Admin relevant selection of examples…

- Merely examples,
  most are sh (bash compatible), some are Unix rather than Linux
- The printed page refuses nothing – taken from textbooks
  - Cobbled from other scripts rather than clean design!?

- Don’t trust every programmed script.
- Test everything…use what’s good!
- Their (problem | situation |environment |configuration| version) may be quite different
  - Probably a panic to meet publishing house deadlines!!!

- The first few were adapted from Unix Hints & Hacks, others from Linux Shell Scripting Cookbook, 2nd Ed.
- Still fairly basic, simple styles… somewhat convoluted…!

Storage stories

Over time, log files grow, users never clean up their files, spooling directories fill, system files and process occasionally go crazy and fill up a **disk**. Some really good areas to monitor on a system **disk** are

- Spooling directories—/var/spool
- Log files—/var/adm or /var/logs
- User home directories—/home or /usr/people
- Temporary areas—/tmp or /usr/tmpd
- Nonjournal filesystems—/lost+found

On systems where there are multiple drives and partitions, watch for the following areas:

- Temporary areas
- Spooling areas that are linked from /var/spool
- Fully compiled source directories—/local/src
- Old versions of third-party software not being used

Not used for assignments, since too much info, and access limitations

The (rather extreme) personal touch…!

Sometimes people don’t realise how much they use, or need encouragement to be ‘good citizens’:-

```
du -s /users | sort -rn | head -10 | \
mail -s "Disk Hogs”  everyone@madmen.com
```

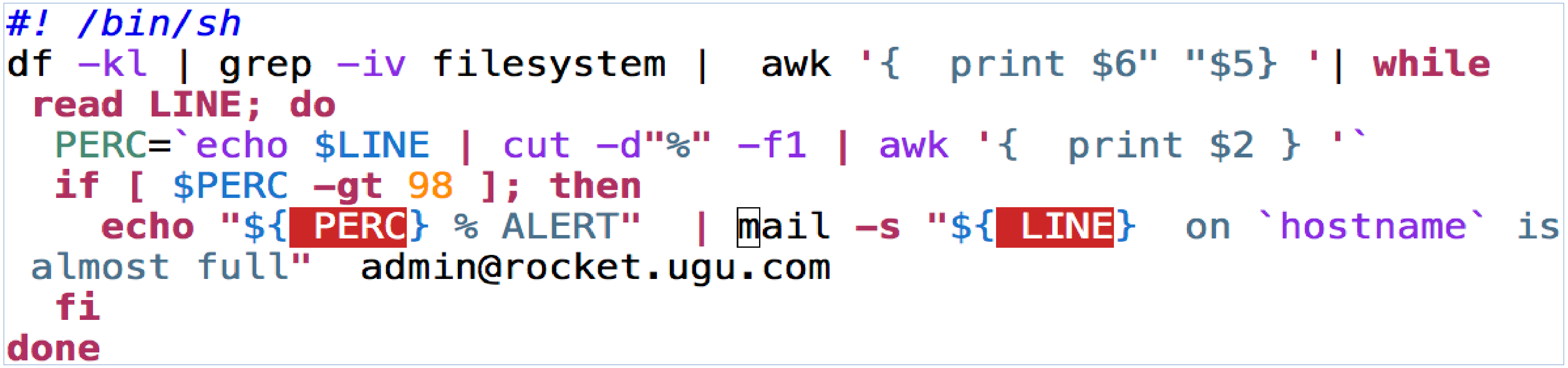
Disk use –s show for each specified,

(need to have admin rights to check others filesystems)

Sort in reverse numerical order, taking the ‘Top 10’

And mailing to everyone with subject “Disk Hogs”

Disk device nearly full!



```
#! /bin/sh
df -kl | grep -iv filesystem |  awk '{  print $6" "$5} '| while
 read LINE; do
  PERC=`echo $LINE | cut -d"%" -f1 | awk '{  print $2 } '`
  if [ $PERC -gt 98 ]; then
    echo "${ PERC} % ALERT"   | mail -s "${ LINE}  on `hostname` is
 almost full"  admin@rocket.ugu.com
  fi
done
```

The long way around.. just to get the number before the % sign; use regex, awk, sed, cut etc.

