Linux

笔记本： 我的第一个笔记本
创建时间： 2020/7/20 9:27 **更新时间：** 2020/8/19 17:31
作者： 1272209351@qq.com

一、基础指令

（一）命令的使用格式：命令名称 [操作选项] [操作对象]

（二）目录的命令：

（1）ls

ls ---- 浏览当前目录（windows 下的文件夹）下的文件信息
ls 默认情况下浏览 当前用户的家目录（当前主机的登录用户）
家目录------操作系统为每一个用户创建的受保护的目录
ls -a 选项：浏览目录下的所有文件，包含隐藏文件----Linux下文件名以 . 开头的文件默认不显示
ls -l 选项：查看目录下文件的详细信息
linux 下有 ----- 一切皆文件，linux 下所有东西都是文件，都可以通过操作文件的方式进行访问
linux 下文件类型并不以后缀名区别，目录也是文件只不过是目录类型文件
使用实例： ls -la Desktop/ --------- 选项可以组合使用
（不同文件显示颜色不同---蓝色：目录类型）
ls -lh：人性化显示

```
-h, --human-readable
       with -l, print sizes in human readable format (e.g., 1K 234M 2G)
```

```
man -ls ： 查看 ls 手册
man fopen ： 查看fopen 函数手册
```

给文件开辟空间大小：

```
drwxrwxr-x 7 san san 4096 Jul 15 21:52 workspace
[san@San ~]$ dd if=/dev/zero of=./hello.txt bs=500M count=1
1+0 records in
```

文件名 大小

（2）pwd

pwd --- 查看当前所在路径（打印工作路径） --- 打印出来的是一个绝对路径
/home/dev : 多层级路径的表达方式，最前边的 / 叫做根目录
绝对路径： 唯一路径，指的是以根目录为起始表达的路径
相对路径：多种多样，指的是以某一个路径作为参照路径---通常说的是以当前目录作为起始的路径

特殊文件：
. 表示目录自身
.. 表示目录的上一层目录

```
[dev@localhost ~]$ ls .
Desktop    Downloads  Music     Public     test        Videos
Documents  install    Pictures  Templates  third_part
[dev@localhost ~]$ ls /home/dev
Desktop    Downloads  Music     Public     test        Videos
Documents  install    Pictures  Templates  third_part
[dev@localhost ~]$ ls
Desktop    Downloads  Music     Public     test        Videos
Documents  install    Pictures  Templates  third_part
[dev@localhost ~]$
```

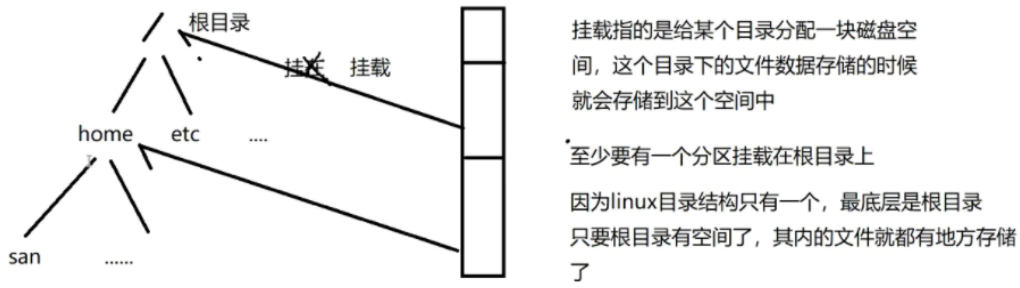
/home/san/workspace

san:./workspace home: ./san/workspace

Linux 下的目录结构：
磁盘：硬盘---存储文件
一个硬盘至少有两个分区：交换分区、文件系统分区----交换分区只有一个，文件系统分区可以有多个
交换分区：作为交换内存使用
文件系统分区：作为文件存储使用

