# Linux上机作业 “脚本程序设计”实验报告

## 实验一 生成TCP活动状况报告

### 实验题目

编写shell脚本程序，每隔1分钟生成1行信息：

1. 当前时间；
2. 这一分钟内TCP发送了多少报文；
3. 接收了多少报文；
4. 收发报文总数；
5. 行尾给出符号+或-或空格（+表示这分钟收发报文数比上分钟多10包以上，差别在10包或以内用空格，否则用符号-）。

运行示例如下：

2020-04-17 00:02 345 314 659

2020-04-17 00:03 1252 1100 2352 +

2020-04-17 00:04 714 570 1284 -

2020-04-17 00:05 151 139 290 -

2020-04-17 00:06 1550 1097 2647 +

2020-04-17 00:07 1385 959 2344 -

2020-04-17 00:08 5 1 6 -

2020-04-17 00:09 5 1 6

2020-04-17 00:10 837 723 1560 +

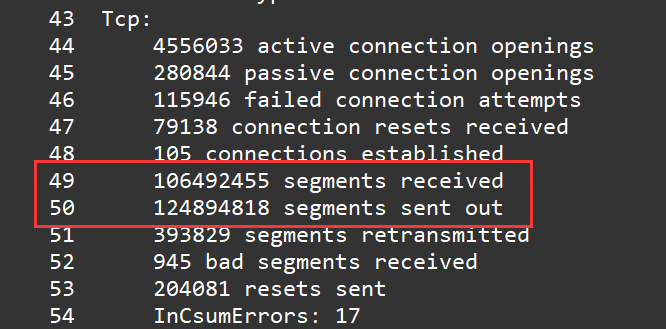
2020-04-17 00:11 22 22 44 -

### 实验分析

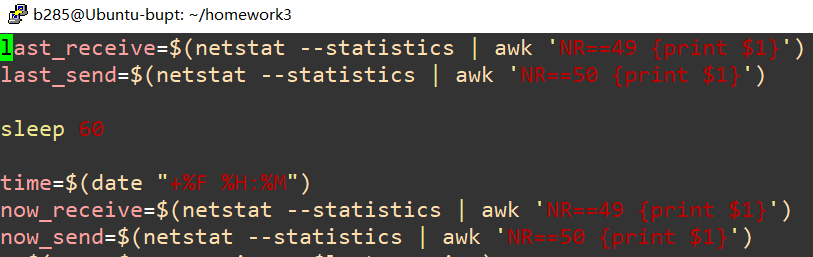
1. 观察实验给出的输出结果，第一行输出最后不需要添加符号，或添加空格。
2. 题目要求每分钟记录一次，并且行尾需要给出+或-或空格，因此程序主体应当为循环和分支结构。

### 实验步骤

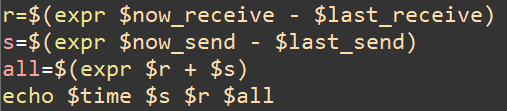
1. 使用netstat --statistics | cat -n命令查看需要的TCP活动状况，以及我们需要的收发情况，分别在第49行和第50行：



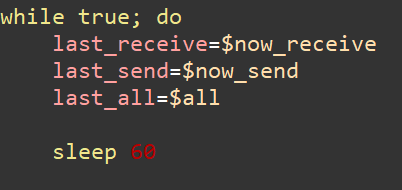
1. 将上述两行可以通过awk指令取出第一列的两个数字，作为变量保存（需要使用NR代表文本行号）；等待60s可以使用sleep命令；而时间信息可以通过date指令获得，date指令后面添加以“+”为首的字符串可以控制输出时间的格式与题目要求一致：



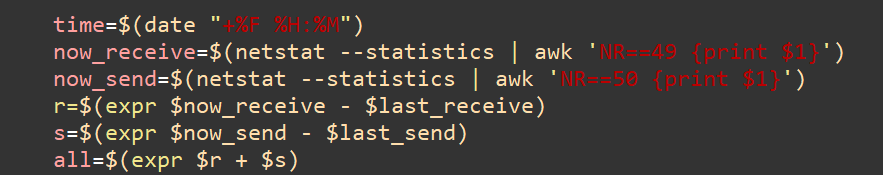
1. 获得了间隔一分钟的收发报文数量，对应的值相减可以得到这一分钟内的收发报文情况，而得到的两个结果相加可以得到这一分钟总的收发情况：