df –kl display free disk space, in k, and locally mounted only..e.g.

```
Filesystem   1024-blocks      Used Available Capacity  Mounted on
  /dev/disk0s2   488050672 305645696 182148976    63%    /
```

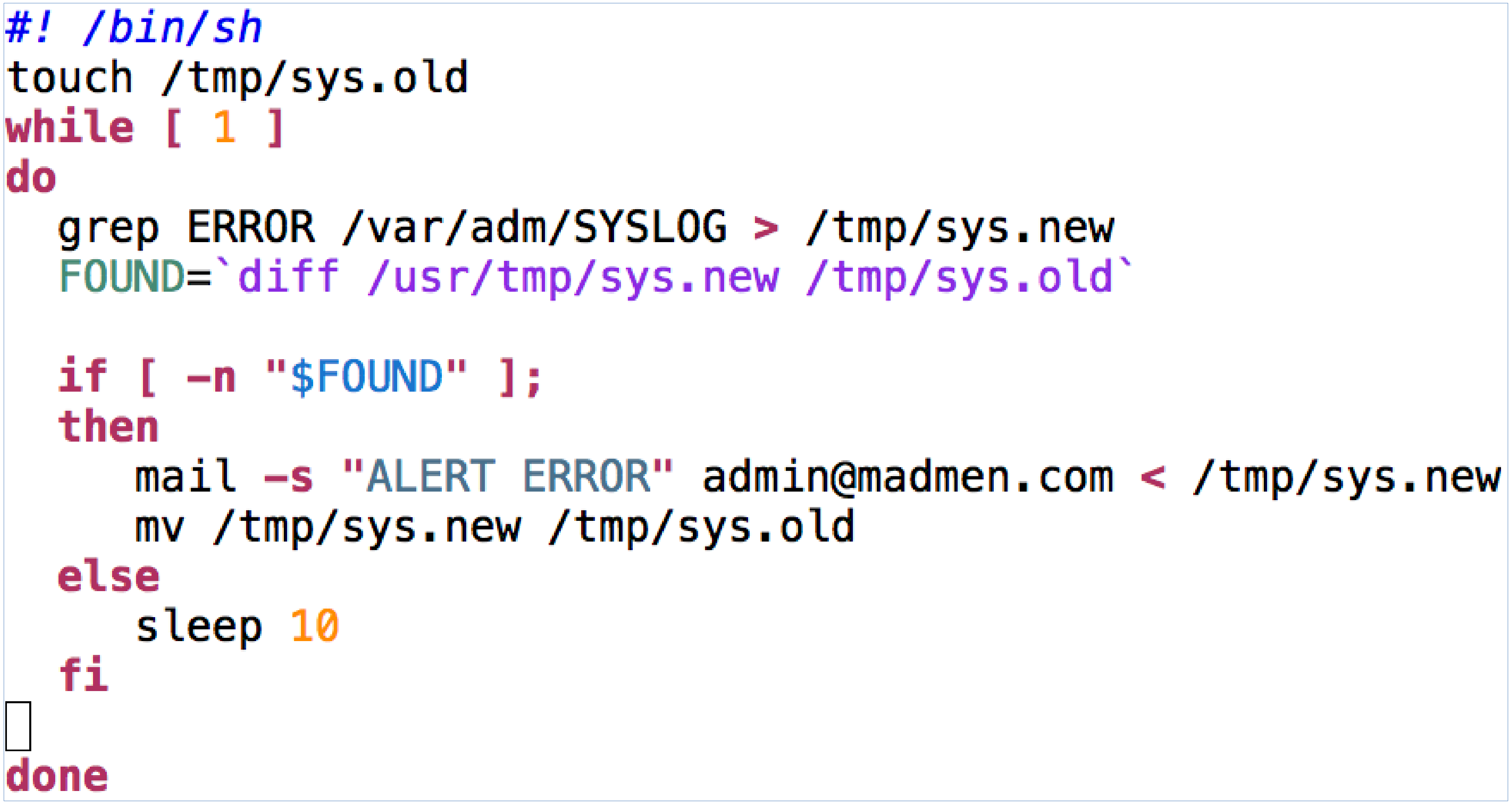
grep –<u>i</u>gnore case, an in<u>v</u>erted match to 'filesystem' (i.e. cut the header line with filesystem)
awk prints 6th (mount point) and 5th (% used) fields separated by a space, to give: 63%
… and pipe all that stuff to a loop, which keeps reading, while there is a line to read…
... echoes the line through a pipe to cut field1, (**-f1**) to the field delimit character (**-d"%"**)
…(a perverse cut, d is not for delete, but denoting '%' as the delimit char for field1 = / 63)
... pipes that to awk, just to extract field 2, now the numeric value of the %, without the '%'!