程序运行，运行信息都占用的是内存。内存中的数据并非都是活跃数据
内存不够用---把非活跃内存区域中的数据取出来放到硬盘---交换分区，腾出来的内存就可以加载新的数据进行处理了

windows 下，磁盘分了多少分区，就可以有多少个盘符，每个盘符都是一个大目录，这个目录下的文件使用的磁盘空间就是这个分区的
windows 下的目录结构可以随着分区多少而改变；linux 下目录结构是唯一的，不会随分区多少而改变
Windows 就像是给某一个空间分配文件夹，linux 就像是给某一目录分配空间



（3）mkdir

mkdir ---- 创建目录（创建文件夹）
mkdir -p ----- 递归多层级创建目录，从外往内，哪一层不存在就创建哪一层

rmdir ---- 删除空目录
rmdir -p : 递归多层级删除目录，从内往外，哪一层为空就删除哪一层

rm ---- 删除文件
rm -r ：递归删除目录下所有文件，最终删除目录
rm -i ：有一个提示信息（rm -ir 删除提示）
rm -f ：忽略提示信息（文件删除会提示是否删除？y / n）
例如：rm -rf tmp

cp ：将一个文件向另一个位置拷贝一份（默认拷贝普通文件）
cp -r：递归将一个**目录及其内部文件**全部拷贝到指定位置
cp -r tem ./test

mv ：移动一个文件或目录到另一个位置（剪切-----改名）

```
test  workspace
[san@San ~]$ mv ./test/ ./tmp
[san@San ~]$ ls
tmp  workspace
[san@San ~]$ ls tmp/
passwd  test.txt
[san@San ~]$
```

cd：改变工作路径，改变当前文件路径，进入某个目录
cd ~ ： 回到当前目录的家目录

常用操作：tab 键自动文件名补全-----避免手敲出错

（三）文件操作

（1）基本操作
touch --- touch 一个文件，若文件存在----刷新文件时间属性；若文件不存在---创建一个新文件
touch -d ： 使用指定时间刷新时间属性
例如：touch -d "2018-08-09 12:12:03" tmp
文件的时间属性：最后一次访问时间、最后一次修改时间、最后一次状态改变
touch -a：仅使用当前系统时间刷新**访问**时间
touch -m：仅使用当前系统时间刷新**修改**时间

stat ---- 查看文件状态信息

cat ----将文件内容打印出来

more ---- 分页显示文件内容
向下按行滚动---回车；向下按页滚动---空格；退出显示---q 键
less ---- 分页显示文件内容
向下按行滚动--- 向下/ 回车； 向下按页滚动---空格 /f 键；
向上滚动-----向上/ b键； 退出显示----q 键
匹配查找字符串： /string 向下找； ?string 向上找

head --- 默认显示文件前十行内容
head -n 数字 -----指定显示行数
tail ---- 默认显示文件末尾十行内容
tail -n 数字 ----- 显示末尾行数
tail -f -------动态一直等待末尾的新数据显示（自己输入）
（ctrl + C 退出）

ifconfig --- 查看网卡信息
echo --- 将后续字符串打印出来（打印字符串）-- 将数据写入标准输出--显示器设备
打印字符串：将字符串显示到界面上----操作系统控制，从键盘读取数据，要把数据写入显示器
linux 下一切皆文件-----所有东西都当做文件进行访问---对显示器操作也是个文件操作---将数据写入显示器文件

（2）重要操作

>> 或者 > ： 叫做重定向符号---进行数据流的重定向---文件重定向
>> 或 > ： 将要操作的数据，不在写入原本的文件而是写入到新文件中
例如：echo "asdf" >> test.txt ---把原本要写入到标准输出文件的数据写入到test.txt 文件中

> ：清空重定向，将数据重定向到指定文件中，但在这之前会清空文件原有内容；
>> ： 追加重定向，将数据重定向到指定文件中，并将新数据追加到该文件末尾。

管道符 | ： 连接两个命令，将前边命令的输出结果作为后边命令的输入数据，让后边命令进行处理
例如：打印23行内容：head -n 23 ./passwd | tail -n 1

```
关机命令：shutdown -h now 立即关机        /    halt 命令
su root --- 切换到管理员用户，需要输入管理员用户密码
```

