

1  
log show --last 2h | egrep ".\*sudo:.\*COMMAND=.\*"

3 错误的:

```
#!/bin/sh
## Example: a typical script with several problems
for f in $(ls *.m3u)
do
    grep -qi hq.*mp3 $f \
    && echo -e 'Playlist $f contains a HQ file in mp3 format'
done
```

正确的

```
6 #!/bin/sh~
5 ## Example: a typical script with several problems~
4 for f in ls *.m3u~
3 do~
2   grep -qi "hq.*mp3 $f" \~
1   && printf "..%s.." "Playlist $f contains a HQ file in mp3 format"~
done~
```

5  
cProfile:

```
python -m cProfile -s time sorts.py | grep sorts.py
336686/10000 0.178 0.000 0.188 0.000 sorts.py:29(quicksort_inplace)
10000 0.171 0.000 0.171 0.000 sorts.py:10(insertionsort)
338874/10000 0.151 0.000 0.232 0.000 sorts.py:20(quicksort)
30000 0.084 0.000 0.587 0.000 sorts.py:5(<listcomp>)
164437 0.036 0.000 0.036 0.000 sorts.py:24(<listcomp>)
164437 0.036 0.000 0.036 0.000 sorts.py:25(<listcomp>)
3 0.023 0.008 1.240 0.413 sorts.py:3(test_sorted)
1 0.000 0.000 1.242 1.242 sorts.py:1(<module>)
```

kernprof line profiler:

quick sort

```
Total time: 0.553738 s
File: sorts.py
Function: quicksort at line 20

Line #      Hits          Time Per Hit   % Time  Line Contents
=====
20          20              0.0      0.0     0.0      @profile
21          20              0.0      0.0     0.0      def quicksort(array):
22     335396     84132.0        0.3    15.2          if len(array) <= 1:
23     172698     34884.0        0.2     6.3              return array
24     162698     36881.0        0.2     6.7              pivot = array[0]
25     162698    159117.0        1.0    28.7              left = [i for i in array[1:] if i < pivot]
26     162698    157622.0        1.0    28.5              right = [i for i in array[1:] if i >= pivot]
27     162698     81102.0        0.5    14.6              return quicksort(left) + [pivot] + quicksort(right)
```

## insertion sort

```
Total time: 1.30088 s
File: sorts.py
Function: insertionsort at line 10
```

Line #	Hits	Time	Per Hit	% Time	Line Contents
10					@profile
11					def insertionsort(array):
12	258705	38197.0	0.1	2.9	for i in range(len(array)):
13	248705	39335.0	0.2	3.0	j = i-1
14	248705	41430.0	0.2	3.2	v = array[i]
15	2257105	433754.0	0.2	33.3	while j >= 0 and v < array[j]:
16	2008400	373694.0	0.2	28.7	array[j+1] = array[j]
17	2008400	326718.0	0.2	25.1	j -= 1
18	248705	46327.0	0.2	3.6	array[j+1] = v
19	10000	1421.0	0.1	0.1	return array

## quick sort inplace

```
Total time: 1.20659 s
File: sorts.py
Function: quicksort_inplace at line 29
```

Line #	Hits	Time	Per Hit	% Time	Line Contents
29					@profile
30					def quicksort_inplace(array, low=0, high=None):
31	335542	71881.0	0.2	6.0	if len(array) <= 1:
32	402	57.0	0.1	0.0	return array
33					
34	335140	63796.0	0.2	5.3	if high is None:
35	9598	2067.0	0.2	0.2	high = len(array)-1
36					
37	335140	63518.0	0.2	5.3	if low >= high:
38	172369	27050.0	0.2	2.2	return array
39					
40	162771	32347.0	0.2	2.7	pivot = array[high]
41	162771	32087.0	0.2	2.7	j = low-1
42	1238497	232624.0	0.2	19.3	for i in range(low, high):
43	1075726	204753.0	0.2	17.0	if array[i] <= pivot:
44	560092	105861.0	0.2	8.8	j += 1
45	560092	140447.0	0.3	11.6	array[i], array[j] = array[j], array[i]
46	162771	47023.0	0.3	3.9	array[high], array[j+1] = array[j+1], array[high]
47	162771	78862.0	0.5	6.5	quicksort_inplace(array, low, j)
48	162771	77392.0	0.5	6.4	quicksort_inplace(array, j+2, high)
49	162771	26826.0	0.2	2.2	return array

## Memroy analysis:

**Quicksort:** a lots more of calls, faster, the left and right variables to construct loops consumes most time, has more occurrences.

