The Opinionated Guide to awk

2022-05-14

Most tools in the UNIX kit have a default output format that targets users and not scripts. ‘awk‘ provides general translation services, ie. converting output from one form to another.

Re-formatting ‘vmstat‘

‘vmstat‘ will output in unfriendly units and break its columnization.

```
vmstat 1 | awk '
!/^procs/{
  if ($1 == "r") {
    buff = "buff"; free = "free"; cache = "cache";
  } else {
    free = int($4 / (1000 * 1000));
    buff = int($5 / 1000);
    cache = int($6 / (1000 * 1000));
  }
  a = sprintf("%-2s %-2s %-4s %-4s ", $1,  $2,  $3,  free);
  b = sprintf("%-4s %-5s %-3s %-3s ", buff, cache, $7, $8 );
  c = sprintf("%-3s %-3s %-4s %-4s ", $9, $10, $11, $12 );
  d = sprintf("%-3s %-3s %-3s %-3s",  $13, $14, $15, $16 );
  print a b c d;
}
'
```

Which then outputs:

```
r  b  swpd free buff cache si  so  bi  bo  in   cs   us  sy  id  wa
1  0  0    14   287  1     0   0   0   0   0    1    20  0   80  0
0  0  0    14   287  1     0   0   0   0   51   55   0   0   100 0
0  0  0    14   287  1     0   0   0   0   19   18   0   0   100 0
0  0  0    14   287  1     0   0   0   0   29   40   0   0   100 0
0  0  0    14   287  1     0   0   0   0   23   24   0   0   100 0
```

In a nutshell, most of the time ‘awk‘ is invoked for one-liners that just select a particular column or pretty-prints output as the above does for ‘vmstat‘. However, sometimes a sledgehammer is needed, and that is when ‘awk‘ excels.

Summarization via Histograms

Make a histogram where keys are binned into ROWS bins, and values are scaled for 0 to COLS

```
#!/usr/bin/awk -f
BEGIN {
  if (! ROWS) { ROWS = 25; }
  if (! COLS) { COLS = 50; }
  ROWS -= 1;
}
function bin(n)   { return int(n / (max_k - min_k) * ROWS); }
function unbin(n) { return int(n * (max_k - min_k) / ROWS); }
function scale(n) { return int(n /  max_v * COLS); }
{
  keys[ int($1) ] += $2;
  max_k = $1;
  if ( NR == 1 ) min_k = $1;
}
END {
# pre-zero the bins, otherwise empties are skipped
  step = ( max_k - min_k ) / ROWS;
  for ( k = min_k; k < max_k; k += step ) bins[ bin(k) ] = 0;

  for ( k in keys ) bins[ bin(k) ] += keys[k];

  for ( k in bins ) if ( bins[k] > max_v ) max_v = bins[k];

  for ( k in bins ) {
    printf("%-5d ", unbin(k));
    printf("%5d ", bins[k]);
    s = scale( bins[k] );
    for ( i = 0; i < s ; i++ ) {
      printf("#");
    }
    printf("\n");
  }
}
```

nb. the script depends on the input being sorted.

eg. to show the distribution of line-lengths for some awk scripts:

```
cat *.awk | ./columns.awk | ./histo.awk
0        39 ##################################################
3        10 ############
7         9 ###########
11       12 ###############
15       12 ###############
19       14 #################
23       18 #######################
27        5 ######
31        7 ########
35       18 #######################
39        3 ###
43        6 #######
47        6 #######
51        6 #######
55        6 #######
59        5 ######
63        1 #
```

```
67         1 #
71         1 #
75         4 #####
79         0
83         0
87         1 #
91         1 #
95         1 #
```

where ‘columns.awk‘ is just

```
#!/usr/bin/awk -f
# read input, count line lengths.
{
  lens[ length($0) ]++;
}
END {
  for (i in lens) {
    printf("%s %s\n", i, lens[i]);
  }
}
```