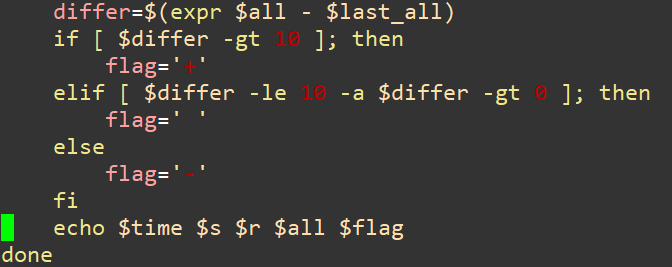
1. 这样就得到了这一分钟内的结果，使用echo将结果依次打印出来。由于这是第一分钟内的情况，它最后没有“+”“-”或空格，所以单独分开来，接下来需要对每一分钟进行判断和记录，需要使用循环结构，循环内部的整体思路和第一分钟的思路大致相同，首先等待60s之前应该记录上一分钟的收发情况：



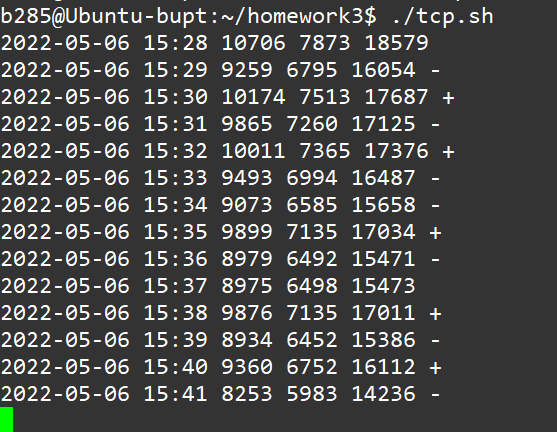
1. 等待过后需要对目前的收发报文情况进行记录，并与一分钟前的记录进行相减：



1. 最后的符号判断需要使用分支结构，只需要比较这一分钟的收发报文总数和上一分钟的即可，-gt表示大于等于，-le表示小于，最后使用echo输出结果：



1. tcp.sh文件的部分执行结果为：



### 实验总结

本次实验主要让我实际运用了循环与分支结构，复习了awk指令的使用，以及脚本程序编写的一些规范，例如使用变量的时候需要在前面加上$，很多时候没有特别注意这么做而使程序崩溃，还有很多指令的规范书写，if条件的书写需要在中括号内输入空格，这些细小的点如果一开始不注意后面很难发现错误，所以该加的符号一定要加上，在此基础上才能进而保证程序的逻辑正确。

### 源代码

|  |
| --- |
| 1. last\_receive=$(netstat --statistics | awk 'NR==49 {print $1}') 2. last\_send=$(netstat --statistics | awk 'NR==50 {print $1}') 3. sleep 60 4. # 第一分钟的输出 5. time=$(date "+%F %H:%M") 6. now\_receive=$(netstat --statistics | awk 'NR==49 {print $1}') 7. now\_send=$(netstat --statistics | awk 'NR==50 {print $1}') 8. r=$(expr $now\_receive - $last\_receive) 9. s=$(expr $now\_send - $last\_send) 10. all=$(expr $r + $s) 11. echo $time $s $r $all 12. # 之后每一分钟的输出 13. while true; do 14. last\_receive=$now\_receive 15. last\_send=$now\_send 16. last\_all=$all 17. sleep 60 18. time=$(date "+%F %H:%M") 19. now\_receive=$(netstat --statistics | awk 'NR==49 {print $1}') 20. now\_send=$(netstat --statistics | awk 'NR==50 {print $1}') 21. r=$(expr $now\_receive - $last\_receive) 22. s=$(expr $now\_send - $last\_send) 23. all=$(expr $r + $s) 24. differ=$(expr $all - $last\_all) 25. if [ $differ -gt 10 ]; then 26. flag='+' 27. elif [ $differ -le 10 -a $differ -gt 0 ]; then 28. flag=' ' 29. else 30. flag='-' 31. fi 32. echo $time $s $r $all $flag 33. done |

