1.find命令

```
find . -type f -name "*.py" | xargs grep -Hn --color=auto "CUDA_VERSION"
find   path   -option   [   -print ]   [ -exec   -ok   command ]   {} \;
find    .     -name    Staff.php
find   /     -type    f
find logs -type f -mtime +5 -exec   -ok   rm {} \;
```

清除日志

```
find /home/homework/log  -name *.log*  -exec rm -f {} \;
```

find . “.” 表示当前目录

-type f 表示普通文件类型，因为find还可以查找块文件，套接字文件等类型。

-name "*.py"过滤搜索的文件名字特征。

-exec [xx] {} \; 针对发现的内容执行XX命令。其中{}表示find的内容,注意 {} 和\之间有空格，\;表示分割不同的find内容。

| xargs [xx] 把前一个命令的输出当做是xx 命令的输入。其中 ”|“表示通道。

-Hn H表示显示文件名称， n表示显示行号。

--color=auto 表示高亮显示输出。

2.grep 命令

1.-A<显示行数> 或 --after-context=<显示行数> ：除了显示符合范本样式的那一列之外，并显示该行之后的内容。

-B<显示行数> 或 --before-context=<显示行数> ：除了显示符合样式的那一行之外，并显示该行之前的内容。

```
 cat error.log |grep -A 1  '2019/01/29 10:46:06' |head -10
```

2.-c 或 --count ：计算符合样式的列数。

```
cat error.log |grep -c  '2019/01/29 10:46:06' |head -10
```

3。-i 或 --ignore-case ：忽略字符大小写的差别。

4. -n 显示行号

3.sed 命令

sed可依照script的指令，来处理、编辑文本文件。

Sed主要用来自动编辑一个或多个文件；简化对文件的反复操作；编写转换程序等。

sed命令的选项(option):

-n： 只打印模式匹配的行

-e： 直接在命令行模式上进行sed动作编辑，此为默认选项

-f： 将sed的动作写在一个文件内，用–f filename 执行filename内的sed动作

-r： 支持扩展表达式

-i： 直接修改文件内容

```
sed -e  '/^this/d' 1.php      d 删除
sed -i '/^this/d' 1.php
sed -i 's/the/& ok/' 1.php    s添加
```

1)使用行号，可以是一个简单数字，或是一个行号范围

x	x为行号
x,y	表示行号从x到y
/pattern	查询包含模式的行
/pattern /pattern	查询包含两个模式的行
pattern/,x	在给定行号上查询包含模式的行
x,/pattern/	通过行号和模式查询匹配的行
x,y!	查询不包含指定行号x和y的行

```
sed  -n '/line/p' 1.php  打印符合 line的 行
sed  -n '/first/,3p' 1.php
 sed  -n '/second/,/last/p' 1.php
 sed -n '1,4{=;p}' 1.php  展示行号
 sed -n '2,3 !{=;p}' 1.php   !前面取反
```

2)使用正则表达式、扩展正则表达式(必须结合-r选项)

^	锚点行首的符合条件的内容，用法格式"^pattern"
$	锚点行首的符合条件的内容，用法格式"pattern$"
^$	空白行
.	匹配任意单个字符

3)sed的编辑命令(sed command)：

p	打印匹配行（和-n选项一起合用）
=	显示文件行号

a\	在定位行号后附加新文本信息
i\	在定位行号后插入新文本信息
d	删除定位行
c\	用新文本替换定位文本

```
1.  sed -i '/connect/s#YES#NO#' test
2.          #匹配connect的行，把YES替换成NO
```

4. awk 命令

awk [-F|-f|-v] ‘BEGIN{} //{command1; command2} END{}’ file

[-F|-f|-v] 大参数，-F指定分隔符，-f调用脚本，-v定义变量 var=value

' ' 引用代码块

BEGIN 初始化代码块，在对每一行进行处理之前，初始化代码，主要是引用全局变量，设置FS分隔符

// 匹配代码块，可以是字符串或正则表达式

{} 命令代码块，包含一条或多条命令

; 多条命令使用分号分隔

END 结尾代码块，在对每一行进行处理之后再执行的代码块，主要是进行最终计算或输出结尾摘要信息

eg:

awk -F ":" '{print $2}' wxmis.log.wf |head -2 输出： 的

awk -F ":" '{print "Username:" $1 " Uid:" $2 }' wxmis.log.wf |head -2 自定义输出

//匹配代码块

awk '/mysql/{print $0}' /etc/passwd //三条指令结果一样

awk '!/mysql/{print $0}' /etc/passwd //输出不匹配mysql的行

//多次匹配

awk -F "optime" '{print $2}' wxmis.log.wf |awk '{print $3}'|head -2

awk -F: 'BEGIN{i=1} {while(i<NF) print i,$i,i++}' wxmis.log.wf 把 匹配的变量 打出来