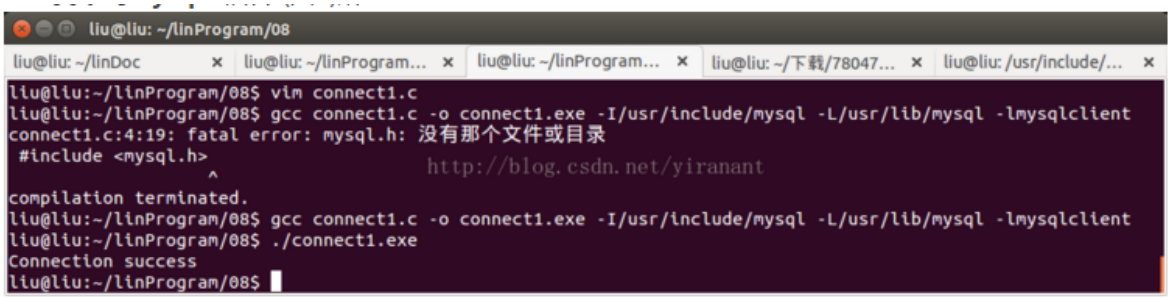
Linux Knowledge Point

Linux下找不到mysql.h(连接不到mysql.h)

- 提示信息与原因：
  - 编写C语言程序connect1.c与MySQL数据库建立连接，在命令行输入：

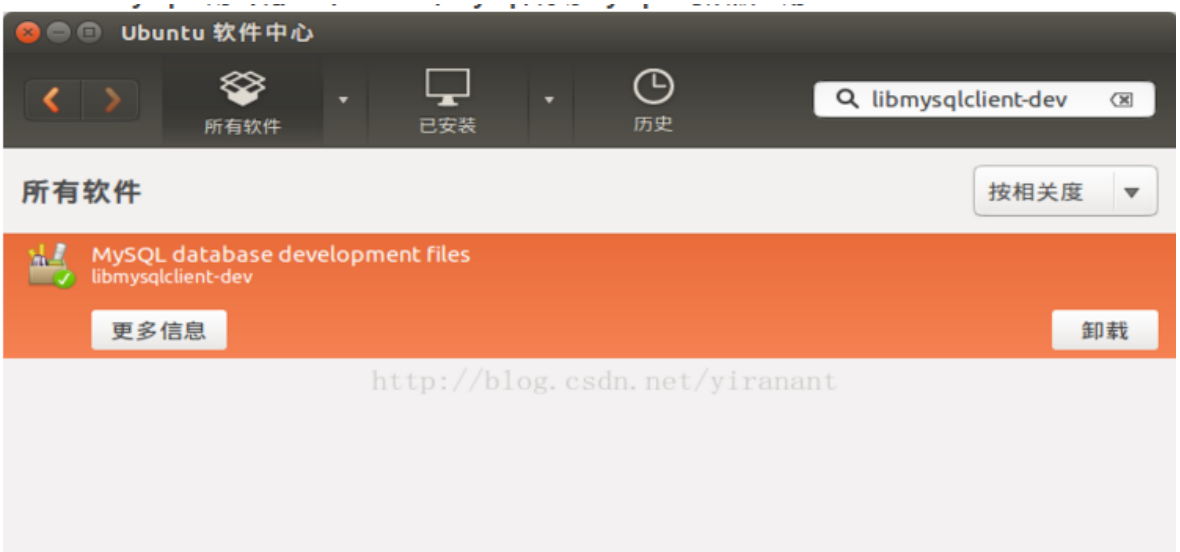
```
gcc connect1.c -0 connect1.exe -I/usr/include/mysql -L/usr/lib/mysql -lmysqlclient
```

  - 截图信息:



  - 原因分析：安装MySQL时只是安装了MySQL的服务器和MySQL的客户端，没有安装MySQL database development files.

