称	修改日期	类型	大小
dev.txt	2020/3/3 13:31	文本文档	1 KB
README.md	2020/3/3 12:52	MD 文件	1 KB

```
18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (dev)  
$ git add dev分支.txt  
  
18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (dev)  
$ git commit -m "create dev分支.txt"  
[dev 61c7dd1] create dev分支.txt  
1 file changed, 1 insertion(+)  
create mode 100644 "dev\345\210\206\346\224\257.txt"
```



## Gitlab使用

现在，dev分支的工作完成，我们就可以切换回master分支：

git checkout master

```
18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (master)
$ :
```

切换回master分支后，在本地工作区看不到dev.txt文档(**dev.txt文档不在master分支进度中**)。

名称	修改日期	类型	大小
 README.md	2020/3/3 14:09	MD 文件	1 KB

# Gitlab使用

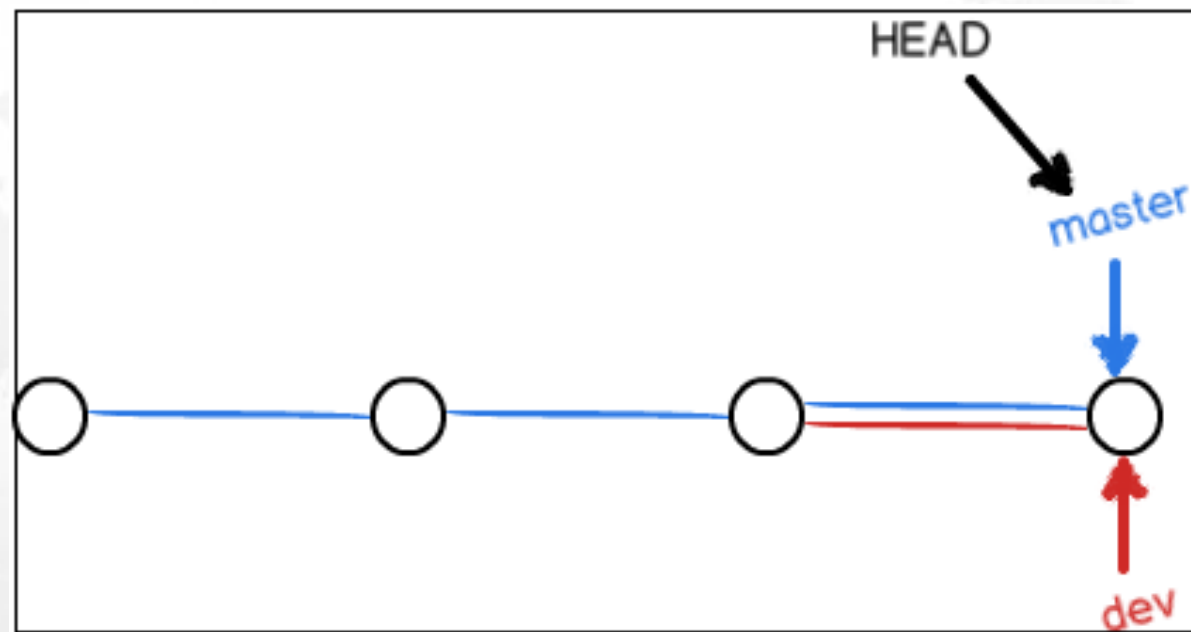
假如我们在dev上的工作完成了，就可以把dev合并到master上。

`git merge dev` (dev为要合并的分支名)

`git merge`命令用于合并指定分支到当前分支。

```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git merge dev
Updating 8a624bf..76fb2b1
Fast-forward
 dev.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 dev.txt

18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$
```



可以看到在master分支下，工作区出现了dev.txt文档，合并完毕。

名称	修改日期	类型	大小
 dev.txt	2020/3/3 14:15	文本文档	1 KB
 README.md	2020/3/3 14:09	MD 文件	1 KB



