

Linux环境基础开发工具使用

一.Linux软件包管理器yum

- 软件安装方式

- 源码安装
- rpm安装 (类似于安装包)
- yum 安装 (本身会考虑依赖关系)
 - linux安装软件, 可能会存在大量的软件依赖关系, 使前两种安装非常麻烦
 - yum就相当于linux的客户端

- yum 安装的部分指令

- yum list 就是搜索, 可以搜索yum上面所有的软件(root用户操作, 普通用户需要用sudo来提升权限)
- 可以通过管道和grep来进行搜索
 - 例:

```
sudo yum list | grep 'sl'  
sudo yum list | grep 'sl\.x86_64' (加个\代表绝对匹配)
```

- yum安装指令:sudo yum install (-y可以不需要询问直接安装完成) sl.x86.64(要下载安装的软件名)
- yum卸载指令: sudo yum remove sl.x86_64(要删除的文件名)
 - 一般删除的时候会有询问是否删除
 - 如果不想询问可以加一个-y: sudo yum -y remove sl.x86.64

```
[xifeng@VM-16-14-centos ~]$ cd /etc/yum.repos.d/  
[xifeng@VM-16-14-centos yum.repos.d]$ ls  
CentOS-Base.repo  CentOS-Epel.repo  epel.repo  epel-testing.repo  
[xifeng@VM-16-14-centos yum.repos.d]$ ll  
total 16  
-rw-r--r-- 1 root root 614 Jan 22 13:31 CentOS-Base.repo  
-rw-r--r-- 1 root root 230 Jan 22 13:31 CentOS-Epel.repo  
-rw-r--r-- 1 root root 1358 Sep  5 01:37 epel.repo  
-rw-r--r-- 1 root root 1457 Sep  5 01:37 epel-testing.repo  
[xifeng@VM-16-14-centos yum.repos.d]$
```

cd /etc/yum.repos.d/ (yum源在这个路径下面), 只需要**关注Centos-Base.repo**这个文件就可以了 (就是yum所使用的基础服务, yum找软件时优先从这里面**找软件**)

- 添加扩展源:

```
sudo yum install -y epel-release
```

- 注意点:

- yum要工作, 必须联网
- centos里面, 只能有一个yum在运行
- 有一个实现linux和windows文件互传的软件安装包: yum install -y lrzsz.x86_64
 - rz 是windows传到linux上

- sz 是linux传到windows上
 - sz file_name(文件名)

二.Linux开发工具

- IDE(集成开发环境) 例: vs2019

编写一个程序，必须要经过一下几个步骤（在vs中将以下步骤打包了）：

- Linux编写代码：**vi** , **vim**：文本编辑器(从定位上和记事本没有区别)
- 编译代码
- 调试代码
- 发布代码、运行
- 代码关系之间的维护

Linux编辑器--vim使用

- vim 就是一个文本编辑器，只能写代码
 - 打开 vim方法：**vim file_name**(文件名，可存在可不存在，不存在直接创建)
 - 退出方法**shift + :** 然后输入q
- vim是多模式的编辑器
 - **命令模式**(打开的默认模式)
 - 输入**i/a/o**进入插入模式
 1. i 进入光标位置不动
 2. a 进入插入模式光标后移一位
 3. o 进入插入模式另开一行
 - 光标相关：
 1. **h(左) j(下) k(上) l(右)**；
 2. 光标位置锚点**shift+^** (行首) **shift+\$** (行尾)
 3. 行跳转：**gg(起始行)** **shift +g (结束行)** **n+shift+g (跳转到指定行, n代表行数)**
 4. **w(向后)/b(向前)**：以单词为单位进行光标移动
 - 文本操作：
 1. **yy** 复制当前行; **nyy**复制当前行及其之后的n行（包含当前行）；
 2. **u**：撤销误操作
 - ctrl+r撤销最近一次的撤销
 3. **p**：粘贴，**np**：一次重复粘贴n行
 4. **dd**：删除光标当前所在行（支持n~~dd~~）
 - （先）dd --> （后）p就是剪切（支持n~~pp~~）
 5. **shift+~** 快速大小写切换
 6. **x**：删除光标之后的一个字符(从左向右删)，支持nx
 - **shift+x(相当于X)**：右向左删，支持nX
 7. **r**：替换单个字符，光标所在的字符，支持nr
 - **shift+r(R)**：替换模式，直接进行多个内容的替换
 - **底行模式-->shift :**
 - w 表示写入，q表示退出，一般对文件进行修改之后需要保存就使用shift+: 然后输入wq
 - !表示强制，可以强制写入或者强制退出。例：q!, w!, wq!
 - set nonu 关闭行号
 - set nu 显示行号