It then sends an alert if this value is –gt than 98, which is dangerously high.

Perverse, & inefficient script example from a book, to show how to 'bash' them together!?

Log monitoring script – alert on new error!

Grep finds lines with ERROR in the logfile (choose one to monitor) & writes to temporary file
Which is compared with the previous copy for differences, which are notified to admin@...
The changed file then becomes the new standard for comparison with any further ERRORs!

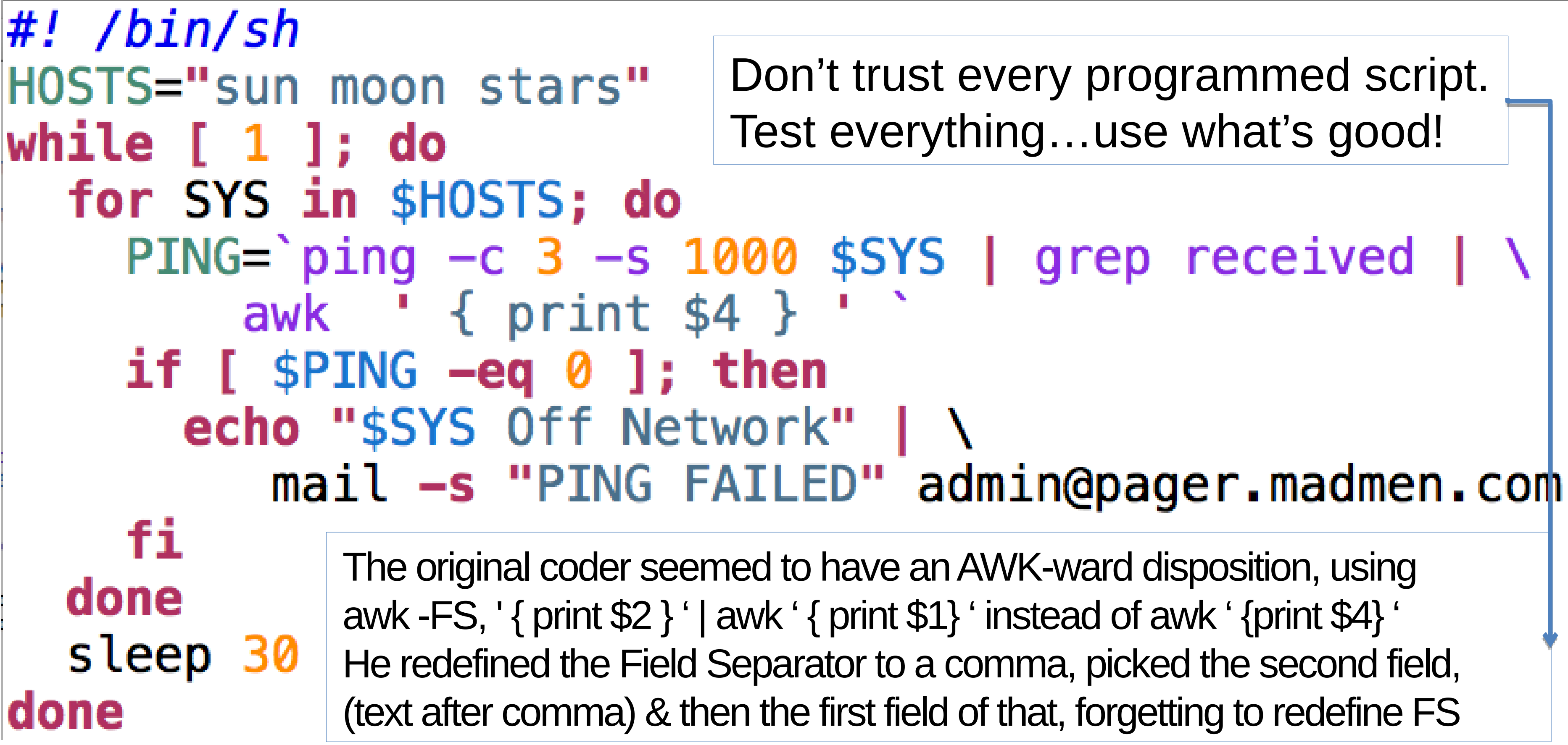


```
#! /bin/sh
touch /tmp/sys.old
while [ 1 ]
do
  grep ERROR /var/adm/SYSLOG > /tmp/sys.new
  FOUND=`diff /usr/tmp/sys.new /tmp/sys.old`

  if [ -n "$FOUND" ];
  then
     mail -s "ALERT ERROR" admin@madmen.com < /tmp/sys.new
     mv /tmp/sys.new /tmp/sys.old
  else
     sleep 10
  fi

done
```