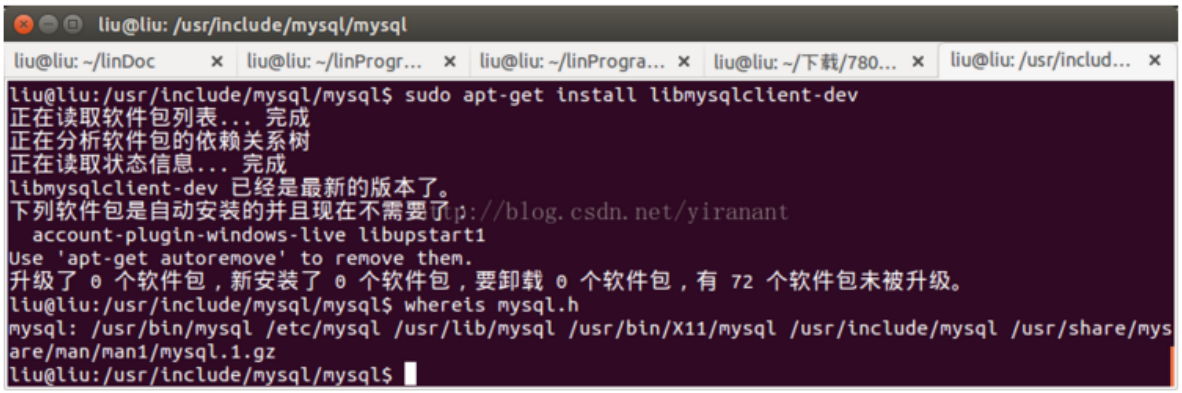
- 解决方法:
  - 图形界面上搜素安装。



  - 命令行方式安装。

    输入: sudo apt-get install libmysqlclient-dev

    确认：where is mysql.h

- 

sed & grep & awk

1. 将字符串中匹配到的数字添加特定的格式：

```
命令：echo abc123d | sed 's/\([0-9]\{3\}\)/[\1]/'
output: abc[123]d
解释说明：
    a): s///: sed中替换功能
    b): ([0-9]{3})匹配字符中的数字，（）和{}需要进行转义
    c): \1代表前面第一个正则匹配到的结果，[\1]：表示将匹配到的结果添加中括号。

例题二：将20101018000000转换成：2010-10-18 00:00:00
  >>>date_info=20101018000000
  >>>echo $date_info | sed 's/\([0-9]\{4\}\)\([0-9]\{2\}\)\([0-9]\{2\}\)\([0-9]\{2\}\)\([0-9]\{2\}\)\([0-9]\{2\}\)/\1-\2-\3 \4:\5:\6/'
  >>> 2010-10-18 00:00:00
```

2. Python中使用re模块实现步骤1

```
1. 常规语法：
    命令：re.sub(r'(\d{4})-(\d{2})-(\d{2})', r'\3/\2/\1', '2017-10-13')
    output: 13/10/2017
2. 分组语法：
    命令：
        re.sub(r'(?P<year>\d{4})-(?P<month>\d{2})-(?P<day>\d{2})',r'\g<day>/\g<month>/\g<year>','2017-10-13')
    output: 13/10/2017
```

3. 使用sed去删空白格

```
echo $line | sed 's/[[:space:]]//g'
```

4. 匹配指定的行直接在原文件上操作

```
a): 匹配#号的行，并删除
     [root@iZm5ef0leha0lvmmhfad0tZ conf]# cat hostlist
        #10.112.32.33
        192.168.33.34
      [root@iZm5ef0leha0lvmmhfad0tZ conf]#  sed -i '/#/d' hostlist
      [root@iZm5ef0leha0lvmmhfad0tZ conf]# cat hostlist
      192.168.33.34
      [root@iZm5ef0leha0lvmmhfad0tZ conf]#
```

5. 使用grep + regexp实现分段截取的功能
    - grep实现截取功能

```
string="--database offlineLog --begin 20171101000001 --end 20171101120000 -s tannei"
[root@iZm5ef0leha0lvmmhfad0tZ test]# echo $string | grep -oE '([0-9]{14})'
20171101000001
20171101120000
[root@iZm5ef0leha0lvmmhfad0tZ test]# echo $string | grep -oE '([0-9]{14}).*([0-9]{14})'
20171101000001 --end 20171101120000
```

    - grep参数: -A -B

```
a): -A: **根据用于筛选的字段往下输出N行。
[root@e07a05468.eu6sqa /usr/local/bin]
#docker inspect  92b9cf353c04 | grep Binds -A 5
            "Binds": [
                "/sys:/rootfs/sys:ro",
                "/disk2/telegraf:/rootfs/mnt/disk2:ro",
                "/disk3/telegraf:/rootfs/mnt/disk3:ro",
                "/usr/local/rds/log/:/usr/local/rds/log/:rw",
                "/disk1/telegraf:/rootfs/mnt/disk1:ro",

[root@e07a05468.eu6sqa /usr/local/bin]
#docker inspect  92b9cf353c04 | grep Binds -A 10
            "Binds": [
                "/sys:/rootfs/sys:ro",
                "/disk2/telegraf:/rootfs/mnt/disk2:ro",
                "/disk3/telegraf:/rootfs/mnt/disk3:ro",
                "/usr/local/rds/log/:/usr/local/rds/log/:rw",
                "/disk1/telegraf:/rootfs/mnt/disk1:ro",
                "/home/telegraf:/rootfs/home:ro",
                "/var/run:/var/run:rw",
                "/proc:/rootfs/proc:ro",
                "/etc/localtime:/etc/localtime:ro"
            ],
b): -B: **根据用于筛选的字段往上输出N行。
[root@e07a05468.eu6sqa /usr/local/bin]
#docker inspect  92b9cf353c04 | grep Binds -A 1 -B 3
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": [
                "/sys:/rootfs/sys:ro",

[root@e07a05468.eu6sqa /usr/local/bin]
#docker inspect  92b9cf353c04 | grep Binds -A 1 -B 5
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": [
                "/sys:/rootfs/sys:ro",
```

