1.调试器-gdb　2.make/makefile　3.git

gdb:

1.注意的点：如果需要在linux操作系统底下，进行调试程序，需要在编译程序的时候，将程序编译成为debug版本

debug：是程序的一个调试版本，增加了一些调试信息，这些调试信息，可以帮助程序员调试代码

release：这个是程序的发布版本，客户一般拿到的是这个版本，这个版本相对与我们的debug版本，编译器在编译的时候做了优化，程序运行更快；

linux操作系统下，默认在编译的时候生成的是release版本。

如果想要生成debug，在编译的时候，需要增加 "-g" 命令行参数

2.调试的起手式

2.1 调试可执行程序

gdb [可执行程序]

- 查看源代码： l(list)

注意：当开始执行gdb命令的时候，可以直接回车，执行上一次的指令

- 打断点： b [源码文件当中的行号]
- 如何去除断点： delete [断点的序号]
- 查看断点信息： i b (info breakpoints)
- 使断点失效：disable [断点的序号]

断点本质上还是在的，但是程序在执行的时候，就不会停留在失效的断点处

- 使断点生效：enable [断点的序号]
- 使程序运行：r (run)
- 逐语句执行，对标到win当中是 F11，执行的命令是： s(step)　逐语句会进入到被调用函数内部进行执行
- 逐过程执行，对标的win当中是F10，执行的命令是： n(next)　逐过程不会进行到被调用函数内部执行
- 继续执行：c，注意，遇到下一个断点的时候就会停止执行

- 打印变量的值：p(print)

p [变量名称]，不仅仅局限于普通变量，还可以打印指针变量，还可以打印对象

- 退出：q (quit)

2.2 调试coredump文件

前提：本质上是在调试程序崩溃之后的内存镜像文件。

Segmentation fault：段错误，内存访问越界，或者访问空指针导致的

产生coredump文件的条件：

1.linux操作系统当中，需要设置core size的大小

ulimit -c

2.磁盘大小

```
-rw------- 1 wudu wudu 249856 Mar  5 20:31 core.31351
```

内存镜像文件保存的是，程序在崩溃的一瞬间内存当中的值。

gdb [可执行程序] [coredump文件]

Program terminated with signal 11, Segmentation fault

程序收到了11号信号，产生了段错误

```
Core was generated by `./bb'.
Program terminated with signal 11, Segmentation fault.
#0  0x00000000004005e3 in main () at test.c:18
18          *lp = 'a';
Missing separate debuginfos, use: debuginfo-install glibc-2.17-317.el7.x86_64
(gdb)
```

提示程序员，崩溃的代码行是哪一行

```
char* lp = (char*)malloc(1024)

strcpy(lp, "aaa");
free(lp)

*lp = 'a';
```

*lp = 'a';

解引用崩溃，有可能是解引用了空指针，也有可能是解引用了野指针

NULL = 0x00000000

```
strcpy(lp, "bbb")
```

func1
func
main

bt：查看调用堆栈

f [堆栈序号]：跳转到某一个具体的堆栈

tips：

11信号：解引用空指针，解引用野指针，越界访问内存

6号信号：double free

make & makefile

1.make自动化解释器　makefile是一个文件

2.通过make解释makefile文件，可以构建可执行程序

3.makefile文件的规则

目标对象：需要生成什么可执行程序，或者目标程序（.o）目标文件

依赖对象：生成目标对象的时候，依赖的文件

编译命令：如何使用依赖对象生成目标对象

```
目标对象:依赖对象
    编译命令
```

tab

4.make解释makefile的原则

make解释器在解释makefile的时候，会对比依赖对象（源文件）和目标对象（可执行程序）的生成时间。

（time - 目标对象的生成时间） < (time - 依赖对象的生成时间)

如果目标对象（可执行程序）生成的时间距离现在较近，说明目标对象是最新的，不需要重新编译

（time - 目标对象的生成时间） > (time - 依赖对象的生成时间)

如果依赖对象（源文件）生成的时间距离现在比较进，说明依赖对象（源码文件）更改过，需要重新编译

1.只为生成第一个目标对象而服务，一旦make解释生成了第一个目标对象，则停止解释。

2.make解释器在解释makefile的时候，为了生成第一个目标对象，也会判断第一个目标对象依赖的对象是否存在，

如果不存在，则在makefile后续的语句当中查找生成依赖对象的方法

先将依赖对象生成，在使用依赖对象，将第一个目标对象生成

5.预定义变量

$^：依赖的所有对象

$@：目标对象

6.makefile清理

想要删除生成的目标对象

clean

7.makefile当中也可以自定义变量

```
1 CFLAGS=-g
2 BIN1=exe1
3 BIN2=exe2
4
5 all:$(BIN1) $(BIN2)
6
7 $(BIN1):main.c a.c b.c d.c c.c
8     gcc $^ -o $@ $(CFLAGS)
9 $(BIN2):test.c
10    gcc $^ -o $@
11
12 clean:
13    rm $(BIN1) $(BIN2)
```

git 版本管理工具

1.克隆仓库

git clone "url"

2.上传

2.1 标记：告诉git工具需要管理那些文件了

git add [filename]

2.2 提交到本地仓库

git commit -m "[提交日志]"

2.3 推送给远端仓库

git push origin master

3.删除仓库当中的内容

本地仓库的内容 + 远端仓库内容

git rm "文件"

git commit -m "xx"

git push origin master

4.从远端仓库下载源代码

git pull