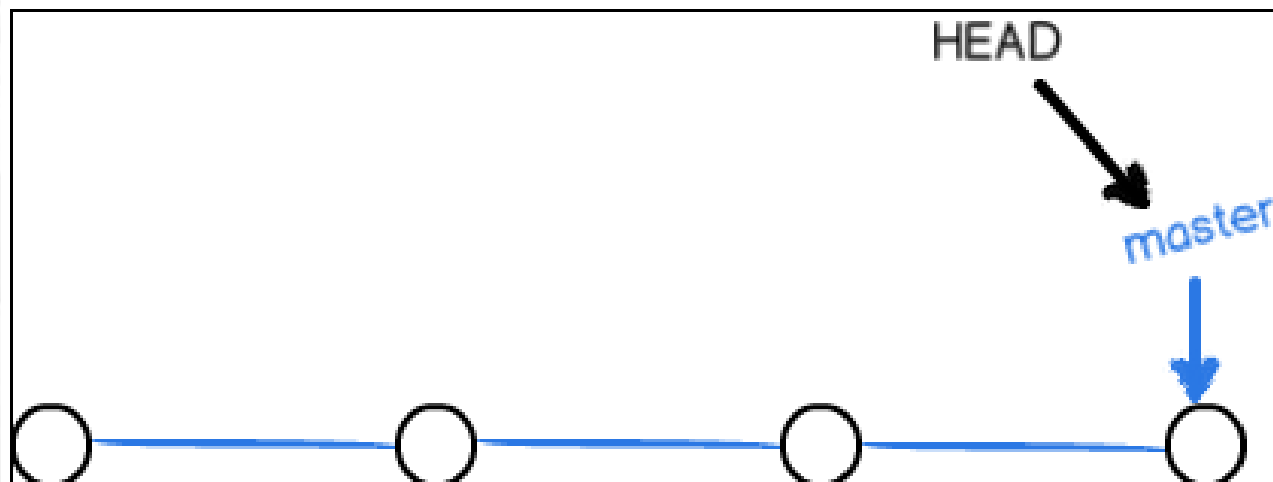
## Gitlab使用

合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：

```
git branch -d dev
```

删除后，查看branch，就只剩下master分支了：

```
git branch
```



```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git branch -d dev
Deleted branch dev (was 76fb2b1).

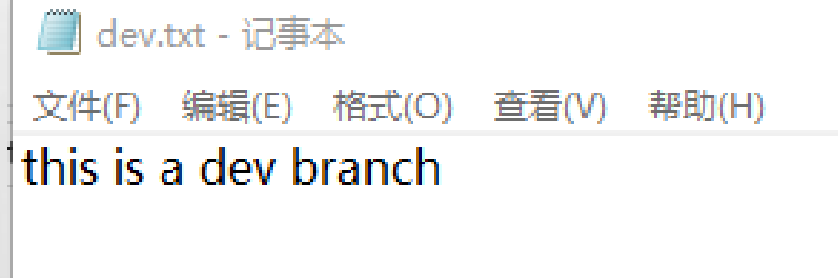
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git branch
* master

18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ ..
```

### 16、解决冲突

合并分支往往也不是一帆风顺的。准备新的dev1分支，继续我们的新分支开发：

```
git checkout -b dev1
```



修改内容前的dev.txt

在dev1分支上提交dev.txt文档：

```
git add dev.txt
```

```
git commit -m "create new branch dev1 first modify"
```



在dev1分支上修改内容后的dev.txt

## Gitlab使用

从dev1分支切换到master分支:

`git checkout master`

```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (dev1)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)
```

Git会自动提示我们当前master分支比远程的master分支要超前1个提交。

在master分支上修改dev.txt文件的内容，添加上“back to master”:

 \*dev.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

this is a dev branch

back to master

## Gitlab使用

在master分支上提交dev.txt文档:

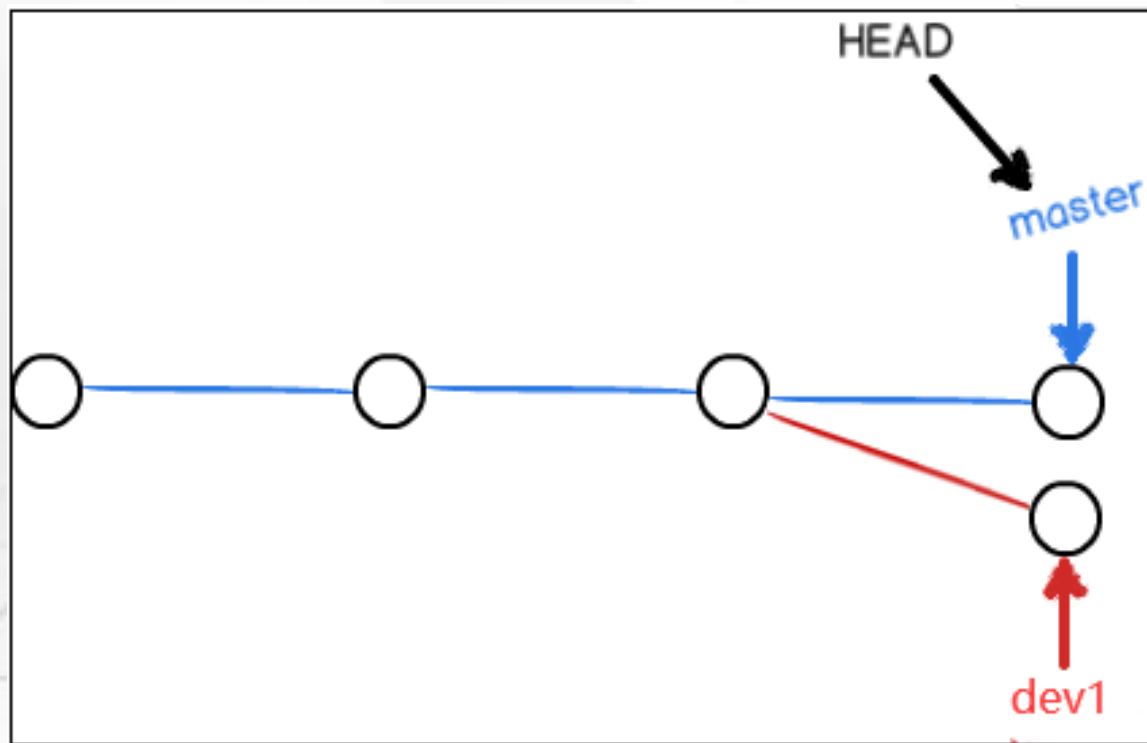
```
git add dev.txt
```

```
git commit -m "back to master first modify"
```

```
18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (master)
$ git add dev.txt
```

```
18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (master)
$ git commit -m "back to master first modify"
[master b42a8d1] back to master first modify
1 file changed, 5 insertions(+), 1 deletion(-)
```

现在，master分支和dev1分支各自都分别有新的提交，变成了这样：



## Gitlab使用

这种情况下执行合并操作时，就可能会有冲突：

```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git merge dev1
Auto-merging dev.txt
CONFLICT (content): Merge conflict in dev.txt 冲突文件
Automatic merge failed; fix conflicts and then commit the result.
```