## 实验二 下载bing图库中图片

### 实验题目

编写脚本程序bing.sh，将https://bing.ioliu.cn/?p=36图库中照片全部下载下来存放到本地bing目录，上面URL中p=36可以换成p=126可访问126号页面，每页有12个图，每个图的日期，中文说明信息和下载地址及文件名html文件中可提取。要求下载后的文件命名为“日期 说明.jpg”例如：

2019-08-03 野花草甸上的一只欧亚雕鸮，德国莱茵兰-普法尔茨.jpg

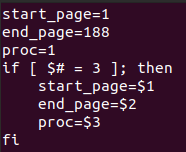
### 实验要求

1. 不重复下载已下载的图片
2. 考虑并发
3. 考虑下载文件出现故障的情况

如果一个图片有5MB，接收1.5MB后网络断开，则残存一个不完整的图片文件。避免这种现象发生的一种方法是wget下载时使用一个临时文件名。判断wget是否成功，若成功则将文件改名为正式名称；若失败，删除临时文件。临时文件名的选取要考虑的并发问题，至少不可以程序中写死一个文件名导致两进程因使用相同的临时文件名而失败。

### 实验步骤

1. 首先需要设置抓取的页面范围，同时获取参数，如果参数数量为3，那么需根据给出的参数来确定页面范围：



1. 根据需要，之后要创建文件夹和文件共四个，分别是bing文件夹，page.txt、image.txt、loaded\_image.txt三个文件；

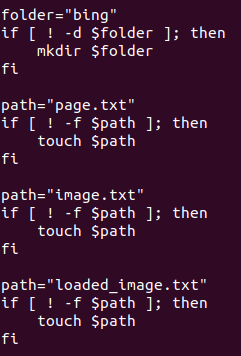
其中：

bing文件夹用来存放下载的图片；

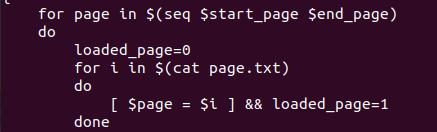
page.txt文件用来存放已经加载过的页面；

image.txt用来存放需要下载的图片信息；

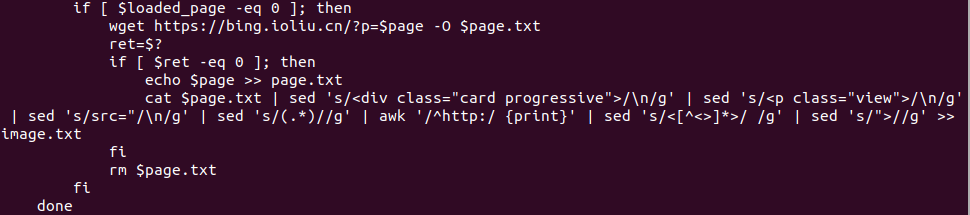
loaded\_image.txt用来存放已经下载过的图片信息。

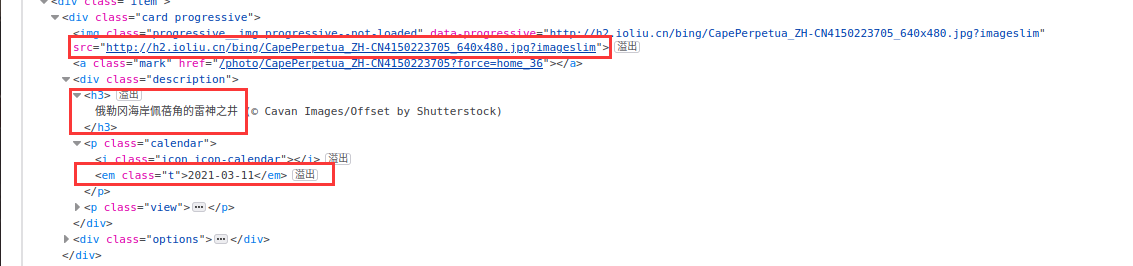


1. 首先需要进行页面的抓取，遍历整个需要抓取的页面范围，由于已经抓取的页面号记录在page.txt中，因此需要找到page.txt中没有的页面号进行抓取即可（如果loaded\_page=0说明没有抓取页面page）：