（3）压缩打包

压缩：将一个文件按照一些压缩算法，将文件数据从多变少
zip（压缩）/ unzip（解压）：需要指定压缩包的名称（.zip）
gzip（压缩）/gunzip（解压缩）：不需要指定名称（Linux 下一般 .gz 后缀）
bzip2（压缩）/ bunzip2（解压缩）：不需要指定名称（.bz2）
（linux 下文件格式并不以后缀名区分，后缀名只是便于用户区分文件功能性质）

打包：将多个文件合成一个文件
tar : linux 下最常用打包工具---将多文件打包成一个文件，提供解包功能，并且打包解包同时可以进行压缩解压缩
-c ： 打包
-x： 解包
-z：打包或解包同时进行 gzip 格式压缩或解压缩
-j：打包或解包同时进行bzip2 格式压缩解压缩
-v :显示 打包解包信息
-f： 用于指定 tar 包名称，通常是作为最后一个选项，因为后边要加上包名称

tar -czvf * .tar.gz **（打包）------ tar -xzvf *** .tar.gz （解包）**
tar -cjvf * .tar.gz **（打包）------- tar -xjvf *** .tar.gz （解包）**
防止解压出错，常不指定压缩包形式：
tar -cvf * .tar.gz **（打包） tar -xvf *** .tar.gz （解包）**
（打包：*** 表示压缩包名称，** 代表将要打包的文件名）

！！默认解压的文件都在当前所在目录下，不一定在压缩包所在目录

（4）匹配查找

在终端执行命令，单引号与双引号区别：
大多数情况下，意义相同，都是为了括起一串数据，表示一个整体
单引号会消除括起来的数据的特殊字符的特殊含义；双引号不会

匹配查找命令：
grep：从文件内容中匹配包含某个字符串的行，常用于在某个文件中找函数
-i：匹配忽略大小写
-v：反向匹配，匹配不包含字符串的行
-R： 对指定目录下的文件递归逐个进行内容匹配
例如：grep -R 'san' ./
grep -v 'nologin' passwd
grep -i 'root' passwd

find：从指定目录中查找指定名称/大小/时间/类型的文件
find ./ -name "*test*" 通过文件名查找文件
find ./ -size "*test*" 通过大小查找文件
find ./ -time "*test*" 通过时间查找文件

find ./ -type d 通过文件类型查找文件------
（f：普通文件；d：目录文件；c：字符设备 b：块设备 p：管道文件 l：符号链接文件 s：套接字文件）

find ./ size -10M 通过文件大小找文件（10 M 以内文件）
+10M 超过10M 大小的文件
find ./ mmin -10 通过时间找文件
cmin、mmin、amin--分钟为单位
ctime、 mtime、atime----以天为单位
（文件的时间属性：最后一次访问时间 c、最后一次修改时间 m、最后一次状态改变 a）

bc：计算器

date：显示时间信息

date +"%s"： 时间戳。从1970年1月1日0时0分0秒到现在的秒数

date + "%Y-%m-%d　%G : %M: %S" ： 年月日 时分秒

date -s "" ： 设置系统时间

```
[san@San sprint]$ date +"%Y-%m-%d"
2020-07-22
[san@San sprint]$ date +"%Y-%m-%d %H:%M:%S"
2020-07-22 11:12:28
[san@San sprint]$
```

cal ： 日历

```
CAL(1)              General Commands Manual              CAL(1)

NAME
       cal - 显示一个日历

总览
       cal [ -mjy ] [ 月份 ] [ 年份 ]

描述
       显示一个简单的日历..  如果没有指定参数，则显示当前月份.
       选项如下所列:

       -m     显示星期一作为一周的第一天..  (缺省为星期日.)

       -j     显示儒略历的(Julian)日期 (以 1 为基的天数，从  1
              月 1 日开始计数) .

       -y     显示当前年份的日历..

       一个单一的参数指定要显示的年份    (1    -    9999)    ;
       注意年份必须被完全地指定: cal 89 不会 显示1989年的日历.
       两个参数表示月份  (1  -  12) 和年份.  如果没有指定参数,
       则显示当前月份的日历.
```

su ： 切换用户

su username

平常尽量避免直接使用root用户，因为root 用户是管理员，可以在系统中为所欲为

sudo：为普通用户的某个操作权限进行临时权限（不用每次切换管理员用户）： 需配置

sudo需要进行配置之后才能使用:

配置过程:

1. su root 切换到管理员用户
2. visudo 打开sudo配置
3. :90 跳转到文档第90行，在90行附近 ---

```
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
test    ALL=(ALL)       ALL
```

4. yy复制root这一行，p粘贴
5. 将用户名从root改变成自己的用户名称
6. :wq保存退出

sudo的使用演示： sudo yum install lrzsz

tab 键----自动补全（补不出来证明没有这个文件）
ctrl+c ---- 中断当前操作（q 键）
man ** ---- 查看帮助手册

（5）shell：为什么在终端输入一个字符串，回车会被当成命令执行，完成某个功能?

操作系统内核与用户之间的内核----命令行解释器
用户不能直接访问系统内核----直接访问内核太危险;
操作系统会提供一些接口---系统调用接口，用户只能通过这些接口完成内核某个特定功能的访问

shell 会捕捉用户的标准输入得到字符串，通过字符串判断用户想要干什么

浏览目录这种功能会涉及很多系统调用接口的使用，为了方便用户使用，因此将系统调用接口封装了一些直接完成常用功能的程序
功能程序：被称为shell命令程序

终端能够执行命令，就是因为终端打开之后默认运行了一个**程序----shell---命令行解释程序**
shell 程序多种多样：base、dash、csh

（用户-->终端-->shell --> 内核）shell 是一个用户与内核之间的沟通桥梁，在linux 下就是一个命令行解释器

权限：限制用户权力的东西
操作系统中的操作权限：在Linux系统中将用户分为两类：　　普通用户---只能干受限的操作
　　管理员用户--为所欲为
使用su 命令切换用户---为了获取这个用户的操作权限

root ->useradd 添加用户　userdel 删除用户

（6）文件权限：

文件访问对用户的分类：
　　文件所有者（u）、文件所属组(g)、 其他用户（o）
文件访问对操作的分类：
　　可读（r）、可写（w）、可执行（x）
文件访问权限表示为：
　rwxrwxrwx 描述了三类用户各自对文件所能进行的操作
例如： **rw-rw-r--：所有者对文件可读可写不可执行，所属组成员对文件可读可写不可执行，其他用户只读、不能写也不能执行**
在系统中权限存储：使用二进制比特位图---0、1
rw-rw-r--：110 110 100 --- 占空间小操作更方便
为了方便表示和记忆：可以使用三个八进制数字分别表示三类用户权限：rw-rw-r-- 》 110 110 100 》 664

　例如：753------》111 101 011
　　u: 可读可写可执行（所有者）
　　g:可读不可写可执行（所属组）
　　o:不可读可写可执行（其他用户）
目录：可浏览 -r 、可在目录下删除创建文件 -w 、可进入-x

（7）文件访问权限指令

创建一个文件的默认权限：
umask： 查看或者设置文件的创建权限掩码也就是说掩码决定了一个文件的创建的默认权限
　-S ： 人性化显示

```
777 ---- 是shell 中 默认给定的文件权限
```

```
计算方法：777（满权限）-（减去）八进制掩码
正规计算方法：给定权限 & （~掩码）---- 777 &（~002）（~ :取反）
```

例如：现在shell 中umask 掩码的值为 002 ， 问创建一个文件后三个权限
　　777 & （~ 022）= 111 111 111 & 111 101 101=755

创建好的文件权限的修改：
chmod 777 hello.text ---- 直接用八进制数进行修改
chmod a-x hello.txt ----- 删除所有用户可执行权限
针对某用户进行权限的增删 ：
　　chmod [a/u/g/o/] +/- [r/w/x] filename

文件用户信息修改：
chown username filename 修改文件所有者（只能使用 root 修改）
chgrp groupname filename 修改文件所属组