6. 使用awk实现四则运算中的除法

```
a) awk实现
    [root@iZm5ef0leha0lvmmhfad0tZ test]interval=`awk 'BEGIN{printf "%d",100/3}'`
    [root@iZm5ef0leha0lvmmhfad0tZ test]# echo $interval
    33
b) 使用bc工具，scale控制小数点后几位：
    [root@iZm5ef0leha0lvmmhfad0tZ test]# echo "scale=2; 100/3" |bc
    33.33
    [root@iZm5ef0leha0lvmmhfad0tZ test]# echo "scale=0; 100/3"|bc
    33
```

```
问题描述：
    shell中位置参数<$1>与awk中$1的发生重复问题：
shell脚本：test.sh
    echo 3 | awk '{ if($1==3) print $1}'
脚本调用：
    sh -x test.sh hello ==> 3
shell改进版：test.sh
    echo 3 | awk -v args=$1 '{ if($1==3) print args}'
脚本调用：
    sh -x test.sh hello ==> hello
```

7. grep -v

```
作用：匹配取反
例如：查找不包含perf的信息
    cat file | grep -v perf
```

Linux安装相关软件

安装上传下载小工具lrzsz

1. yum安装

```
yum install lrzsz -y
```

2. 接收和发送文件

```
接收： rz + 回车 + 弹框选择文件
发送单个或多个文件： sz filename[filename2]
发送目录下的全部文件： sz *
```

安装MySQLdb

1. yum安装(redhat、centos)

```
命令：yum install MySQL-python -y
验证：import MySQLdb
```

2. apt-get安装(ubuntu系统)

```
sudo apt-get install python-mysqldb
```

tail

1. 查看最后10条记录：

   a): tail -n 10 /var/log/pfs.log

   b): tailf /var/log/pfs.log (效果等同于a，需要注意的是，最后需要按ctrl+c才能进行其他操作。)

ls列出目录内容

```
a)：列出目录内容：
    ls
b)：列出目录内容，包含详细信息：
    ls -l
c)：列出目录所有内容，包含隐藏文件：
    ls -a
d)：列出目录内容，并对最后修改的时间和日期进行排序：
    ls -lt
e)：列出目录内容，并对最后修改的时间和日期进行排序，最小的显示在前。
    ls -ltr
```

Linux技巧：让进程在后台可靠运行的几种方法

```
背景：
    我们用telnet/ssh连接到liunx服务器，运行一些耗时比较久的任务，但结果却由于网络的不稳定导致任务中途失败，如何让命令提交后不受本地终端关闭/网络连接断开的干扰？
解决方法：
    我们知道终端关闭或者网络断开时，终端会收到HUP(hangup)信号从而关闭其所有的子进程。所以我们的解决方案是：a)：进程忽略hangup信号。b)：让进程运行在新的回话中从而成为不属于此终端的子进程。
参考网址：https://www.ibm.com/developerworks/cn/linux/l-cn-nohup/index.html
```

1. nohup/setsid/&

   a): nohup

```
原理：忽略hangup信号
格式：nohup command &
示例：[root@iZm5ef0leha0lvmmhfad0tZ ~]# nohup ping www.google.com &
[2] 7661
[root@iZm5ef0leha0lvmmhfad0tZ ~]# nohup: ignoring input and appending output to ‘nohup.out’
(默认将标准输出和标准错误重定向到nohup.out文件。我们也可以自定义输出： >filename 2>&1)
[root@iZm5ef0leha0lvmmhfad0tZ ~]# ps -ef | grep 7661
root      7661 28708  0 18:10 pts/7    00:00:00 ping www.google.com
root      7663 28708  0 18:10 pts/7    00:00:00 grep --color=auto 7661
```

   b): setsid

```
原理：改变进程归属，使其成为不属于接收HUP信号的终端的子进程。
格式：setsid command
示例：[root@iZm5ef0leha0lvmmhfad0tZ ~]# ps -ef | grep www.google.com
root      7674     1  0 18:18 ?        00:00:00 ping www.google.com
root      7683 28708  0 18:19 pts/7    00:00:00 grep --color=auto www.google.com
进一步解释：从上面可以看到我们的进程ID(PID)7674，而它的父进程ID(PPID)1(即为init进程ID)，不再成为当前终端的进程。（可以对比nohup）
```