- vs file_name(文件名) 可以进行多文件操作
 1. [ctrl+w+w]光标在多文件中切换
 2. **注意**: 光标在哪个文件退出的就是哪个文件

- **插入模式**

- 想要退出按 **esc** 可以退出到命令模式
- 一般不能插入模式直接转换成底行模式

vim的简单配置

- vim配置在自己的配置文件中，只会影响自己的操作
- root 有自己的vim配置文件，只影响自己
- 配置
 1. 首先在工作目录下创建 .vimrc,然后用vim打开
 2. 之后就可以在里面写指令然后保存退出，例：set nu保存退出后再使用vim就会自动显示行号

3. `curl -sLf https://gitee.com/HGtz2222/VimForCpp/raw/master/install.sh -o ./install.sh && bash ./install.sh`

只支持centos 7

4. sudo 添加信任关系

```
99 ## Allow root to run any commands anywhere
100 root ALL=(ALL) ALL
101 xifeng ALL=(ALL) ALL
102
```

首先进入/etc这个目录，然后用vim sudoers打开文件只需要找到上面##后面的语句然后在root下面添加上自己的用户即可

Linux编译器--gcc/g++基础使用

一.gcc的使用

- `gcc file_name` (文件名) 就可以直接编译了

```
[xifeng@VM-16-14-centos lesson6]$ vim test.c
[xifeng@VM-16-14-centos lesson6]$ gcc test.c
[xifeng@VM-16-14-centos lesson6]$ ll
total 16
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 10:43 a.out
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
[xifeng@VM-16-14-centos lesson6]$
```

运行程序就是 `./a.out` (也可以用绝对路径 方式运行)

- -o 是可以重命名 (不加默认生成的是a.out)

- `gcc test.c -o test`

- `gcc -E(预处理) file_name`

- 预处理核心工作：进行头文件展开，去注释，条件编译，宏替换等等
- -E 就是说开始翻译，完成预处理之后停下来。

- 如果后面不跟一个临时文件，-E 后会把结果输出到显示屏上不方便观察，所以一般再加个 -o file_name.i (-E后形成的临时文件后缀为.i)

- 用法: gcc -E file_name -o file_name.i

```

-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 10:43 a.out
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
[xifeng@VM-16-14-centos lesson6]$ gcc -E test.c -o test.i
[xifeng@VM-16-14-centos lesson6]$ ll
total 36
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 10:43 a.out
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 10:52 test.i
[xifeng@VM-16-14-centos lesson6]$ vim test.i
[xifeng@VM-16-14-centos lesson6]$

```

• gcc -S(编译) file_name.i (也可以再用file_name编译)