文件权限的沾滞位：
特殊的权限位 --- 主要用于设置目录沾滞位，其他用户在这个目录下能够创建文件，可以删除自己的文件，但是不能删除别人的文件
chmod + t filename

二、常用工具

（一）软件包管理工具：安装其他的软件工具
yum --- 类似于手机上的应用管家
redhat系列 ----- Linux 中默认的管理工具
提供软件包的查看、安装、移除等管理操作

三板斧操作：
查看：yum list ---查看所有软件包 yum search --- 搜索指定软件包 例如：yum search gcc
安装：su root（管理员权限) ----- yum install --安装包安装
移除：(卸载) su root (管理员权限) -- yum remove --- 软件包移除

yum makecache ---- 将软件包信息保存到本地
安装常用软件工具：
编译器：
gcc/gcc-c++、编辑器：vim、
调试器：gdb、版本管理工具：git

查看版本信息： gcc-v 、gdb-v 、 git-v 、
查看常用工具是否有版本信息（是否安装）：
vim --version、gcc--version、gdb--version、git--version
(对应安装：yum install vim、yum install gcc gcc-c++、yum install gdb、yum install git)

（二）编辑器：写代码
vim -- Linux下常用的编辑器
vim 命令行编辑器，不能使用鼠标（鼠标无效）--- 在命令行中实现光标的移动、文本的操作、文本的编辑，vim 具有多种操作模式（12种），**常用三种：插入模式、普通模式、底行模式**

插入模式：进行文本数据的编辑插入
普通模式：进行文本常见操作，复制、粘贴、剪切、删除、撤销、返回、文本对齐、光标快速移动
底行模式：进行文本的保存退出，以及文本的匹配查找替换操作

vim 的基本使用：
打开文件：vim filename -- 打开一个已有文件，若没有自动创建，打开之后默认进入普通模式
注：所在目录必须具有可写权限
vim 打开文件，每次其实是打开一个临时文件，作为中间交换文件然后关闭源文件，编辑的操作都是在中间文件中完成的，只有正规退出，：wq 的时候才会将改变的数据写入源文件中并且删除中间文件，否则中间文件存在的情况下，下次vim 打开文件时就会有提示信息（删除这个中间文件后就没有）

操作模式的切换：所有模式都是围绕普通模式进行切换的
普通模式--》插入模式：i -- 进入插入模式； a -- 光标后移一个字符进入插入模式；o -- 光标所在行下方创建新行进入插入模式
普通模式--》底行模式：使用英文 （冒号）：进入底行模式实现文本的保存和退出
底行模式--》其他模式：ESC （任何模式下按ESC键都可以返回到普通模式）

ctrl + z 暂停插入
底行模式下常见操作：
：w --- 保存　：q ---- 退出　　：wq　---- 保存并退出
：q!　-- 强制退出，不保存

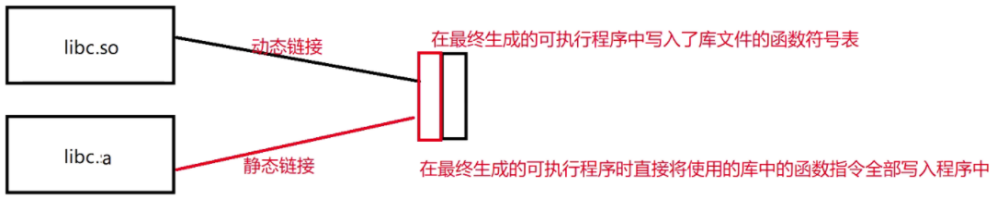
普通模式下的操作指令：
光标移动：h (上)、 j(左)、 k(右) 、 l (下)、 w /b（按单词移动光标）、ctrl + f/b （上下翻页）、gg （光标自动到文档首行）、G（文档尾行）
文本操作：yy / nyy -- 复制　 p/P:粘贴（粘贴到下方/上方）
（例如，n=2 时候2yy :复制两行内容）
vim 中没有删除--删除就是剪切，dd/ndd （剪切，n代表剪切行数
x :删除字符；dw :删除单词
其他操作：u: 撤销上部操作，ctrl+r 撤销的反向操作