必须手动解决冲突后再提交，打开dev.txt文档：

dev.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

this is a dev branch

<<<<<< HEAD

back to master

=====

create branch dev1

>>>>>> dev1

Git用<<<<<<，=====，>>>>>>标记出不同分支的内容，我们修改如下后保存：

\*dev.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

this is a dev branch

create branch dev1

back to master

## Gitlab使用

在被合并的分支上再次提交：

```
git add dev.txt
```

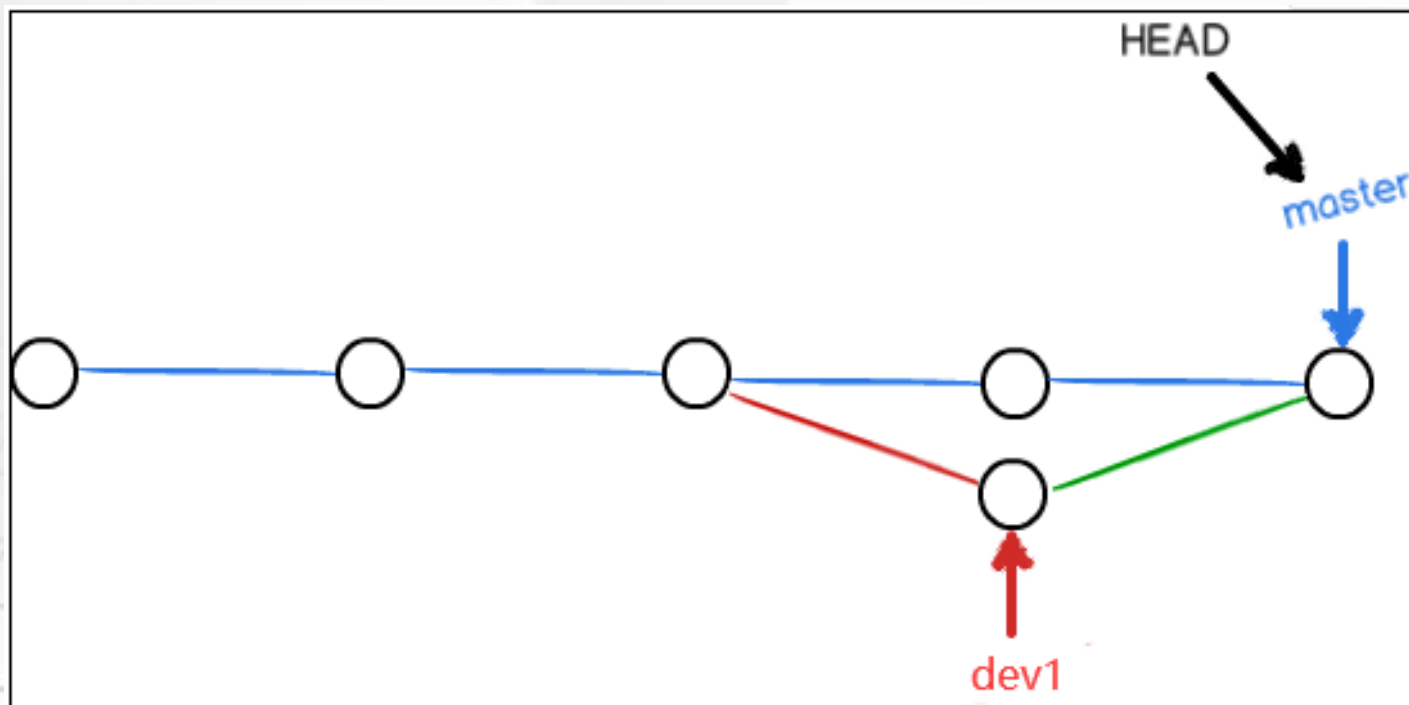
```
git commit -m "fixed conflicts"
```

```
18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (master|MERGING)
$ git add dev.txt

18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (master|MERGING)
$ git commit -m "fixed conflicts"
[master 2e20b2c] fixed conflicts

18810@DESKTOP-RSTG08N MINGW64 ~/Desktop/testproject (master)
$ .....
```

解决完冲突后，master分支和dev1分支变成了下图所示：





### 17、回退（reset和checkout）

当在本地做了修改后，不想提交，想恢复如初，Git提供了回退到前一次或前几次提交代码的办法，改写某些内容或者只是想对所推送的内容进行更正。

```
git log           //显示从最近到最远的提交日志
git log --oneline //显示log,但是不显示很多凌乱的信息
q                //显示log版本信息有很多，使用q键停止查看
git reflog        //查看曾经使用过的命令
git reset <--hard> head //当前add的工作全部消失，回到上一次commit
git reset <--hard> head^ //回到当前master的上一个commit
git reset <--hard> +commit_id //回到某个版本号的版本
```