1. 如果wget抓取成功则需要对整个页面进行正则处理，取出我们需要的信息保存在image.txt中，抓取失败则删除刚刚的临时文件。





最后在image.txt中应该是如下效果：



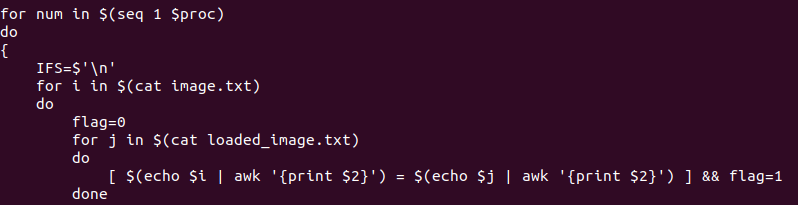
1. 下载图片时采用并发的概念使用如下结构：

|  |
| --- |
| for num in $(seq 1 $proc)  do  {}&  done |

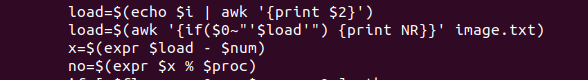
这样每一个循环实际上都是在后台进行，因此达到了并发的目的，下载图片的程序需要写在{}内部。

1. 之后需要对image.txt 文件中的每一个图片进行下载，和下载html一样，遍历image.txt时需要对loaded\_image.txt文件进行查看，找到loaded\_image.txt中没有的图片进行下载，如果下载成功则将这个图片记录写入 loaded\_image.txt中，并将临时文件改名，如果下载失败则删除临时文件：

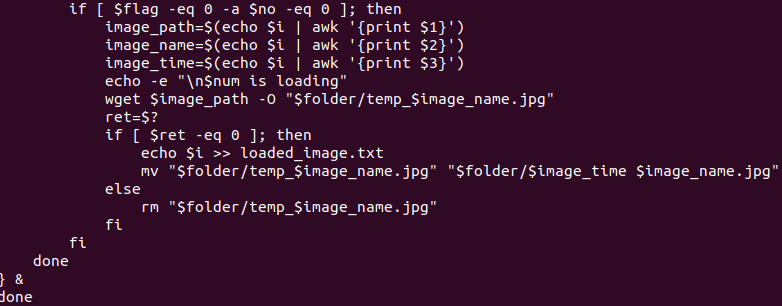
判断图片是否已经下载：



接下来为了确保每一个进程下载不同的图片，故需要给每一个进程分配其需要下载哪一行的图片，这里采用“进程号+n\*进程数”的规则，如果是一号线程，且一共有三个线程，那么一号线程需要下载第1，4，7，……行的图片。



接下来只有满足行号的要求且图片未被下载过才可以下载图片（需要判断wget是否成功）：

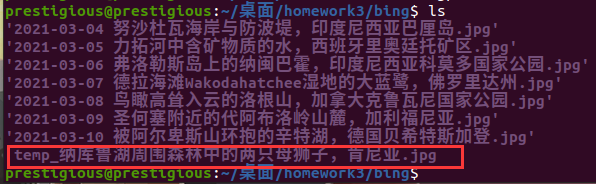


### 实验结果

1. 使用./bing.sh 36 36 1命令，表示启动1个进程下载第36页的图片。下载至一半停止：



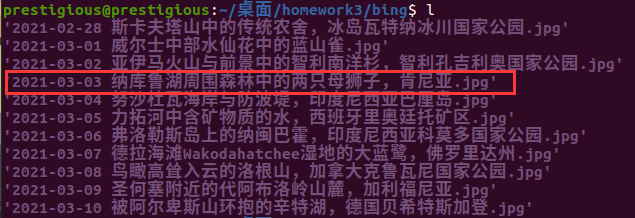
可以看到bing文件夹中存在临时文件“temp\_”：



当我们再次使用./bing.sh 36 36 1时：



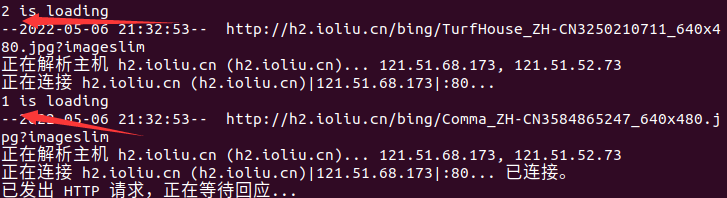
可以看到从这个临时文件开始下载之后的图片了，并且执行完毕后临时文件不见了，只剩下完整的图片了：