- 编译的核心工作：生成汇编
 - 计算机不可以直接执行汇编语言，并且需要编译器
- S 开始进行程序的编译，完成编译之后停下来
- 形成的文件后缀为.s

- 用法: gcc -S file_name.i -o file_name.s

```

[xifeng@VM-16-14-centos lesson6]$ gcc -S test.i -o test.s
[xifeng@VM-16-14-centos lesson6]$ ll
total 900
-rw-rw-r-- 1 xifeng xifeng 74 Mar 4 2022 Makefile
-rw-rw-r-- 1 xifeng xifeng 96 Feb 27 2022 mytest.cpp
-rwxrwxr-x 1 xifeng xifeng 9640 Mar 2 2022 sum
-rw-rw-r-- 1 xifeng xifeng 159 Mar 2 2022 sum.c
-rwxrwxr-x 1 xifeng xifeng 861216 Feb 27 2022 test1
-rw-rw-r-- 1 xifeng xifeng 196 Mar 4 2022 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 2022 test.i
-rw-rw-r-- 1 xifeng xifeng 1640 Feb 27 2022 test.o
-rw-rw-r-- 1 xifeng xifeng 596 Oct 18 16:07 test.s

```

• gcc -c(汇编) file_name.s(同样可以使用file_name)

- 形成文件后缀为.o

```

[xifeng@VM-16-14-centos lesson6]$ gcc -c test.s -o test.o
[xifeng@VM-16-14-centos lesson6]$ ll
total 44
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 10:43 a.out
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 10:52 test.i
-rw-rw-r-- 1 xifeng xifeng 1640 Feb 27 11:22 test.o
-rw-rw-r-- 1 xifeng xifeng 596 Feb 27 11:12 test.s
[xifeng@VM-16-14-centos lesson6]$ vim test.s
[xifeng@VM-16-14-centos lesson6]$ vim test.o
[xifeng@VM-16-14-centos lesson6]$ od test.o
00000000 042577 043114 000402 000001 000000 000000 000000 00
0000
00000020 000001 000076 000001 000000 000000 000000 000000 00
0000
00000040 000000 000000 000000 000000 001450 000000 000000 00
0000
00000060 000000 000000 000100 000000 000000 000100 000015 00

```

- 以二进制形式查看: od file_name.o
- 注：汇编形成的二进制文件，并不可以直接执行，叫可重定向目标文件（类似于vs下的.obj）
- c：开始进行程序的翻译，完成汇编工作就停下

- gcc file_name.o -o file_name(链接直接加上.o文件即可)

- 链接

```
[xifeng@VM-16-14-centos lesson6]$ gcc test.o -o test
[xifeng@VM-16-14-centos lesson6]$ ll
total 56
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 10:43 a.out
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 11:40 test
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 10:52 test.i
-rw-rw-r-- 1 xifeng xifeng 1640 Feb 27 11:22 test.o
-rw-rw-r-- 1 xifeng xifeng 596 Feb 27 11:12 test.s
[xifeng@VM-16-14-centos lesson6]$ ./test
```

- 需要链接来将我们自己代码中的函数调用，外部数据和库关联起来
- 语言也是有库的（c语言提供了一套头文件+一套库文件libc.a和libc.so）

- 直接编译代码的两种写法（以test.c为例）

- gcc test.c -o test(这个名字可以不和原文件名相同)
- gcc -o test test.c
- ldd 可以看自己的程序用了哪个库

```
[xifeng@VM-16-14-centos lesson6]$ ldd test
linux-vdso.so.1 => (0x00007ffe9bb3e000)
libc.so.6 => /lib64/libc.so.6 (0x00007f8c61571000)
/lib64/ld-linux-x86-64.so.2 (0x00007f8c6193f000)
[xifeng@VM-16-14-centos lesson6]$
```

- 链接如何理解

- 可以理解为：为了让多个功能模块之间同时开发组合实现可执行
- Linux中：静态库：.a 动态库：.so；都与程序成功运行有关，默认使用的是动态库
- 链接就是把自己写的C程序和语言上或者第三方库提供的方法关联起来
- 所以链接也分为：静态链接 和 动态链接
 - 静态链接：库中的有关代码拷贝进自己的可执行程序中（不在需要任何库）
 - 动态链接简单理解就是：当程序运行到某个函数，自己没有定义，就会自动到库中去寻找相关信息，找到了就会执行并返回结果
- gcc默认采用动态链接方式，形成可执行程序

```
/lib64/ld-linux-x86-64.so.2 (0x00007f8c6193f000)
[xifeng@VM-16-14-centos lesson6]$ file test
test: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=b89aa1f6b052cdc3164c1853eb20441a65219d11, n
ot stripped
```