# Gitlab使用

在dev.txt中添加“rollback”内容并保存：

```
dev.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
this is a dev branch

create branch dev1

back to master

rollback
```

```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git add dev.txt
添加到暂存区

18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

changes to be committed:
(use "git restore --staged <file>..." to unstage)
    modified:   dev.txt
准备好提交

18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git reset dev.txt
拉回到工作区
Unstaged changes after reset:
M       dev.txt

18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   dev.txt
未准备好提交

no changes added to commit (use "git add" and/or "git commit -a")
```

# Gitlab使用

在dev.txt中删除第三行的内容并保存：



dev.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

this is a dev branch

create branch dev1

rollback

将dev.txt添加到暂存区  
并提交到本地仓库：

```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git add dev.txt

18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git commit -m "modify dev.txt"
[master 2335217] modify dev.txt
1 file changed, 1 insertion(+), 1 deletion(-)
```

## Gitlab使用

接下来我们想要回删除第三行之前的内容，先使用`git log --oneline`查看日志：

```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git log --oneline
2335217 (HEAD -> master) modify dev.txt
2e20b2c (origin/master, origin/HEAD) fixed conflicts
aabb9711 back to master first modify
4dfbf4f create new branch dev1 first modify
3c6c750 back to master first modify
b42a8d1 back to master first modify
76fb2b1 create dev.txt
8a624bf Initial commit
```

“modify dev.txt”是删除第三行内容后的提交，那么前一次提交(即ID: 2e20b2c)是删除第三行内容之前的提交，使用`git checkout 2e20b2c`回到那个状态。



```
dev.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
this is a dev branch

create branch dev1

back to master
```

# Gitlab使用

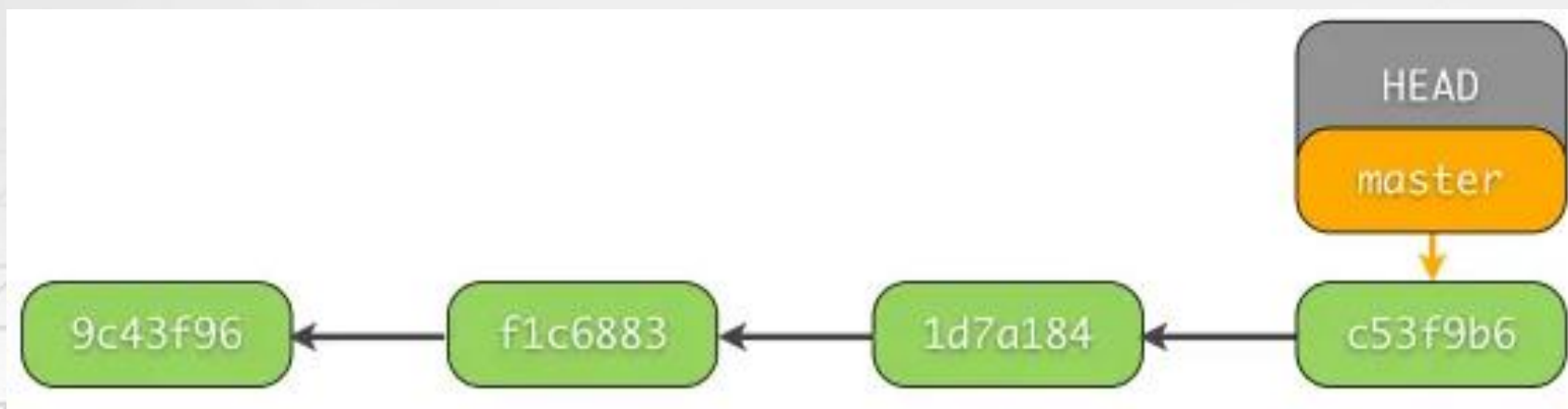
若想要在从过去回到现在，我们可以使用 `git checkout master`:

```
dev.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
this is a dev branch

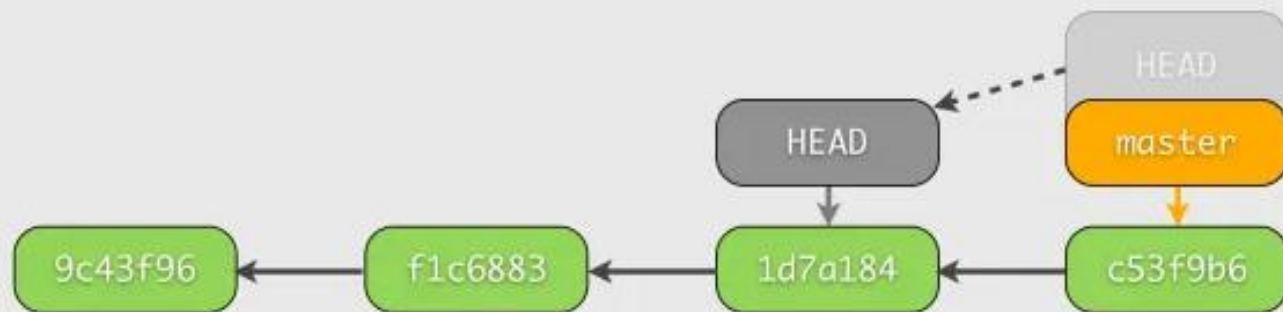
create branch dev1

rollback
```