**Insertionsort:** less calls, slower, while loop consumes most time.

**Quicksort inplace:** least memory, no need to use memory to save temporily outcomes.

```
python3 -m memory_profiler sorts.py
Filename: sorts.py
Line #    Mem usage    Increment    Occurrences    Line Contents
=====
12  19.453 MiB -53171.438 MiB      10000    @profile
13                                def insertionsort(array):
14  19.453 MiB -1381066.531 MiB      259859        for i in range(len(array)):
15  19.453 MiB -1327870.125 MiB      249859            j = i-1
16  19.453 MiB -1327870.234 MiB      249859            v = array[i]
17  19.453 MiB -12091503.062 MiB      2275331                while j >= 0 and v < array[j]:
18  19.453 MiB -10763628.094 MiB      2025472                    array[j+1] = array[j]
19  19.453 MiB -10763630.938 MiB      2025472                        j -= 1
20  19.453 MiB -1327875.719 MiB      249859                    array[j+1] = v
21  19.453 MiB -53196.422 MiB      10000                return array

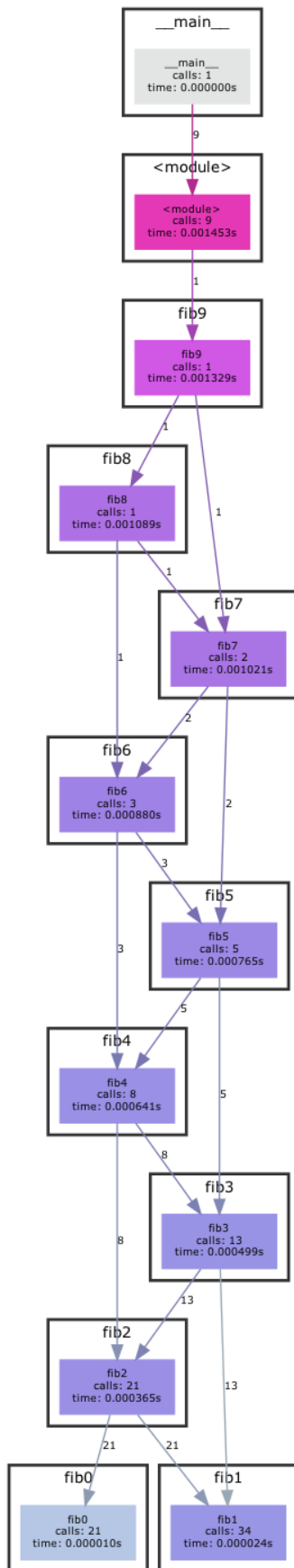
Filename: sorts.py
Line #    Mem usage    Increment    Occurrences    Line Contents
=====
23  13.969 MiB -722627.000 MiB      334678    @profile
24                                def quicksort(array):
25  13.969 MiB -722641.812 MiB      334678                if len(array) <= 1:
26  13.969 MiB -372124.906 MiB      172339                    return array
27  13.969 MiB -350517.422 MiB      162339                    pivot = array[0]
28  13.969 MiB -3370611.734 MiB      1557285                        left = [i for i in array[1:] if i < pivot]
29  13.969 MiB -3370616.734 MiB      1557285                        right = [i for i in array[1:] if i >= pivot]
30  13.969 MiB -350540.641 MiB      162339                    return quicksort(left) + [pivot] + quicksort(right)

Filename: sorts.py
Line #    Mem usage    Increment    Occurrences    Line Contents
=====
33  8.984 MiB -4842.109 MiB      338454    @profile
34                                def quicksort_inplace(array, low=0, high=None):
35  8.984 MiB -4850.984 MiB      338454                if len(array) <= 1:
36  8.984 MiB -5.531 MiB      393                    return array
37
38  8.984 MiB -4845.453 MiB      338061                if high is None:
39  8.984 MiB -138.297 MiB      9607                    high = len(array)-1
40
41  8.984 MiB -4845.453 MiB      338061                if low >= high:
42  8.984 MiB -2491.969 MiB      173834                    return array
43
44  8.984 MiB -2353.484 MiB      164227                pivot = array[high]
45  8.984 MiB -2353.484 MiB      164227                j = low-1
46  8.984 MiB -17830.906 MiB      1247212                    for i in range(low, high):
47  8.984 MiB -15477.453 MiB      1082985                        if array[i] <= pivot:
48  8.984 MiB -8086.094 MiB      562318                            j += 1
49  8.984 MiB -8086.094 MiB      562318                            array[i], array[j] = array[j], array[i]
50  8.984 MiB -2353.547 MiB      164227                    array[high], array[j+1] = array[j+1], array[high]
51  8.984 MiB -2353.609 MiB      164227                    quicksort_inplace(array, low, j)
52  8.984 MiB -2353.688 MiB      164227                    quicksort_inplace(array, j+2, high)
53  8.984 MiB -2353.594 MiB      164227                    return array
```

6

安装pycallgraph出现问题: error in pycallgraph setup command:  
use\_2to3 is invalid.