gg=G　--- 全文对齐

（三）编译器：将我们所写的高级语言代码解释成机器指令集gcc：C语言编译器　　g++：C++编译器
编译器作用：C、C++是高级语言，机器无法识别这些代码，需要编译器将高级语言代码解释为机器指令生成可执行程序文件才能可执行。
可执行文件：一段功能的机器指令集
编译型语言:C/C++　　脚本型：python、lua、shell
编译型语言：程序编译之后才可以执行　--- 运行性能高，但是编码较慢
脚本语言:编写完毕直接执行----逐行解释型，由解释工具逐行解释然后执行功能（不同脚本语言有不同解释器）----编码较快但是运行效率低

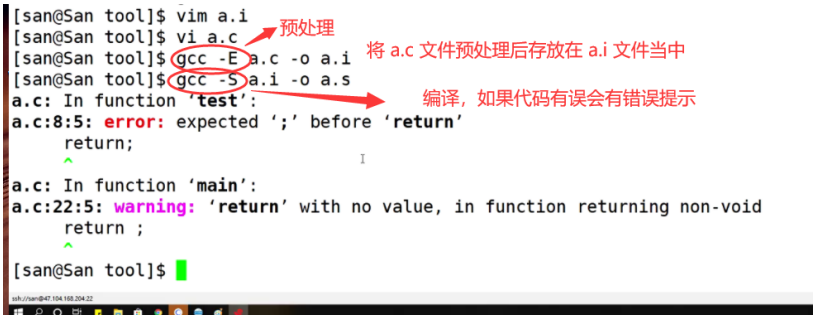
编译过程:
（1）预处理：
展开所有代码（引入头文件、宏替换、删除注释..）**gcc -E**
（2）编译：
纠正、没有错误了将代码解释成汇编代码　**gcc -S**
（3）汇编：
将汇编代码解释为机器指令
（4）链接：
将所有用到的机器指令文件打包到一起，生成可执行程序

链接库文件时的两种链接方式：(gcc默认链接：动态链接)
（1）动态链接：libc.so 链接动态库，在可执行程序中记录函数符号信息表，生成的可执行程序比较小，但是运行时需要加载动态库，多个程序可以在内存中使用同一个相同的库不会在内存中造成代码冗余
（2）静态链接：libc.a 链接静态库，直接将使用的函数实现写入可执行程序中，生成的可执行程序比较大，但是运行时不需要额外依赖加载库文件，但是如果多个程序使用了相同的静态库，运行时会在内存中造成代码冗余。



不仅仅是我们自己写的代码文件，还包括库文件
库文件：大佬已经写好的常见功能代码，打包起来的文件
例如： printf 函数 就是一个库函数，程序中如果调用printf 函数，我们的可执行程序运行时，就必须找到printf 对应的实现指令
gcc 编译器，在链接生成可执行程序时会默认链接标准C库

gcc 编译器常见操作选项：
-E：只进行预处理；-o：指定要生成的文件名称；
-S：只进行编译； -c：只进行汇编；



vi ---- 打开进入某一个文件（代码）
vnew --- 打开一个新文件窗口

未定义：告诉编译器有这个东西（变量/函数），声明 了这个变量/函数，但是编译器未找到这个东西
未声明：从未告诉编译器有这个东西就直接使用了

.h 文件--- 在C语言中自己要用于进行变量/函数的声明，告诉编译器有这个变量/函数的存在，你先编译具体变量的定义/函数的实现有可能在其他的 .c 文件中
通过 .h 文件只是为了告诉编译器有什么，编译链接时候必须把所有的 .c 文件的指令以及库文件中的指令打包到一起才行
在一个 .c 文件中引入其他 .h 文件，只是为了告诉编译器在编译这个文件时候有哪些其他文件中定义的函数/变量可以使用