使用的是动态链接的共享库

- file file_name(可执行程序) 可以查看文件信息
- 如果想要静态链接需要在最后加 -static

```
[xifeng@VM-16-14-centos lesson6]$ gcc test.c -o test1 -static
/usr/bin/ld: cannot find -lc
collect2: error: ld returned 1 exit status
[xifeng@VM-16-14-centos lesson6]$
```

如果出现这样的报错就意味着没有安装C的静态库，安装一下就好了：**sudo yum install glibc-static**报错使用这个指令安装就可以了

```
Complete!
[xifeng@VM-16-14-centos lesson6]$ gcc test.c -o test1 -static
[xifeng@VM-16-14-centos lesson6]$ ll
total 900
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 10:43 a.out
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 11:40 test
-rwxrwxr-x 1 xifeng xifeng 861216 Feb 27 12:53 test1
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
```

- 当使用类似于(for(int i=0;i<10;+i))这种报错时，可以在编译时在结尾加上-std=c99)

```
[xifeng@VM-16-14-centos lesson6]$ vim sum.c
[xifeng@VM-16-14-centos lesson6]$ gcc sum.c -o sum
sum.c: In function 'main':
sum.c:9:3: error: 'for' loop initial declarations are only allowed in C99 mode
    for(int i=0;i<100;i++)
    ^
sum.c:9:3: note: use option -std=c99 or -std=gnu99 to compile your code
[xifeng@VM-16-14-centos lesson6]$ gcc sum.c -o sum -std=c99
[xifeng@VM-16-14-centos lesson6]$
```

二.g++使用

- 首先gcc不能编译C++，但是g++可以跑C，但是不建议用g++去跑C
- g++的操作基本上跟gcc相同

```
[xifeng@VM-16-14-centos lesson6]$ vim mytest.cpp
[xifeng@VM-16-14-centos lesson6]$ g++ mytest.cpp
[xifeng@VM-16-14-centos lesson6]$ ll
total 904
-rwxrwxr-x 1 xifeng xifeng 8968 Feb 27 17:37 a.out
-rw-rw-r-- 1 xifeng xifeng 96 Feb 27 17:36 mytest.cpp
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 11:40 test
-rwxrwxr-x 1 xifeng xifeng 861216 Feb 27 12:53 test1
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 10:52 test.i
-rw-rw-r-- 1 xifeng xifeng 1640 Feb 27 11:22 test.o
-rw-rw-r-- 1 xifeng xifeng 596 Feb 27 11:12 test.s
[xifeng@VM-16-14-centos lesson6]$ ./a.out
hello word!
```

Linux调试器--gdb基础使用

- 首先是进入：**gdb file_name**(可执行程序)；退出直接输入**quit**
 - readelf -S filename (查看一个可执行程序的段构成)
- 如果一个程序是可以被调试的，那么该程序的二进制文件一定加入了一些debug信息，反之成立
 - 在centos中，默认可执行程序是release版本
 - 想要可以调试需要在执行的时候加入一些选项，例如：**gcc test.c -o test -g**(加上-g就是debug版本)


```

total 892
-rw-rw-r-- 1 xifeng xifeng 96 Feb 27 17:36 mytest.cpp
-rwxrwxr-x 1 xifeng xifeng 8360 Feb 27 11:40 test
-rwxrwxr-x 1 xifeng xifeng 861216 Feb 27 12:53 test1
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 10:52 test.i
-rw-rw-r-- 1 xifeng xifeng 1640 Feb 27 11:22 test.o
-rw-rw-r-- 1 xifeng xifeng 596 Feb 27 11:12 test.s
xifeng@VM-16-14-centos lesson6]$ gcc test.c -o test -g
xifeng@VM-16-14-centos lesson6]$ ll
total 892
-rw-rw-r-- 1 xifeng xifeng 96 Feb 27 17:36 mytest.cpp
-rwxrwxr-x 1 xifeng xifeng 9368 Feb 27 17:58 test
-rwxrwxr-x 1 xifeng xifeng 861216 Feb 27 12:53 test1
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 10:52 test.i
-rw-rw-r-- 1 xifeng xifeng 1640 Feb 27 11:22 test.o
-rw-rw-r-- 1 xifeng xifeng 596 Feb 27 11:12 test.s
xifeng@VM-16-14-centos lesson6]$