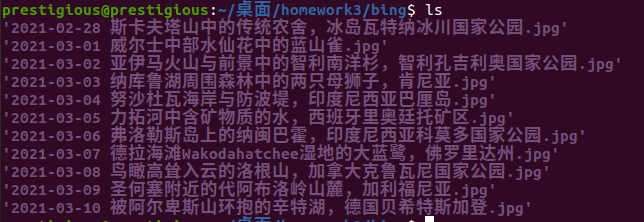
1. 使用./bing.sh 36 36 3表示启动三个进程。







可以同时看到三个进程分别下载图片，输出不规律说明它们其实是同时在进行的，最后bing文件夹内的图片也和之前相同：



### 实验总结

本次实验总体来说考察的时脚本程序逻辑结构的编写，而写作规范已经成为必备技能，总体来说包括对于变量和指令的灵活调用，对文件的读写等等，训练了抓取网页数据的一些技能，可以说收获很大。

第一天还能正常访问图片页面，第二天写报告的时候就不能打开这个网页了，但不是被禁了，几个小时可能能打开一次，不知道之后会不会好转。

### 源代码

|  |
| --- |
| 1. start\_page=1 2. end\_page=188 3. proc=1 4. if [ $# = 3 ]; then 5. start\_page=$1 6. end\_page=$2 7. proc=$3 8. fi 9. # 保存图片 10. folder="bing" 11. if [ ! -d $folder ]; then 12. mkdir $folder 13. fi 14. # 保存已经加载的页号 15. path="page.txt" 16. if [ ! -f $path ]; then 17. touch $path 18. fi 19. # 保存需要下载的图片 20. path="image.txt" 21. if [ ! -f $path ]; then 22. touch $path 23. fi 24. # 保存已经下载的图片 25. path="loaded\_image.txt" 26. if [ ! -f $path ]; then 27. touch $path 28. fi 29. # 下载每一个页面并提取图片信息 30. for page in $(seq $start\_page $end\_page) 31. do 32. loaded\_page=0 33. for i in $(cat page.txt) 34. do 35. [ $page = $i ] && loaded\_page=1 36. done 37. if [ $loaded\_page -eq 0 ]; then 38. wget https://bing.ioliu.cn/?p=$page -O $page.txt 39. ret=$? 40. if [ $ret -eq 0 ]; then 41. echo $page >> page.txt 42. cat $page.txt | sed 's/<div class="card progressive">/\n/g' | sed 's/<p class="view">/\n/g' | sed 's/src="/\n/g' | sed 's/(.\*)//g' | awk '/^http:/ {print}' | sed 's/<[^<>]\*>/ /g' | sed 's/">//g' >> image.txt 43. fi 44. rm $page.txt 45. fi 46. done 47. # 启动线程，并下载图片 48. for num in $(seq 1 $proc) 49. do 50. { 51. IFS=$'\n' 52. for i in $(cat image.txt) 53. do 54. flag=0 55. for j in $(cat loaded\_image.txt) 56. do 57. [ $(echo $i | awk '{print $2}') = $(echo $j | awk '{print $2}') ] && flag=1 58. done 59. # 需要判断某行的图片是不是该线程需要下载的 60. load=$(echo $i | awk '{print $2}') 61. load=$(awk '{if($0~"'$load'") {print NR}}' image.txt) 62. x=$(expr $load - $num) 63. no=$(expr $x % $proc) 64. if [ $flag -eq 0 -a $no -eq 0 ]; then 65. image\_path=$(echo $i | awk '{print $1}') 66. image\_name=$(echo $i | awk '{print $2}') 67. image\_time=$(echo $i | awk '{print $3}') 68. echo -e "\n$num is loading" 69. wget $image\_path -O "$folder/temp\_$image\_name.jpg" 70. ret=$? 71. if [ $ret -eq 0 ]; then 72. echo $i >> loaded\_image.txt 73. mv "$folder/temp\_$image\_name.jpg" "$folder/$image\_time $image\_name.jpg" 74. else 75. rm "$folder/temp\_$image\_name.jpg" 76. fi 77. fi 78. done 79. } & 80. done 81. wait 82. rm page.txt 83. rm image.txt 84. rm loaded\_image.txt |