

# GIT 版本控制器

---

作者：LiuHai

## 一、简介

目前来说Git是最流行最好用的分布式的版本控制器

它能够让我们很方便的管理我们的项目版本

在我看来他是一个工具，学习它不必花费很大精力，就像我们使用日常软件一样，不必去精通它，因为有些东西可能一辈子也用不了！！

## 二、安装

### 1、在Linux系统安装

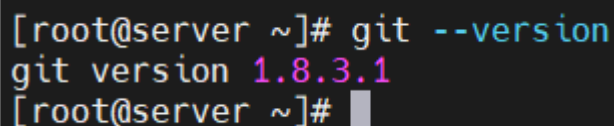
```
yum -y install git
```

只需简单的一条命令就能解决了！

然后输入命令

```
git --version
```

能够看到版本信息就说明已经成功安装了！

A terminal window screenshot with a black background and white text. The prompt is [root@server ~]#. The command git --version is entered, and the output is git version 1.8.3.1. The prompt is then shown again with a cursor.

```
[root@server ~]# git --version
git version 1.8.3.1
[root@server ~]#
```

### 2、在windows上安装git

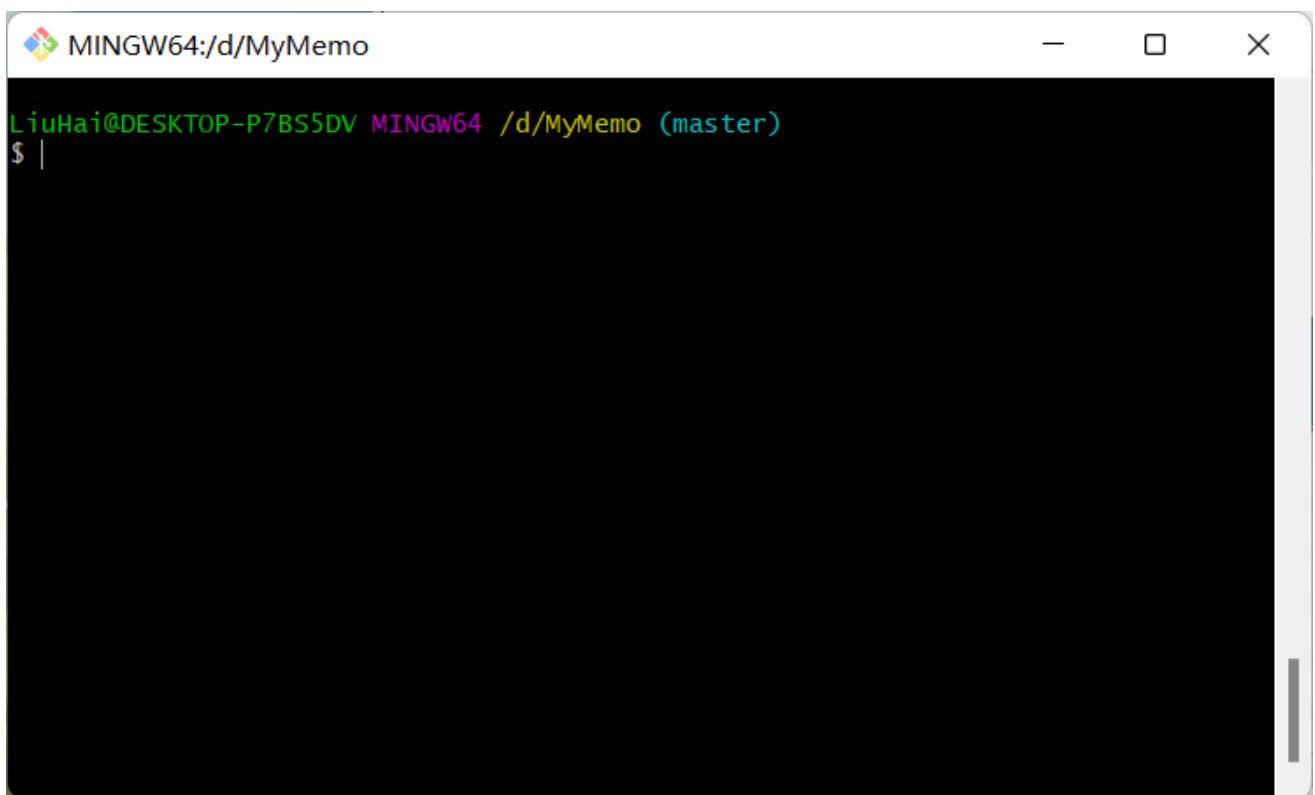
首先在官网下载安装程序：<https://git-scm.com/download/win>

然后一直Next就OK了

完成安装后，桌面右击就能看到下面这个样子就ok了



单击Git Bash Here 看到类似下面这个 就可以开始工作了



### 三、常用的指令

作为一个使用者来说掌握下面的这些常用的指令就已经可解决大部分的问题了

#### 1、创建用户和邮箱

当我们在进行推送 (push) 的时候我需要有一个身份来继续我们的操作，所以需要创建用户和邮箱

```
# 创建用户和邮箱
git config --global user.name "tom"
git config --global user.email "tom@163.com"
```