Summarization for tcpdump

‘tcpdump‘ will output packets as they arrive. On a busy server, it can be difficult to follow a TCP flow.

```
tcpdump -t -nn -l -i eth0 -s 0 -x tcp port 25
IP 10.0.2.26.35604 > 10.0.2.9.25: Flags [R], seq 3661213342, win 0, length 0
        0x0000:  4500 0028 0000 4000 4006 22ae 0a00 021a
        0x0010:  0a00 0209 8b14 0019 da39 ae9e 0000 0000
        0x0020:  5004 0000 83b8 0000
IP 10.0.2.9.25 > 10.0.2.26.35604: Flags [F.], seq 2016, ack 1742, win 252, options [nop,nop,TS
val 2329008704 ecr 2328771435], length 0
        0x0000:  4500 0034 d683 4000 4006 4c1e 0a00 0209
        0x0010:  0a00 021a 0019 8b14 1574 178c da39 ae9e
        0x0020:  8011 00fc f14b 0000 0101 080a 8ad1 da40
        0x0030:  8ace 3b6b
IP 10.0.2.26.35604 > 10.0.2.9.25: Flags [R], seq 3661213342, win 0, length 0
        0x0000:  4500 0028 0000 4000 4006 22ae 0a00 021a
        0x0010:  0a00 0209 8b14 0019 da39 ae9e 0000 0000
        0x0020:  5004 0000 83b8 0000
```

To instead output transaction by transaction, pipe the hex output of ‘tcpdump‘ into the following script, where lines are processed in a ‘sed‘-like fashion, ie. accumulate the hex data line by line in a ‘hold‘, and then process that ‘hold‘ when the header of a new packet is seen.

TCP SYN and RESET packets are discarded, FIN packets mean the transaction is complete and can be printed. Otherwise, convert the hex to ascii and append it to the ‘pkts‘ hash keyed by the remote IP address and port.

```
#!/usr/bin/awk -f
# OFFSET is 52 for IPv4 + Timestamp Option + TCP
BEGIN {
  if (! TARGET) { TARGET = "10.0.2.26.25"; }
  if (! REGEXP) { REGEXP = "552 5.0.0 Headers too large"; }
  if (! DELIM ) { DELIM  = "##### TRANSACTION #####"; }
  if (! OFFSET) { OFFSET = 52; }
  FLAGS = "[S";
}

function unhex(hex,    c) {
  c = strtonum("0x" hex);
  return ( c == 0x0a || (c >= 0x20 && c < 0x7f) ) ? sprintf("%c", c) : " ";
}

function process(hold,    i, n) {
  if ( substr(FLAGS, 2, 1) == "S" ) { return; }
  if ( substr(FLAGS, 2, 1) == "R" ) { return; }
  if ( substr(FLAGS, 2, 1) == "F" ) {
    if ( ! (KEY in pkts) ) { return; }
    if ( match( pkts[KEY], REGEXP ) ) {
      printf("%s\n%s\n", DELIM, pkts[KEY]);
    }
    delete pkts[KEY];
    return;
  }
  gsub(/[a-f0-9][a-f0-9]/, "& ", hold);
  n = split(hold, fields);
  for (i=OFFSET+1; i <= n; i++) {
    pkts[KEY] = pkts[KEY] unhex(fields[i]);
  }
}

/^IP/ {
  process(hold); hold = "";
  gsub(/:/, "", $0);
  src = $2 ; dst = $4; FLAGS = $6;
  KEY = (src == TARGET) ? dst : src;
}
$1 ~ /0x/ {
  $1 = "";
  hold = hold " " $0;
}
```

nb. the options required are `-t -l -s 0 -x` which drop the timestamp, buffer output by line, grab the entire packet, and output its contents in hex. ie. invoke as follows:

```
tcpdump -t -nn -l -i eth0 -s 0 -x tcp port 25 | awk -f tcpdump-follow.awk -v REGEXP=.
```