It looks like setuptools $\geq$ 58 breaks support for use\_2to3



Generated by Python Call Graph v1.0.1  
<http://pycallgraph.slowchop.com>

方法: `pip install setuptools=58`, 在安装。

ans: `fib(0)` gets call 21 times.

pycallgraph 使用方法:

```

19 #!/usr/bin/env python~
18 ~
>> 17 from pycallgraph import PyCallGraph~
>> 16 from pycallgraph.output import GraphvizOutput~
15 ~
14 def mygod():~
13     print(5-2)~
12 ~
11 def f(n):~
10     print(n)~
9     d(2)~
8 ~
7 def d(n):~
6     print(n)~
5     mygod()~
4 ~
3 if __name__ == '__main__':~
2     graphviz = GraphvizOutput()~
1     graphviz.output_file = 'basic2.png'~
20 ...~
1     with PyCallGraph(output=graphviz):~
2         f(9)~

```

7 taskset可现在任务绑定在某个cpu核心，也可以限制资源消耗。

Start a new process with affinity for a single CPU:

```
taskset --cpu-list {{cpu id}} {{command}}
```

ans: taskset --cpu-list 0,2 stress -c 3

8

Ans:

stress -m 3 --vm-bytes 512M

创建3个进程来不停的申请 512M 内存

lssubsys -am

检查设备是否已经挂载，没有就 mount -t

cgroup -o memory mem\_only\_group  
/sys/fs/cgroup/memory

- 先变为root user:

sudo -i

- 在cgroup下创建memory子系统，创建名叫mem\_only\_group节点

cgcreate -g memory:mem\_only\_group

- 设置在memory子系统下的mem\_only\_group的内存大小限制为100MB。

echo 100M > /sys/fs/cgroup/memory/mem\_only\_group/  
memory.limit\_in\_bytes

- 在mem\_only\_group申请111MB内存测试一下，失败。99MB可以！

cgexec -g memory:mem\_only\_group stress -m 1 --vm-bytes 111M

```
root@primary:/sys/fs/cgroup/devices/test# cgexec -g memory:mem_only_group stress -m 1 --vm-bytes 111M
stress: info: [1481] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
stress: FAIL: [1481] (415) <-- worker 1482 got signal 9
stress: WARN: [1481] (417) now reaping child worker processes
stress: FAIL: [1481] (451) failed run completed in 0s
```

```
Last login: Fri Jun 17 10:39:40 2022 from 192.168.64.1
ubuntu@primary:~$ lssubsys -am
cpuset /sys/fs/cgroup/cpuset
cpu,cpuacct /sys/fs/cgroup/cpu,cpuacct
blkio /sys/fs/cgroup/blkio
memory /sys/fs/cgroup/memory ☆
devices /sys/fs/cgroup/devices
freezer /sys/fs/cgroup/freezer
net_cls,net_prio /sys/fs/cgroup/net_cls,net_prio
perf_event /sys/fs/cgroup/perf_event
hugetlb /sys/fs/cgroup/hugetlb
pids /sys/fs/cgroup/pids
rdma /sys/fs/cgroup/rdma
ubuntu@primary:~$
```

参考来源: <https://tech.meituan.com/2015/03/31/cgroups.html>

- Linux中，用户可以使用mount命令挂载 cgroups 文件系统，其中 subsystems 表示需要挂载的 cgroups 子系统 (cpu, cpuset, memory, devices, cpuacct ... )，格式为：

mount -t cgroup -o subsystems name /cgroup/name

- 比如挂载 cpuset, cpu, cpuacct, memory 4个subsystem到/cgroup/cpu\_and\_mem 目录下，就可以使用

```
mount -t cgroup -o remount,cpu,cpuset,memory cpu_and_mem /cgroup/  
cpu_and_mem
```

- 使用`cgcreate`命令的方法创建 `cgroups` 层级结构中的节点。比如通过命令 `cgcreate -t sankuai:sankuai -g cpu:test` 就可以在 `cpu` 子系统下建立一个名为 `test` 的节点。
- `cgset` 命令也可以设置 `cgroups` 子系统的参数，格式为 `cgset -r parameter=value path_to_cgroup`。
- 当需要删除某一个 `cgroups` 节点的时候，可以使用 `cgdelete` 命令，比如要删除上述的 `test` 节点，可以使用 `cgdelete -r cpu:test` 命令进行删除
- 把进程加入到 `cgroups` 子节点也有多种方法，可以直接把 `pid` 写入到子节点下面的 `task` 文件中。也可以通过 `cgclassify` 添加进程，格式为 `cgclassify -g subsystems:path_to_cgroup pidlist`，也可以直接使用 `cgexec` 在某一个 `cgroups` 下启动进程，格式为 `cgexec -g subsystems:path_to_cgroup command arguments`。

实例：

对于这样的“一小份”数据，对及时更新的要求不高，生成商品信息又是一个比较费资源的任务，所以我们把这个任务的cpu资源使用率限制在了50%。

首先在 `cpu` 子系统下面创建了一个 `halfapi` 的子节点：

```
cgcreate abc:abc -g cpu:halfapi
```

然后在配置文件中写入配置数据：

```
echo 50000 > /cgroup/cpu/halfapi/cpu.cfs_quota_us
```

`cpu.cfs_quota_us` 中的默认值是100000，写入50000表示只能使用50%的 `cpu` 运行时间。

最后在这个`cgroups`中启动这个任务：

```
cgexec -g "cpu:/halfapi" php halfapi.php half >/dev/null 2>&1
```