（四）调试器：调试程序的运行过程 （gdb）
调试一个程序的运行过程能够让我们从运行过程中发现程序哪里有问题，可以适当的改变数据到达某种调试目的
并不是所有程序都可以进行调试，调试器只能调试具有调试符号信息的程序 --- debug 版本的程序

gcc 生成可执行程序，默认会生成 release 版本的程序，程序中没有调试符号信息，想要生成 debug 版本需要加上 -g 选项

调试一个程序的前提：
这个程序是一个 debug 版本程序
是 gcc -g a.c b.c -o main 生成的程序（a.c / b.c 是两个需要链接起来调试的文件名，生成一个新的 main 文件）

1、调试器加载程序
gdb ./main 直接使用 gdb 加载程序 ./main
gdb -p 30507 直接对一个正在运行中的程序进行调试
（ -p 用于指定进程的 id ）

2、开始调试程序
run 直接运行 start 开始进行逐步调试

3、常用调试指令
（1）流程控制：
next / n 下一步，遇到函数直接运行，不会跟踪进入函数
step / s 下一步，遇到函数则跟踪进入函数
list / l 默认查看调试行附近代码，也可以指定行:list file :line
until: 直接运行到指定行也可以指定文件: until file :line

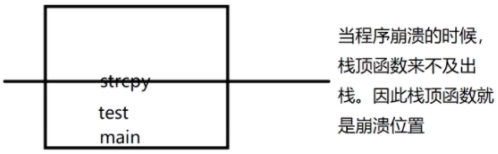
例如：list a.c:20　　until a.c:26
continue / c 继续运行直到断点处停下

（查看 a.c 15 行 附近代码）

```
(gdb) list a.c:15
10              return;
```

（2）断点操作　break / b
break file :line （例如：break a.c:9）：给指定文件指定行打断点，程序运行到断点就会停下来
break function ：给函数打断点
info break ： 查看断点信息
watch a ： 给变量 a 打断点，当变量数据发生改变时停下来
delete / d ： 删除断点

（3）内存操作
print / p 查看变量数据：例如：print a
print a=10 设置变量数据
backtrace / bt 查看程序运行调用栈信息，程序一旦崩溃，查看栈可以快速定位崩溃位置---栈顶函数



当程序崩溃的时候，栈顶函数来不及出栈。因此栈顶函数就是崩溃位置

for(int a=10;a<20;a++）变量 a 在使用时定义，出错的处理方法：

```
[san@San tool]$ gcc -std=c99 a.c b.c -o main
```

（五）项目自动化构建工具：自动化的将某个项目构架成功
make / Makefile　----------　（vi Makefile）
Makefile :普通的文本文件,用于记录项目的构建规则流程
make :Makefile 解释程序，逐行解释Makefile 中的项目构建规则，执行构建指令完成项目的构建

Makefile 的编写规则：
目标对象：依赖对象
制表符\t 执行指令

目标对象：要生成的可执行程序的名称
依赖对象：生成目标对象所需要的源码文件
make 在解释执行的时候会找到当前所在目录中的Makefile 文件，在这个文件中找到第一个目标对象，执行这个目标对象对应的指令

gcc 要是不使用 -o 指定生成文件名称，则默认生成 a.out

make 的解释执行规则：
（1）make 一旦执行，则运行make 解释程序，会在当前所在目录寻找Makefile 文件
（2）make 的执行规则中，要生成目标对象，首先要保证依赖对象已经生成，则会递归向下寻找依赖对象的生成规则
（3）在Makefile 中找到第一个目标对象，根据与依赖对象的时间关系判断是否需要重新生成（只生成第一个目标对象就会退出）
（4）若是需要重新生成，则执行对应下方的指令（指令不一定非是生成目标对象指令）
生成第一个目标对象后，make 就会退出不会生成第二个目标对象