Conversion of base-64

Non-ascii email content will typically be base-64 encoded.

```
#!/usr/bin/awk -f
BEGIN {
  base64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
}
function b64_val(c,  r) {
  r = index(base64, c);
  return r > 0 ? r - 1 : 0;
}
function b64_decode(str,  i,  a,  b,  c,  d,  ret) {
  ret = "";
  split(str, strs, "");
  for (i = 1; i < length(str); i += 4) {
    a = b64_val(strs[i])
      b = b64_val(strs[i+1])
      c = b64_val(strs[i+2])
      d = b64_val(strs[i+3])
      ret = sprintf("%s%c", ret, lshift(a,2) + rshift(b,4))
      ret = sprintf("%s%c", ret, lshift(and(15,b), 4) + rshift(c,2))
      ret = sprintf("%s%c", ret, lshift(and(3,c), 6) + d)
  }
  return ret
}
```

Condensing OpenBSD reports

OpenBSD has a report '/etc/daily' which shows the load average, disk usage, and network counters for interfaces.

```
 1:30AM  up 5 days, 11 hrs, 1 user, load averages: 0.08, 0.02, 0.01

Checking subsystem status:

disks:
Filesystem  1K-blocks      Used     Avail Capacity iused   ifree  %iused  Mounted on
/dev/sd0a    27982348   3843354  22739878    14%   44743 3541047     1%   /
mfs:52090      252383     38788    200976    16%     803   76379     1%   /var
mfs:81761        4399        32      4148     1%    1231    3887    24%   /dev
mfs:38157        9839      5164      4184    55%     452    2106    18%   /etc

Last dump(s) done (Dump '>' file systems):

network:
Name    Mtu   Network     Address              Ipkts Ifail    Opkts Ofail Colls
lo0     32768 <Link>                               6     0        6     0     0
lo0     32768 ::1/128     ::1                      6     0        6     0     0
lo0     32768 fe80::%lo0/64 fe80::1%lo0              6     0        6     0     0
lo0     32768 127/8       127.0.0.1                6     0        6     0     0
em0     1500  <Link>      00:0d:b9:52:0b:48  6152773     0     7272     0     0
em0     1500  10.0.0/23   10.0.0.3           6152773     0     7272     0     0
em1     1500  <Link>      00:0d:b9:52:0b:49 32510605     0  5045807     1     0
em1     1500  172.31.255.254/30 172.31.255.254    32510605     0  5045807     1     0
em2     1500  <Link>      00:0d:b9:52:0b:4a   164426     0    78059     0     0
em2     1500  180.255.66.50/30 180.255.66.50       164426     0    78059     0     0
em3     1500  <Link>      00:0d:b9:52:0b:4b        0     0        0     0     0
enc0*   0     <Link>                               0     0        0     0     0
carp1   1500  <Link>      00:00:5e:00:01:01   752877     0        0     0     0
carp1   1500  10.0.0/23   10.0.0.1            752877     0        0     0     0
carp2   1500  <Link>      00:00:5e:00:01:02   438078     0        0     0     0
carp2   1500  10.0.2/23   10.0.2.1            438078     0        0     0     0
carp3   1500  <Link>      00:00:5e:00:01:03   629885     0        0     0     0
carp3   1500  10.0.4/23   10.0.4.1            629885     0        0     0     0
carp4   1500  <Link>      00:00:5e:00:01:04  2397755     0        0     0     0
carp4   1500  10.0.6/23   10.0.6.1           2397755     0        0     0     0
carp5   1500  <Link>      00:00:5e:00:01:05  1408828     0        0     0     0
carp5   1500  10.0.8/22   10.0.8.1           1408828     0        0     0     0
carp6   1500  <Link>      00:00:5e:00:01:06   394715     0        0     0     0
carp6   1500  10.0.12/23  10.0.12.1           394715     0        0     0     0
pfsync0 1500  <Link>                        32367006     0  4792310     0     0
vlan2   1500  <Link>      00:0d:b9:52:0b:48   438345     0      230     0     0
vlan2   1500  10.0.2/23   10.0.2.3            438345     0      230     0     0
vlan3   1500  <Link>      00:0d:b9:52:0b:48   630865     0      299     0     0
vlan3   1500  10.0.4/23   10.0.4.3            630865     0      299     0     0
vlan4   1500  <Link>      00:0d:b9:52:0b:48  2522692     0     3636     0     0
vlan4   1500  10.0.6/23   10.0.6.3           2522692     0     3636     0     0
vlan5   1500  <Link>      00:0d:b9:52:0b:48  1410613     0     1111     0     0
vlan5   1500  10.0.8/22   10.0.8.3           1410613     0     1111     0     0
vlan6   1500  <Link>      00:0d:b9:52:0b:48   394720     0        6     0     0
vlan6   1500  10.0.12/23  10.0.12.3           394720     0        6     0     0
wg0     1420  <Link>                            9608     0     9002     0     0
wg0     1420  172.100.101.3/29 172.100.101.3         9608     0     9002     0     0
pflog0  33136 <Link>                               0     0  2973112     0     0
```