c): &

```
原理：改变进程归属，使其成为不属于接收HUP信号的终端的子进程。
格式：(command &)
实例：[root@iZm5ef0leha0lvmmhfad0tZ ~]# (ping www.google.com &)
[root@iZm5ef0leha0lvmmhfad0tZ ~]# PING www.google.com (66.220.149.32) 56(84) bytes of data.
[root@iZm5ef0leha0lvmmhfad0tZ ~]# ps -ef | grep www.google.com
root      7729     1  0 19:09 pts/7    00:00:00 ping www.google.com
root      7733 28708  0 19:09 pts/7    00:00:00 grep --color=auto www.google.com
```

```
使用pstree查看直观感受下：
[root@iZm5ef0leha0lvmmhfad0tZ ~]# pstree -H 7729
systemd─┬─AliYunDun───14*[{AliYunDun}]
        ├─AliYunDunUpdate───3*[{AliYunDunUpdate}]
        ├─agetty
        ├─aliyun-service
        ├─atd
        ├─auditd───{auditd}
        ├─crond
        ├─dbus-daemon
        ├─dhclient
        ├─irqbalance
        ├─mysqld_safe───mysqld───29*[{mysqld}]
        ├─ntpd
        ├─ping  >>>(可以确认ping进程是init的子进程，不再属于终端的子进程)
        ├─polkitd───5*[{polkitd}]
        ├─rsyslogd───2*[{rsyslogd}]
        ├─sshd─┬─sshd───bash───su───bash
        │      ├─2*[sshd───bash───su───bash───ssh]
        │      ├─sshd─┬─bash───su───bash
        │      │      ├─2*[bash───su───bash───vim]
        │      │      └─bash───su───bash───mysql
        │      └─sshd───bash───pstree
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-udevd
        └─tuned───4*[{tuned}]
```

2. disown

使用场景：

如果事先在命令前加上nohup或者setsid可以使其避免受到HUP信号影响，那如果我们没有加这些命令直接运行，那么该如何补救呢？

解决方案：

通过作业调度和disown.

disown -h jobspc 来使某个作业忽略HUB信号。

disown -ah 使得所有的作业忽略HUB信号。

disown -rh 使正在运行的作业忽略HUB信号。

示例一：（如果提交命令时已使用“&”将命令放到后台运行，那么可直接使用disown）

```
[root@pvcent107 build]# cp -r testLargeFile largeFile &
[1] 4825
[root@pvcent107 build]# jobs
[1]+  Running                 cp -i -r testLargeFile largeFile &
[root@pvcent107 build]# disown -h %1
[root@pvcent107 build]# ps -ef |grep largeFile
root      4825   968  1 09:46 pts/4    00:00:00 cp -i -r testLargeFile largeFile
root      4853   968  0 09:46 pts/4    00:00:00 grep largeFile
[root@pvcent107 build]# logout
```

实例二：（如果提交命令时未使用“&”将命令放到后台运行，先使用ctrl+z暂停其运行，再通过bg将其放入后台运行，最后使用disown）

```
[root@pvcent107 build]# cp -r testLargeFile largeFile2

[1]+  Stopped                 cp -i -r testLargeFile largeFile2
[root@pvcent107 build]# bg %1
[1]+ cp -i -r testLargeFile largeFile2 &
[root@pvcent107 build]# jobs
[1]+  Running                 cp -i -r testLargeFile largeFile2 &
[root@pvcent107 build]# disown -h %1
[root@pvcent107 build]# ps -ef |grep largeFile2
root      5790  5577  1 10:04 pts/3    00:00:00 cp -i -r testLargeFile largeFile2
root      5824  5577  0 10:05 pts/3    00:00:00 grep largeFile2
[root@pvcent107 build]#
```

3. screen

使用场景：

我们已经知道了如何让进程免收HUP信号影响，但是如果有大量命令需要在稳定的后台中运行，如何避免我们队每一条命令进行这样的操作？

解决方案：

screen提供了ANSI/VT100的终端模拟器，使他可以在一个真实的终端下运行多个全屏的伪终端。

```
a):未使用screen时的进程树:
[root@pvcent107 ~]# ping www.google.com &
[1] 9499
[root@pvcent107 ~]# pstree -H 9499
init─┬─Xvnc
     ├─acpid
     ├─atd
     ├─2*[sendmail]
     ├─sshd─┬─sshd───bash───pstree
     │      └─sshd───bash───ping
b): 使用了screen时的进程树:
[root@pvcent107 ~]# screen -r Urumchi
[root@pvcent107 ~]# ping www.ibm.com &
[1] 9488
[root@pvcent107 ~]# pstree -H 9488
init─┬─Xvnc
     ├─acpid
     ├─atd
     ├─screen───bash───ping
     ├─2*[sendmail]
```