```

因为加入了一些debug信息所以内存要大点

- gdb调试程序，必须是debug方式，也就是加个-g
- **r/run** 直接跑程序相当于 vs的ctrl+F5
- **list / l** 显示行号
- 断点：**b**(全称**breakpoint**)
 - 用法：**b/breakpoint n**(n是行号)
 - 一般 **b fun**(函数名) 断点打在了函数的入口处
- 取消断点：**d/delete n**(编号)

- 可以连续取消

```

(gdb) info b
Num      Type             Disp En
2        breakpoint      keep y
3        breakpoint      keep y
4        breakpoint      keep y
(gdb) d 3
(gdb) info b
Num      Type             Disp En
2        breakpoint      keep y
4        breakpoint      keep y
(gdb)

```

编号

- **disable** 禁用断点 **enable** 启用断点

```

(gdb) b 10
Breakpoint 5 at 0x400558: file sum.c, line 10.
(gdb) info b
Num      Type           Disp Enb Address
5        breakpoint     keep y  0x0000000000400558
(gdb) disable 5
(gdb) info b
Num      Type           Disp Enb Address
5        breakpoint     keep n  0x0000000000400558
(gdb) █

```

- **info b** 显示断点

```

(gdb) b 9
Breakpoint 1 at 0x40054f: file sum.c, line 9.
(gdb) info b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y  0x000000000040054f in main at sum.c:9
(gdb) █

```

代表断点打开

- **s/step** 进入函数 (逐语句)
- **n/next** 逐过程
- **display var**(变量名) 常显示(类似于监视窗口)
 - **display &var** 查看地址

```

(gdb) n
9          for(int i=0;i<100;i++)
4: &i = (int *) 0x7fffffffef43c
3: &sum = (int *) 0x601038 <sum>
2: i = 9
1: sum = 45
(gdb) n █

```

这些就是编号

- **undisplay n**(这里是编号) 取消常显示

```

(gdb) undisplay 3
(gdb) n
11         topsum(i);
4: &i = (int *) 0x7fffffffef43c
2: i = 10
1: sum = 45
(gdb) █

```

- **p/P** 只显示一次
 - **p var**(变量名)
- **finish** 结束当前函数
- **continue(c)** 跑完当前断点, 到下一个断点处
- **until** 跳转到指定行

- **bt** 查看调用堆栈
- **set var**(变量) 修改变量的值

例: **set var i=90**

Linux项目自动化构建工具--make/Makefile

首先了解**make** 是一条命令 **Makefile**是一个文件; **Makefile** 会维护两种东西: 依赖关系和依赖方法; **make** 和**Makefile** 加起来可以达到形成可执行程序的目的

- 创建Makefile文件

```
[xifeng@VM-16-14-centos lesson6]$ touch Makefile
[xifeng@VM-16-14-centos lesson6]$ ll
total 908
```

M/m都行

- 用vim 打开, 尽量不要空格不要空行

```
1: Makefile
1 test:test.c
2 gcc test.c -o test -std=c99
4 .PHONY:clean
5 clean:
6 rm -f test
```

代表test依赖于test.c

需要tab

依赖方法

类似于vs的清理解决方

- : 左边的文件依赖于右边的文件
- 然后直接回车到第二行然后按tab键开始写依赖方法
- **.PHONY**修饰对应的符号, 他是一个伪目标的概念(总是可执行的)
- **.PHONY:clean** (有点像类型关键字一样), 后面的**clean:** (clean没有依赖关系)