Pinger panic!



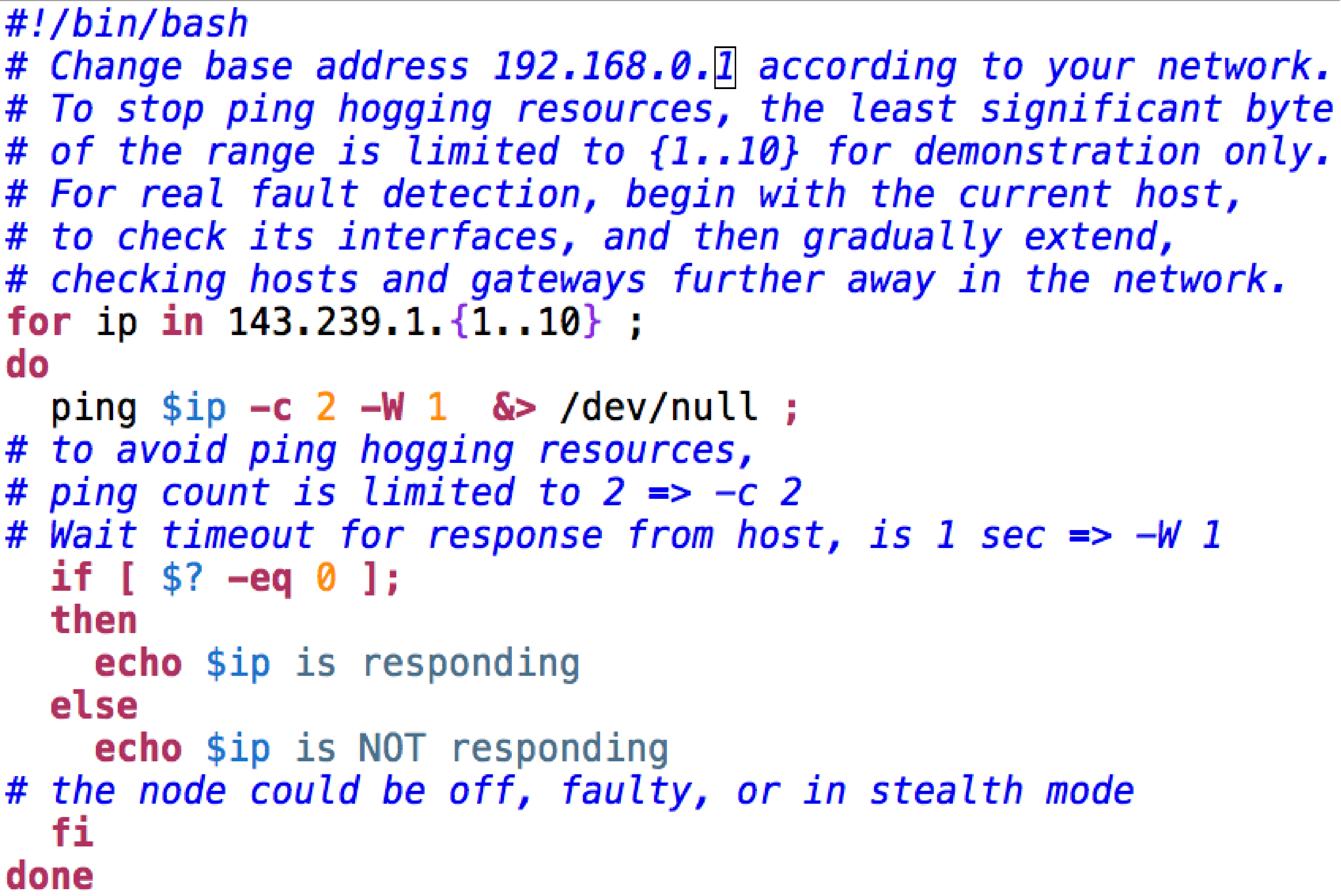
```
#! /bin/sh
HOSTS="sun moon stars"
while [ 1 ]; do
  for SYS in $HOSTS; do
    PING=`ping -c 3 -s 1000 $SYS | grep received | \
        awk  ' { print $4 } ' `
    if [ $PING -eq 0 ]; then
      echo "$SYS Off Network" | \
        mail -s "PING FAILED" admin@pager.madmen.com
    fi
  done
  sleep 30
done
```