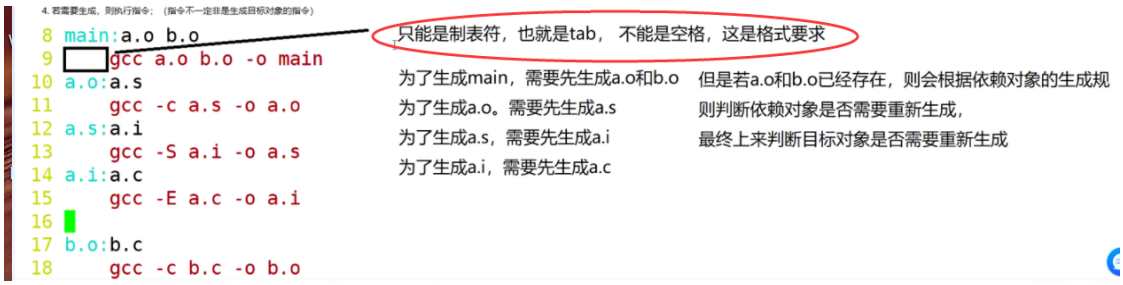
为了生成 main 需要先生成 a.o 和 b.o

为了生成 a.o 需要先生成 a.s
为了生成 a.s 需要先生成 a.i
为了生成 a.i 需要先生成 a.c　　（递归过程）

（若a.o 和 b.o 已经存在，则会根据依赖对象的生成规则判断依赖对象是否需要重新生成，最终来判断目标对象是否需要重新生成）

```
 8 main:a.o b.o
 9     gcc a.o b.o -o main
10 a.o:a.s
11     gcc -c a.s -o a.o
12 a.s:a.i
13     gcc -S a.i -o a.s
14 a.i:a.c
15     gcc -E a.c -o a.i
16
17 b.o:b.c
18     gcc -c b.c -o b.o
```

```
[san@San tool]$ ls
a.c  b.c  b.h  Makefile
[san@San tool]$ make
gcc -E a.c -o a.i
gcc -S a.i -o a.s
gcc -c a.s -o a.o
gcc -c b.c -o b.o
gcc a.o b.o -o main
[san@San tool]$
```



Makefile 编写规则进阶：

make 中的预定义变量：预定义变量通常是在执行指令中使用

$@：表示目标对象
$^：表示所有依赖对象
$<：表示所有依赖对象中的第一个依赖对象

```
main:a.o b.o
    gcc $^ -o $@
a.o:a.s
    gcc -c $^ -o $@
a.s:a.i
    gcc -S $^ -o $@
a.i:a.c
    gcc -E $^ -o $@

b.o:b.c
    gcc -c $^ -o $@
```

或：% 通配符，匹配识别任意字符串

```
main:a.o b.o
    gcc $^ -o $@        % :通配符
%.o:%.c
    gcc -c $< -o $@
```

wildcard / patsubst：

(shell 中 # 注释)

```
src=$(wildcard ./*.c) #从当前目录下找到以.c结尾的文件名称，放到src变量中
#$src = a.c b.c
obj=$(patsubst %.c, %.o, $(src)) #将src变量中的数据，后缀名由.c修改为.o将结果赋值给obj变量
#$obj = a.o b.o
main:$(obj)
    gcc $^ -o $@
%.o:%.c
    gcc -c $< -o $@
#makefile中的清理动作,这个对象是第二个目标对象，不会自动执行，需要我们制定执行
#声明伪对象，伪对象不管如何每次都要重新生成
.PHONY:clean
clean:
    rm -rf main $(obj)
```

执行时 make 默认只执行第一个main 文件，要想指定执行第二个 clean 文件需要输入 make clean ,如果当前文件中存在clean 文件则需要在vim 中声明伪代码

伪对象的使用：
.PHONY：目标对象名称
伪对象的作用：不管对象是否最新每次都要重新生成

（六）项目版本管理工具：可以实现项目的回滚、合并等管理操作
git： 项目的版本管理工具
版本管理工具：对项目的开发周期进行管理，每一次提交的修改都会有对应的版本号，能够让我们在程序出问题的时候回滚回去

1、从远程仓库克隆到本地： git clone https连接路径
2、本地提交修改信息：git add ./*
3、提交本地版本管理：git commit -m "删除了一个无效文件"
4、将本次版本提交到远程服务器仓库：
git push origin master
origin ---- 用于指定分支名称
master ---- 主分支