`rm -f test` 是clean的依赖方法，之后make clean 就会执行rm -f test这个功能

```
[xifeng@VM-16-14-centos lesson6]$ make
gcc test.c -o test -std=c99
[xifeng@VM-16-14-centos lesson6]$ ll
total 912
-rw-rw-r-- 1 xifeng xifeng 74 Mar 4 11:35 Makefile
-rw-rw-r-- 1 xifeng xifeng 96 Feb 27 17:36 mytest.cpp
-rwxrwxr-x 1 xifeng xifeng 9640 Mar 2 18:35 sum
-rw-rw-r-- 1 xifeng xifeng 159 Mar 2 18:34 sum.c
-rwxrwxr-x 1 xifeng xifeng 8360 Mar 4 11:36 test
-rwxrwxr-x 1 xifeng xifeng 861216 Feb 27 12:53 test1
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 10:52 test.i
-rw-rw-r-- 1 xifeng xifeng 1640 Feb 27 11:22 test.o
-rw-rw-r-- 1 xifeng xifeng 596 Feb 27 11:12 test.s
[xifeng@VM-16-14-centos lesson6]$ make clean
rm -f test
[xifeng@VM-16-14-centos lesson6]$ ll
total 900
-rw-rw-r-- 1 xifeng xifeng 74 Mar 4 11:35 Makefile
-rw-rw-r-- 1 xifeng xifeng 96 Feb 27 17:36 mytest.cpp
-rwxrwxr-x 1 xifeng xifeng 9640 Mar 2 18:35 sum
-rw-rw-r-- 1 xifeng xifeng 159 Mar 2 18:34 sum.c
-rwxrwxr-x 1 xifeng xifeng 861216 Feb 27 12:53 test1
-rw-rw-r-- 1 xifeng xifeng 217 Feb 27 10:43 test.c
-rw-rw-r-- 1 xifeng xifeng 16921 Feb 27 10:52 test.i
-rw-rw-r-- 1 xifeng xifeng 1640 Feb 27 11:22 test.o
-rw-rw-r-- 1 xifeng xifeng 596 Feb 27 11:12 test.s
```

所以make和make clean 相当于vs里面的生成和清理解决方案

make在去扫描makefile文件的时候只执行一个目标依赖关系，默认是第一个，所以想要执行对应的需要make file(文件名)

例：make clean /如果clean在上test在下那就是make test

- .PHONY修饰对应的符号，伪目标的概念（伪目标总是可执行的）

```
-rw-rw-r-- 1 xifeng xifeng 596 Feb 27 11:12 test.s
[xifeng@VM-16-14-centos lesson6]$ make
gcc test.c -o test -std=c99
[xifeng@VM-16-14-centos lesson6]$ make
make: `test' is up to date.
[xifeng@VM-16-14-centos lesson6]$ make
make: `test' is up to date.
[xifeng@VM-16-14-centos lesson6]$ make clean
rm -f test
[xifeng@VM-16-14-centos lesson6]$ make clean
rm -f test
[xifeng@VM-16-14-centos lesson6]$ make clean
rm -f test
[xifeng@VM-16-14-centos lesson6]$ make clean
rm -f test
```

接下来是Makefile的特殊符号

- 首先\$@ 代表的就是依赖关系中的目标文件，\$^代表的是右边的文件列表

```

1: Makefile
1 test:test.c
2 gcc -o $@ $^
3 .PHONY:clean
4 clean:
5 rm -f test
6