shell中重定向

参考网站：http://xstarcd.github.io/wiki/shell/exec_redirect.html

1. *输出重定向*

Command	Explain
command > filename	把标准输出重定向到一个文件中
command >> filename	把标准输出重定向到一个文件中(追加)
command 2> filename	把标准错误重定向到一个文件中
command 2>> filename	把标准错误重定向到一个文件中(追加)
command > filename 2>&1	把标准错误和输出重定向到一个文件中
command >>filename 2>&1	把标准错误和输出重定向到一个文件中(追加)

2. *输入重定向*

Command	Explain
command < filename	Command命令以filename文件作为标准输入。
command filename2	Command以filename1作为标准输入，以filename2作为标准输出
command << delimiter	从标准输入中输入，以delimiter为结束符。

3. *绑定重定向*

Command	Explain
command >&m	把标准输出重定向到文件描述符m中。
command m>&n	把往文件描述m中的标准输出重定向到文件描述符n中。
command <&-	关闭标准输入
command 2>&-	关闭标准错误输出

4. 绑定实例demo

```
step one: 查看当前fd信息
    [root@iZm5ef0leha0lvmmhfad0tZ ~]# ls /proc/self/fd
    0  1  2  3
step two: 将标准输出与fd6绑定(个人理解：编程中两个变量互换时的temp临时变量)
    [root@iZm5ef0leha0lvmmhfad0tZ ~]# exec 6<&1
step three: 再次查看fd信息
    [root@iZm5ef0leha0lvmmhfad0tZ ~]# ls /proc/self/fd
    0  1  2  3 6
step four: 将标准输出重定向到文件
    [root@iZm5ef0leha0lvmmhfad0tZ ~]# exec 1>test.txt
    [root@iZm5ef0leha0lvmmhfad0tZ ~]# ls
    >>> 此时控制台无任何信息输出，全部都重定向到文件中。
step five: 恢复标准输出
    [root@iZm5ef0leha0lvmmhfad0tZ ~]# exec 1<&6
    >>>如果这条命令执行失败，可通过: exec 1>/dev/tty进行恢复。
step six: 关闭fd6文件符
    [root@iZm5ef0leha0lvmmhfad0tZ ~]# exec 6>&-
```

Linux下查看操作系统当前登陆用户信息

1. 使用w命令查看登陆用户正在使用的进程信息：

```
信息来源：/var/run/utmp,
命令输出格式：用户名 用户的机器名称或tty 远程主机地址 登陆时间 空闲时间 jcpu(附加到tty的进程时间)
pcpu(当前进程所使用时间) 正在执行的事情
```

```
1. w
[testcc@iZm5ef0leha0lvmmhfad0tZ /]$ w
 09:46:46 up 47 days, 22:54,  4 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    42.120.74.107    09:03    6.00s  0.06s  0.00s w
root     pts/1    42.120.74.107    09:04   38:54   0.02s  0.02s ssh -i
/home/testcc/shymiyao.pem root@47.104.8.103
root     pts/2    42.120.74.107    09:06   39:58   0.01s  0.01s ssh -i
/home/testcc/shymiyao.pem root@47.104.8.64
root     pts/3    42.120.74.96     09:39    7:15   0.00s  0.00s bash
2. 忽略头文件信息：-h
[testcc@iZm5ef0leha0lvmmhfad0tZ /]$ w -h
root     pts/0    42.120.74.107    09:03    1.00s  0.06s  0.00s w -h
root     pts/1    42.120.74.107    09:04   40:33   0.02s  0.02s ssh -i
/home/testcc/shymiyao.pem root@47.104.8.103
root     pts/2    42.120.74.107    09:06   41:37   0.01s  0.01s ssh -i
/home/testcc/shymiyao.pem root@47.104.8.64
root     pts/3    42.120.74.96     09:39    8:54   0.00s  0.00s bash
3. 忽略jcpu，pcpu信息：-s
[testcc@iZm5ef0leha0lvmmhfad0tZ /]$ w -s
 09:48:16 up 47 days, 22:56,  4 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM              IDLE WHAT
root     pts/0    42.120.74.107     0.00s w -s
root     pts/1    42.120.74.107    40:24  ssh -i /home/testcc/shymiyao.pem
root@47.104.8.103
root     pts/2    42.120.74.107    41:28  ssh -i /home/testcc/shymiyao.pem root@47.104.8.64
root     pts/3    42.120.74.96      8:45  bash
```