Nobody actually wants to slog through this every day, so let awk handle it (via procmail), ie. define some thresholds in a BEGIN section, and then only print the relevant parts when those thresholds are exceeded.

```
#!/usr/bin/awk -f

BEGIN { max_disk = 0.950 ; min_ifree = 2000; max_load = 1.0; }

/^$/ { is_disk = 0; is_network = 0; }
/^disks:/ { is_disk = 1; next; }
/^network:/ { is_network = 1; next; }
/load averages:/ {
  if ( $NF > max_load ) { print; }
  next;
}

! is_network && ! is_disk { print; }

is_network {
  name = $1; colls = int($NF); ofail = int($(NF-1)); ifail = int($(NF-3));
  if ( colls > 0 || ofail > 0 || ifail > 0 ) {
    printf("net %s input errs %s output errs %s collisions %s\n", name, ifail, ofail, colls);
  }
}

is_disk {
  name = $NF; total = int($2); used = int($3); ifree = int($7);
  if ( name == "on" ) { next; }
  if ( used / total > max_disk ) {
    printf("disk %s at %.2f used\n", name, used / total);
  }
  if ( ifree < min_ifree ) {
    printf("disk %s ifree at %d\n", name, ifree);
  }
}
```

Unpacking Quoted ASCII

Email subjects will occasionally be rendered in quoted-printable format.[1]

```
#!/usr/bin/awk -f

function unhex(a,b,   i,hex) {
  hex = "0123456789abcdef";
  i = 16 * ( index(hex, tolower(a)) - 1 ) + index(hex, tolower(b)) - 1 ;
  return (i >= 0x20) && (i < 0x7f) ? sprintf("%c", i) : "?";
}

function unqp(str,   chars,nchars,q,i,c,ret) {
  nchars = split(str, chars, "");
  if ( chars[1] chars[2] chars[nchars-1] chars[nchars] != "=??=" ) {
    return str;
  }
  for ( i = 1 ; i < nchars-1 && q <= 2 ; i++ ) {
    if ( chars[i] == "?") { q++; }
  }
  for ( ; i < nchars-1; i++ ) {
    c = chars[i];
    if ( chars[i] == "_" ) { c = " "; }
    if ( chars[i] == "=" ) { c = unhex(chars[i+1], chars[i+2]); i += 2; }
    ret = ret c;
  }
  return ret;
}

!is_body && /^Subject: / {
  line = "";
  for (i = NF ; i >= 1; i-- ) {
    line = unqp($i) ( substr( $i, 1, 2 ) == "=?" ? "" : " " ) line;
  }
  $0 = line;
}

/^$/ { is_body = 1; }

{ print; }
```

This is useful in a ‘procmailrc‘ so as to normalize headers to plain ascii prior to filtering. eg.

```
:0 fw
| decode-subject.awk
```

[1] https://datatracker.ietf.org/doc/html/rfc2047
https://datatracker.ietf.org/doc/html/rfc1341

awk + sed

'sed' pairs well with awk because sed can format text into records that awk can easily handle.