```

git

- 使用之前先查看有没有安装git(查看git --version), 如果没有使用**sudo yum install git**安装git
- 首先是gitee创建库, 然后复制库的http链接
 - readme 类似于说明书
 - 复制好链接之后到linux中输入: **git clone <https://gitee.com/linux-learning.git>**(链接粘贴就好了), 这样库中就会多一个文件

```

Cloning into 'linux-learning'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
[xifeng@VM-16-14-centos ~]$ ll
total 36
-rw-rw-r-- 1 xifeng xifeng 827 Feb 26 18:56 install.sh
drwxrwxr-x 2 xifeng xifeng 4096 Feb 22 22:22 lesson1
drwxrwxr-x 3 xifeng xifeng 4096 Feb 23 17:05 lesson2
drwxrwxr-x 5 xifeng xifeng 4096 Feb 23 16:50 lesson3
drwxrwxr-x 3 xifeng xifeng 4096 Feb 25 11:38 lesson4
drwxrwxr-x 2 xifeng xifeng 4096 Feb 26 19:09 lesson5
drwxrwxr-x 2 xifeng xifeng 4096 Mar  4 12:20 lesson6
drwxrwxr-x 2 xifeng xifeng 4096 Mar  5 09:37 lesson7
drwxrwxr-x 3 xifeng xifeng 4096 Mar  5 09:54 linux-learning
[xifeng@VM-16-14-centos ~]$ cd linux-learning
[xifeng@VM-16-14-centos linux-learning]$ ll

```

- 在.gitignore中出现的后缀, 在上传的时候会自动屏蔽

```

[xifeng@VM-16-14-centos linux-learning]$ ls -al
total 24
drwxrwxr-x  3 xifeng xifeng 4096 Mar  5 09:54 .
drwx----- 16 xifeng xifeng 4096 Mar  5 09:54 ..
drwxrwxr-x  8 xifeng xifeng 4096 Mar  5 09:54 .git
-rw-rw-r--  1 xifeng xifeng  350 Mar  5 09:54 .gitignore
-rw-rw-r--  1 xifeng xifeng  845 Mar  5 09:54 README.en.md
-rw-rw-r--  1 xifeng xifeng  934 Mar  5 09:54 README.md
[xifeng@VM-16-14-centos linux-learning]$

```

一般不要动这个文件, 这个是个本地与远端同步的文件

- 接着是上传代码

- **git status**可以查看本地代码和本地仓库的对应关系

```
[xifeng@VM-16-14-centos linux-learning]$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       procbar/
nothing added to commit but untracked files present (use "git add"
to track)
[xifeng@VM-16-14-centos linux-learning]$
```

- **git add file_name(文件名)**

```
[xifeng@VM-16-14-centos linux-learning]$ git add procbar
[xifeng@VM-16-14-centos linux-learning]$ ll
total 12
drwxrwxr-x 2 xifeng xifeng 4096 Mar  5 10:08 procbar
-rw-rw-r-- 1 xifeng xifeng  845 Mar  5 09:54 README.en.md
-rw-rw-r-- 1 xifeng xifeng  934 Mar  5 09:54 README.md
[xifeng@VM-16-14-centos linux-learning]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   procbar/Makefile
#       new file:   procbar/proc
#       new file:   procbar/proc.c
#
```

- **git commit -m "日志信息"**

```
[xifeng@VM-16-14-centos linux-learning]$ git commit -m "进度条代码"
```

日志信息必须要写，不写会报错

- 之后直接**git push** 就好

- 如果第一次传需要输入**git config user.email "写自己的邮箱"**和**git config user.name "xifeng"** (" "里面写自己的用户名)
- **git log** 查看git的日志信息
- 删除已经add的文件使用 **git rm** 命令即可，有两种选择：
 - **git rm --cached "文件路径"**，不删除物理文件，仅将该文件从缓存中删除
 - 一种是 **git rm -f "文件路径"**，不仅将该文件从缓存中删除，还会将物理文件删除
- “已经 add” 不等同于 “已经 commit”，如果要撤销上一次提交的话，可以用**git reset --soft commit_id**，commit_id 通过 **git log** 命令获取，复制上一次提交的 commit_id，粘贴即可！