Don’t trust every programmed script.
Test everything…use what’s good!

The original coder seemed to have an AWK-ward disposition, using awk -FS, ' { print $2 } ' | awk ' { print $1} ' instead of awk ' {print $4} '
He redefined the Field Separator to a comma, picked the second field, (text after comma) & then the first field of that, forgetting to redefine FS

It repeatedly pings through hosts defined in a string (could also be obtained from a file), waits 30 seconds and repeats the process, alerting the admin on a pager on host failure.
To save time and resources with ping, packet count is limited to 3 (-c 3), with fairly large packets (-s 1000 (bytes)) which should uncover network comms issues. Finally, the response is cut to the number of packets returned, which if 0, causes an alert to be mailed to admin.

Neater pinger - check exit status of ping – no awkward stuff!



```bash
#!/bin/bash
# Change base address 192.168.0.1 according to your network.
# To stop ping hogging resources, the least significant byte
# of the range is limited to {1..10} for demonstration only.
# For real fault detection, begin with the current host,
# to check its interfaces, and then gradually extend,
# checking hosts and gateways further away in the network.
for ip in 143.239.1.{1..10} ;
do
  ping $ip -c 2 -W 1  &> /dev/null ;
# to avoid ping hogging resources,
# ping count is limited to 2 => -c 2
# Wait timeout for response from host, is 1 sec => -W 1
  if [ $? -eq 0 ];
  then
    echo $ip is responding
  else
    echo $ip is NOT responding
# the node could be off, faulty, or in stealth mode
  fi
done
```

Top 10 Processes over an interval

```
#!/bin/bash
SECS=60                         # Change SECS to overall total monitoring time
UNIT_TIME=6                     # UNIT_TIME is the interval between each sampling
STEPS=$(( $SECS / $UNIT_TIME )) # bash converts and divides numbers as strings
echo Watching CPU usage... ;
for((i=0;i<STEPS;i++))
do
  ps -eocomm,pcpu | \
        tail -n +2 >> /tmp/cpu_usage.$$
  sleep $UNIT_TIME
done
echo CPU eaters :
cat /tmp/cpu_usage.$$ | awk ' { process[$1]+=$2; }
END{ for(i in process) printf("%-20s %s\n",i, process[i]) ; }' | sort -nrk 2 | head
rm /tmp/cpu_usage.$$            # Remove the temporary log file
```

top & htop (interactive) are alternative builtin dynamic systems monitoring programs, but their text is more difficult to process as their dynamic screen rewrite uses weird non-printing characters

ps, gives process and % CPU use, runs every UNIT_TIME, with header record ((first line) stripped off, (tail –n + 2 => tail of the file starting at line number 2), appended to the temporary logfile, for this process, identified by $$. This file is then analysed by awk to accumulate the % CPU use from each ps run, in an array / list element for each process.

Finally, the for(i in process) prints the array index (the process name), followed by the cumulative % CPU use over the hour, which is piped to the shell sort command (-nrk 2 = numeric, reverse (descending order) on field 2) which outputs the top 10 using the default listlength for head; 10.   Finally the temporary log file is removed.