For example, html can be spaghetti-printed across lines, which is not awk-friendly because the line would need to be split into chars and then iterated.

Far better to tidy the html so that (a) each tag starts a new line, (b) the entire <tag attr=... > is on that line, and (c) there is only one pair of angle brackets on a line. ie. filter the html using the following sed script:

```
: start
/<[^>]*$/ { N; s/\n//; b start }
s/</\n</g
s/>/>\n/g
```

As a concrete example, Wordpress installs will publish an RSS feed while also banning hotlinking, so that any images included in the RSS feed fail to render.

Given an HTML RSS feed entry and the source URL, the images can be downloaded and then included directly into the HTML with the following:

```
cat $eml | sed -f ./html-tidy.sed | awk -v url="$url" '
BEGIN {
    cmdfmt = "wget -q --no-check-certificate --referer \"%s\"  -O - \"%s\" | base64 -w 0";
}
function resize(link,   i) {
    i = match(link, "-[0-9]+x[0-9]+\\.jpg");
    return i <= 0 ? link : substr(link, 1, i-1) ".jpg";
}
/^<img/ {
    for ( i = 1; i <= NF ; i++ ) {
        if (substr($i, 1, 4) == "src=" ) {
            lnk = substr($i, 6, length($i)-6);
            img = resize(lnk);
            cmd = sprintf(cmdfmt, url, img);
            cmd | getline b64
            close(cmd);
            $0 = sprintf("<img src=\"data:image/jpeg;base64,%s\">", b64);
        }
    }
}
{ print; }
'
```

Note that Wordpress will also provide an image-set where the 'src' attribute has a low-res version of the image, the resize() function just recovers the original uploaded image.

Pretty-printing

The above ‘sed‘ example of tidying HTML is problematic because it breaks pre-formatted text (eg. code snippets) and anchor tags by introducing whitespace.

A more awk-ish solution would be to reset the Record Separator to < instead of Line Feed.

```
BEGIN { RS="<"; prev = 0;}

{
    if ( prev ) {
        ORS="\n<";
        if ( is_pre ) { ORS="<"; }
        print prev;
    }
    prev = $0;
}

ENDFILE {
    ORS = "\n";
    print prev;
}


/^pre/   { is_pre = 1; }
/^\/pre/ { is_pre = 0; }

/^code/   { is_pre = 1; }
/^\/code/ { is_pre = 0; }

/^a[> \n]/   { is_pre = 1; }
/^\/a[> \n]/ { is_pre = 0; }
```

nb. awk will output a final record separator at the end of the input file. To prevent that extra < the above script prints only the previously seen line, and relies on ENDFILE to print a final Line Feed.

Controlling Input

Multiple passes over the same input are needed sometimes, eg. extracting a certain MIME part from a multi-part email.

ic. get the boundary on the first pass, then ”rewind” the input by adding the current file to the list of files (from the command-line) to be processed, then parse that ”nextfile” with the boundary as the record separator.

```
BEGIN {
    if (! ctype) { ctype = "text/html"; }
    ctype_header = "Content-Type: " ctype;
}

! is_part && /^Content-Type: multipart/ {
    is_part = 1;
    boundary = substr($NF, 11);
    boundary = substr(boundary, 1, length(boundary)-1 );
    RS = "--" boundary;
    ARGC++;
    ARGV[ARGIND+1] = FILENAME
    nextfile
}

is_part && match($0, ctype_header) { print ; }
```

nb. stdin cannot be rewound, only files passed as arguments to awk.