## 2、创建版本库

可以简单理解为一个目录，只是这个目录可以被git所管理，可以对目录做了什么动作进行追踪，回退

```
# 既然是一个目录那么我们就创建一个目录
# 可以是任何名字的目录
mkdir /git
# 现在还不能被git所管理，所以我们要进行进一步的操作
# 这样就完成了一个版本库的创建
git init /git
```

## 3、管理版本库

我们的git其实分三部分分别是工作区、暂存区和版本库

工作区：顾名思义就是正真工作的地方，所有做写操作的地方；

版本库：是我们工作区中的一个隐藏文件夹.git，里面最重要的就是stage（或者叫index）的暂存区，git会为我们自动创建一个分支master，以及一个指向master的指针HEAD；

暂存区：将我们的修改暂时存储到暂存区，然后一次性提交到版本库中，而后进行推送。

### 创建版本库

```
# 在工作区创建一个文件或目录
# 一定要cd到工作目录中
cd /git/
touch a.txt
# 将创建的文件放到暂存区
git add a.txt
# 如果我们有多个文件需要同时存放可以使用
git add .
# 当我们完成某个要求后就可以尝试将暂存区的资源提交到仓库
git commit -m "描述信息"
```

### 版本穿梭

当我们进行多次提交后，可能连自己也忘记，那么这个时候就需要查找的功能了

```
# 查看当前所在版本之前的版本
git log
# 查看当前所在版本之前的版本，相比git log更加简介
git log --oneline
# 查看每一次执行的命令
git reflog
```

现在我们已经能够找到历史版本了，也就是说我们找到了后悔药

在git中有一个指针叫HEAD指向的就是当前版本，上一个版本就是HEAD^,上上个版本就是HEAD^^以此类推，当我们的你是版本过多时我们能要写很多^,所以我们可以采取HEAD~n(n是一个数值代表去到那个版本)，那如果数也数不过来我们还可以通过commit id的方式进行版本的穿梭

```
# 通过^回退一个版本
git reset --hard HEAD^
# 通过数值的方式
git reset --hard HEAD~10
# 通过版本ID的方式
git reset --hard c30efe0
```

#### 4、版本的比较

版本的比较分为两种情况

```
# 1) 为添加到管理区的
git diff a.txt
# 2) 已经添加到管理区的 (cached:缓存)
git diff --cached a.txt
```

#### 5、撤销修改

撤销修改也分为两种情况

```
# 1) 修改还未添加到暂存区，那么撤销后将回到修改前的状态
# 2) 修改已经添加到暂存区，那么撤销后将回到暂存区的状态
git checkout -- a.txt
```

#### 6、删除文件

同样删除文件也有两种方案

```
# 1) 方案一
# 删除工作区的 --> 添加到暂存区 --> 提交到仓库
rm -rf a.txt
git add .
git commit -m "描述信息"

# 1) 方案二
# 从版本库中删除直接
git rm a.txt
```

#### 7、标签管理

```
# 首先要切换到要打标签的分支
git checkout master
# 创建标签
git tag v1.0
```

```
# 给之前的版本创建标签
git tag v0.9 f52c633
# 查看所有的标签
git tag
# 查看标签信息
git show v1.0
# 删除标签
git tag -d v1.0
# 推送一个标签到远程仓库
git push origin v1.0
# 一次性全部推送
git push origin --tag
# 删除已经推送到远程仓库的标签
# 先本地删除
git tag -d v1.0
# 再远程删除
git push origin :refs/tags/v1.0
```

## 四、分支管理

分支之间是互不影响的，每个分支都是独立的，当一个分支的任务完成后就会往master分支上合并，master通常是一个稳定的分支，不在他上面工作。

### 1、创建与合并分支

```
# 创建分支 (branch:分支)
git branch dev
# 切换分支 (checkout:查看)
git checkout dev
# 另一种切换分支 (switch:转变)
git switch master
# 创建并切换分支
git checkout -b dev
git switch -c dev
# 查看分支
git branch
# 合并分支 (merge:合并)
git merge dev
# 删除分支
git branch -d dev
```

### 2、合并分支冲突

一般很少会出现冲突的问题，但是有时候会也有可能出现

当出现冲突是需要手动的解决

```
# 查看分支合并图
git log --graph
```

### 3、Fast forward模式

这种模式下删除分支后会丢失分支信息

如果强制禁用的话，Git就会在merge（合并）时生成新的commit，这样就可以看到历史的分支信息了。这样会可以看出曾经做过合并

```
# 强制禁用Fast forward
git merge --no-ff -m "描述信息" dev
```

因为禁止合并会创建commit所以需要加 -m 参数

## 五、远程仓库

世界知名的：Github

国内知名的：Gitee

要实现远程的克隆与推送，需要将公钥文件拷贝到远程仓库上

```
# 关联远程仓库
git remote add my-gitlab git@192.168.10.23:maven-test/maven-proj.git
# 查看关联信息
git remote -v
# 克隆远程仓库
git clone git@192.168.10.12:/git/pub.git
# 推送到远程仓库 分支使用自己的
git push origin master
```

对于运维来说使用上，到在这里，作为日常使用其实已经够用了