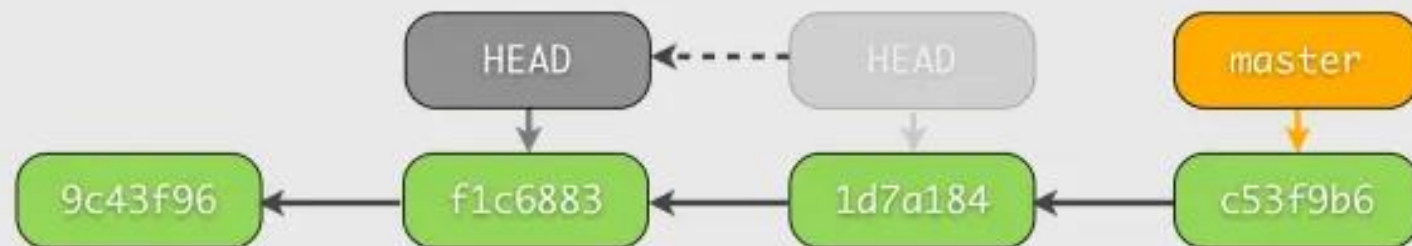
我们可以通过下面的图片对checkout进行理解:



# Gitlab使用



`$ git checkout 1d7a184`



`$ git checkout f1c6883`



`$ git checkout master`



## Gitlab使用

如果我们想要让HEAD彻底回到某一个commit，我们可以使用在reset后面加上--hard参数：

```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git reset --hard 2e20b2c
HEAD is now at 2e20b2c fixed conflicts
```

dev.txt回退到上一次提交的状态，日志也少了一条“modify dev.txt”的信息。

dev.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

this is a dev branch

create branch dev1

back to master

```
18810@DESKTOP-RSTGO8N MINGW64 ~/Desktop/testproject (master)
$ git log --oneline
2e20b2c (HEAD -> master, origin/master, origin/HEAD) fixed conflicts
aab9711 back to master first modify
4dfbf4f create new branch dev1 first modify
3c6c750 back to master first modify
b42a8d1 back to master first modify
76fb2b1 create dev.txt
8a624bf Initial commit
```

**介绍结束，感谢大家！**