2. 使用woami命令查看所使用的登陆名称：

```
1. who: 输出已登录到系统的所有用户信息
[testcc@iZm5ef0leha0lvmmhfad0tZ /]$ who
root     pts/0        2017-10-23 09:03 (42.120.74.107)
root     pts/1        2017-10-23 09:04 (42.120.74.107)
root     pts/2        2017-10-23 09:06 (42.120.74.107)
root     pts/3        2017-10-23 09:39 (42.120.74.96)
2. who am i:只显示当前登陆用户信息
[root@iZm5ef0leha0lvmmhfad0tZ /]# who am i
root     pts/0        2017-10-23 09:03 (42.120.74.107)
3. 只显示用户名称：
a): who | cut -d" " -f1 | sort| uniq
b): 直接使用现成命令：users（已经登陆的用户）
    [testcc@iZm5ef0leha0lvmmhfad0tZ /]$ users
    root root root root
c): whoami(当前权限用户)
    [root@iZm5ef0leha0lvmmhfad0tZ /]# whoami
    root
```

3. 查看系统历史信息：

```
1. 查看所有历史信息：last
2. 查看最后五个历史信息： last | tail -n 5
```

Vim下通过V来实现剪切复制

```
1. 按ESC退出编辑模式，进入到正常模式
2. 移动光标到目标位置，点击V进入到可视模式，移动光标选择目标信息。
3. 按d表示：删除， 按x表示：剪切， 按y表示：复制， 按P表示：粘贴
```

Linux下杀死进程的N中方法

1. 查找进程号

```
1.1：列出当前系统存在的进程
    a)：ps -ef
    b)：ps -aux
1.2：列出指定程序的进程：比如python
    a)：ps -aux | grep python |grep -v grep| awk '{print $2}'
        输出格式：每个pid换行输出：123\n456\n
    b)：pgrep python
        输出格式：每个pid换行输出：123\n456\n
    c)：pidof python（这里的进程必须是全名）
        输出格式：每个pid在同一行输出：123 456 567
备注：grep -v grep：在列出的进程中排除grep进程号。
```

2. 查找并删除指定进程号

```
2.1：通过xrags来接收pid
    a)：ps -aux | grep python |grep -v grep | awk '{print $2}' | xargs kill -9
    b)：pgrep python | xargs kill -9
2.2：通过``命令来接收pid：
    a)：kill -9 `pgrep python`
    b)：kill -9 `ps -aux | grep python | awk '{print $2}'`
```

查看当前主机Hostname和IP

1. 查看当前主机名称

```
a)：hostname
b)：uname -a | awk '{print $2}'
```

2. 查看当前主机ip

```
a)：ifconfig
b)：ip addr show
```

shell获取当前脚本的绝对路径

1. 获取当前脚本的绝对路径。

```
a): $(cd `dirname $0`; pwd)
b): $(dirname $(readlink -f $0)) ： 是取得软连接绝对路径
```

2. 根据步骤一种的路径获取其他文件路径。

```
a). fyx文件下的目录结构
├── bin
│   └── dir_test.sh
└── conf
    └── my.conf
b): 根据步骤一获取conf文件路径。
    dir_test.sh脚本中的code如下：
        BIN=$(cd `dirname $0`; pwd)
        CONF=$BIN/../conf
        DIR=$BIN/..
```

liunx数组

1. 数组的添加和获取

```
# 添加数组
[root@e010125004162.bja /mnt/fyx/bin]
#ins_array[0]=11

[root@e010125004162.bja /mnt/fyx/bin]
#ins_array[1]=22

# 获取数组内所有数值
[root@e010125004162.bja /mnt/fyx/bin]
#echo ${ins_array[@]}
11 22

# 通过index获取数组对应的值
[root@e010125004162.bja /mnt/fyx/bin]
#echo ${ins_array[0]}
11
[root@e010125004162.bja /mnt/fyx/bin]
#echo ${ins_array[1]}
22
```

shell中数值计算的几种方法

1. let

```
i=1
let i=$i+1  (+号之间都不能有空格)
echo $i ===> 2
```

2. expr

```
i=1
i=`expr $i + 1` (+号之间一定要有空格)
echo $i  ===> 2
```

3. (())

```
i=1
i=((i+1)) / i=((i + 1)) (+号之间空格无要求，但是i的变量不需要$引用)
echo $i
```

timeout

```
1.简介
    用来控制程序运行时间（运行指定命令，如果在指定时间后仍在运行，则杀死该进程。）
3.示例
    timeout 5 command
    (command如果在5秒内结束，则平安结束，否则，强行杀死该进程)
```

vim

- 替换

```
a):全局替换
    :%s/old_world/new_world/g
b):从第n行开始进行全局替换
    :n,%s/old_world/new_world/g
c):替换每一行的第一个
    :%s/old_world/new_world/
d):从第n行开始替换每一行的第一个
    :n,%s/old_world/new_world/
```

- 搜索

```
a):从上往下搜索:
    /pattern
    /^abc<Enter> #查找以abc开始的行
    /test$<Enter> #查找以test结束的行
    //^test<Enter> #查找^test字符串
b):从下往上搜索:
    ?pattern
```

修改时区

1. 查看当前时区信息

```
date -R; date +%z
```

2. 修改时区

```
cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

Python Knowledge Summary

python中extend与append的区别

1. extend

```
extend接收一个参数，这个参数总是一个list，并且把list中的每一个元素添加到原list中。
>>>list=[1,2,3]
>>>list.extend(['a','b','c'])
>>>list
[1, 2, 3, 'a', 'b', 'c']
>>>list.extend('hi')
>>>list
[1, 2, 3, 'a', 'b', 'c', 'h', 'i']
```

2. append

```
append接收一个参数，这个参数可以是任意形式，并简单追加到list的尾部。
>>>list=[1,2,3]
>>>list.append(['a','b','c'])
>>>list
[1, 2, 3, ['a', 'b', 'c']]
```

Python中range与xrange的区别

1. xrange:

```
其用法与range相同，不同的是range生成一个数组，而xrange生成一个生成器。
result=xrange(0,100)
type(result) ===> <type 'xrange'>
print result ===> xrange(0,100)
result[99] ===> 99
```

2. range

```
result=range(0,100)
type(result) ===> <type 'list'>
print result ===> [0,1,2,3,4,...99]
result[99] ===> 99
```

Python random seed

random.seed(int): 要每次产生随机数相同就要设置种子，相同种子数的Random对象，相同次数生成的随机数字是完全相同的；

Python logging module

1. logging介绍

总体概念：

logging提供了通用的日志系统，该模块提供了不同的日志级别，并可以采用不同的方式记录日志，如文件，HTTP GET/POST, SMTP, Socket等。

知识点：

a): logger:

提供日志接口，供应代码使用。最常用的两个功能是配置和发送日志消息。可以通过logging.getLogger(name)获取logger对象，如果不指定name则返回root对象，多次使用相同的name调用getLogger()返回同一个对象。

b): handler:

将日志记录(log record)发送到合适的目的地(destination)，比如文件，socket等。一个logger对象可以通过addHandler方法添加0到多个handler,同时每个handler又可以定义不同日志级别，以实现日志分级过滤显示

c): filter

提供一种优雅的方式决定一个日志记录是否发送到handler

d): formatter:

指定日志输出的具体格式。例如：[%(asctime)s] - [%(levelname)s] [%(module)s.%(funcName)s:%(lineno)d] [%(process)d-%(thread)d] --%(message)s

关于formatter配置，格式: %(关键字)s，对应信息如下如所示：

Format	Description
%(name)s	Name of the logger (logging channel).
%(levelno)s	Numeric logging level for the message (DEBUG, INFO, WARNING, ERROR, CRITICAL).
%(levelname)s	Text logging level for the message ('DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL').
%(pathname)s	Full pathname of the source file where the logging call was issued (if available).
%(filename)s	Filename portion of pathname.
%(module)s	Module (name portion of filename).
%(funcName)s	Name of function containing the logging call.
%(lineno)d	Source line number where the logging call was issued (if available).
%(created)f	Time when the LogRecord was created (as returned by time.time()).
%(relativeCreated)d	Time in milliseconds when the LogRecord was created, relative to the time the logging module was loaded.
%(asctime)s	Human-readable time when the LogRecord was created. By default this is of the form "2003-07-08 16:49:45,896" (the numbers after the comma are millisecond portion of the time).
%(msecs)d	Millisecond portion of the time when the LogRecord was created.
%(thread)d	Thread ID (if available).
%(threadName)s	Thread name (if available).
%(process)d	Process ID (if available).
%(message)s	The logged message, computed as msg % args.

2. 实例样式

   磁盘日志文件循环实现查看网址

   http://python.usyiyi.cn/translate/python_278/library/logging.handlers.html

```
import logging
import logging.handlers

LOG_FILE = 'test.log'
# 实例化handler
handler = logging.handlers.RotatingFileHandler(LOG_FILE, maxBytes = 1024*1024,
backupCount=5)  # 支持磁盘日志文件的循环
handler.setLevel(logging.DEBUG) # 设置handler日志级别

# 设置日志输出格式
fmt_str = "[%(asctime)s] - [%(levelname)s] [%(module)s.%(funcName)s:%(lineno)d] [%
(process)d-%(thread)d] --%(message)s"
formatter = logging.Formatter(fmt_str) #实例化formatter
# 为handler添加formatter
handler.setFormatter(formatter)

# 创建实例名test的logger对象
logger = logging.getLogger('test')
logger.addHandler(handler) # 为logger对象添加handler
logger.setLevel(logging.CRITICAL) # 设置日志级别

logging.info("first into message")
logging.debug("first debug message")
logging.error("first error message")
```

3. logger的level级别

```
import logging
print logging.NOTSET
0
print logging.DEBUG
10
print logging.INFO
20
print logging.WARNING
30
print logging.ERROR
40
print logging.CRITICAL
50
排序：  DEBUG < INFO < WARNING < ERROR < CRITICA
```

encoding

1. 编码

```
encoding="latin1"
```

python中的functools

1. 偏函数partial

- 概念和作用

```
概念：
    函数式编程思想，允许我们“重新定义”函数签名。
作用：
    对于共性部分参数进行绑定，固定原函数的部分参数，减少参数的传递，简化函数的调用。
```

- 具体应用例子1
  - **需求: 选择某一天，然后以这天为准，次日留存，3日留存，7日留存，14日留存，30日留存**
  - 常规做法

```
from datetime import datetime,timedelta
   def GetNextDay(baseday, n):
        fmt_day = datetime.strptime(str(baseday), '%Y-%m-%d')
       newday = (fmt_day + timedelta(days=n)).date()
       print str(newday)
   #基准时间
   baseday='2017-11-14'
   GetNextDay(baseday,1)
   GetNextDay(baseday,2)
   GetNextDay(baseday,3)
   GetNextDay(baseday,7)
   GetNextDay(baseday,14)
```

  - 偏函数

```
import functools
nday = functools.partial(GetNextDay, baseday)
nday(1) ===> GetNextDay(baseday,1)
nday(2)
nday(3)
nday(7)
```

- 具体应用例子2
  - **需求: 按照不同的进制转换数据**
  - 常规方法

```
int(): 将字符串转成整数，该函数提供额外的base参数，默认值为：10.
a): 函数默认以10进制进行转换
    int("10") ===> 10
b): 8进制进行转换
    int("10", base=8) ===> 8
c): 16进制进行转换
    int("10", base=16) ===> 16
```

根据上图我们编写对应的转换函数，便于我们应用：

```
def int8(num, base=8):
    return int(num,base)
def int16(num, base=16):
    return int(num, base)
```

这样我们在处理大量数据的转换时，直接调用即可：

```
a): 8进制转换的
    int8('110'), int8("1111")
b): 16进制转换的
    int16('110'), int16('1111')
```

- 偏函数

运用偏函数，简化in8和int16函数的定义.

```
int8 = functools.partial(int, base=8)
int16 = functools.partial(int, base=16)
int8('110'), int16("110")
```

2. 装饰器wraps

- 概念和作用

概念：返回函数的高阶函数
作用：在不修改原有的函数的基础上，在运行期间动态增加功能。

- 实例说明
  - 需求：**在函数运行前打印log信息**
  - 代码

```
from functools import wraps
def log(func):
    @wraps(func)
    def call_it(*args, **kwargs):
        """wrap func: call_it2"""
        print "before call"
        return func(*args, **kwargs)
    return call_it
@log
def hello():
    """test hello"""
    print "hello world!"

if "__name__" == "__main__":
    hello()
    print hello.__name__
    print hello.__doc__
result:
    before call
    hello world!
    hello
    test hello
```

python中requests的警告信息

- InsecurePlatfromWarning

python InsecurePlatformWarning 解决方法参考

yu Code 2 Comments Saturday, November 28th, 2015 Hide Sidebar

python 更新，有时候会报错如下：

```
/usr/lib/python2.7/site-packages/pip-7.1.2-
py2.7.egg/pip/_vendor/requests/packages/urllib3/util/ssl_.py:90:
InsecurePlatformWarning: A true SSLContext object is not available. This prevents
urllib3 from configuring SSL appropriately and may cause certain SSL connections
to fail. For more information, see
https://urllib3.readthedocs.org/en/latest/security.html# insecureplatformwarning.
InsecurePlatformWarning
```

解决方法是

```
$ sudo pip install pyopenssl ndg-httpsclient pyasn1 --upgrade
```

或者

```
$ sudo pip install requests[security]
```

Reference :

- http://stackoverflow.com/questions/29134512/insecureplatformwarning-a-true-sslcontext-object-is-not-available-this-prevent

来自杭州的你，很高兴你能看到这儿。若本文对你有所用处，或者内容有什么不足之处，敬请毫不犹豫给个回复。谢谢！

- InsecureRequestWarning

```
from requests.packages.urllib3.exceptions import InsecureRequestWarning
# 禁用安全请求警告
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
```