awk, the other useful scripting language

Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan

AT&T Bell Labs (1970s)

Turing complete

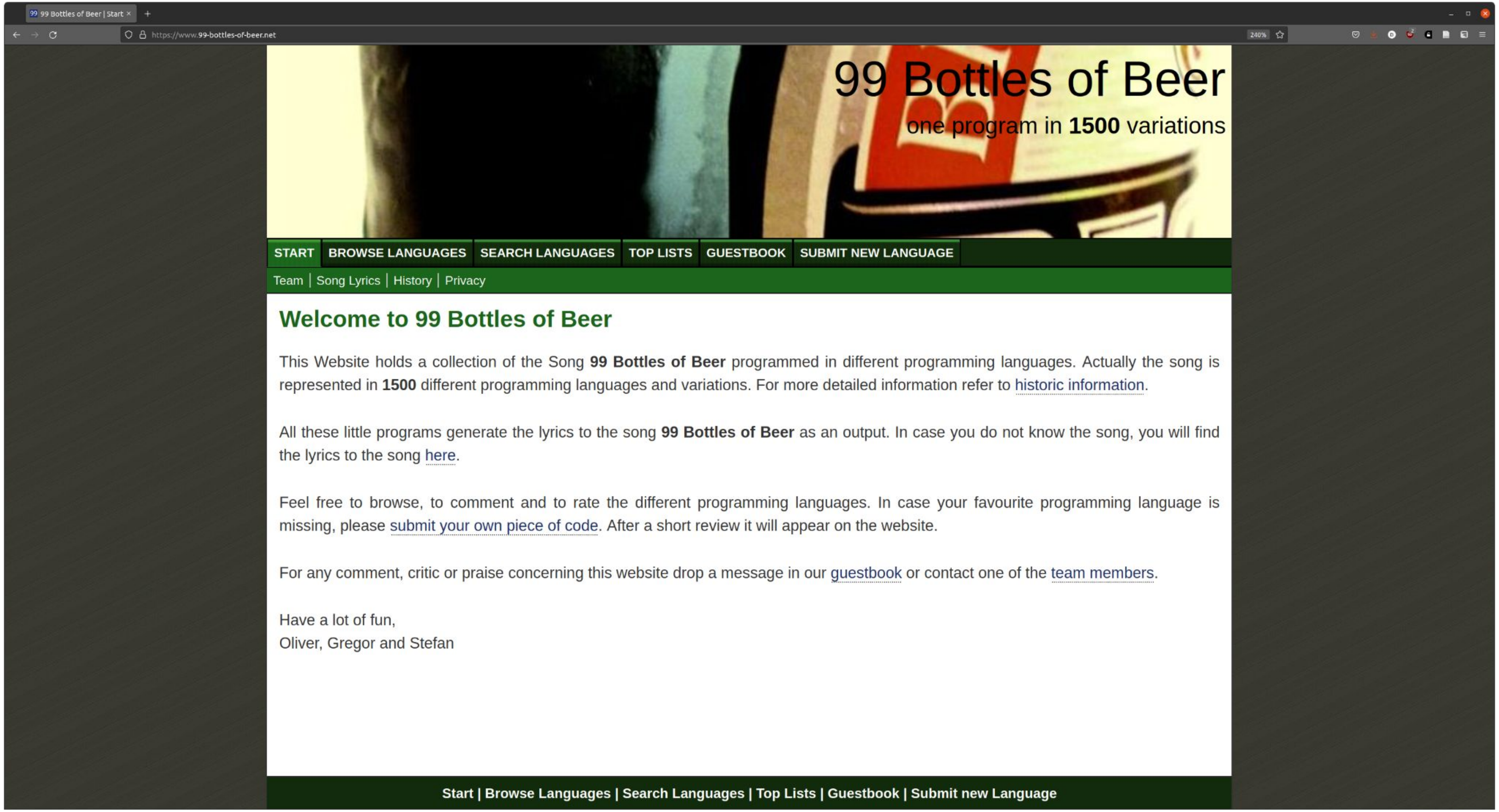
many (very similar) versions (e.g. gawk, mawk, tawk)

scripts can be translated into C programs (e.g. awka)

standard tool in POSIX operating systems

designed to work one (stdin) line at a time

awk limitations inspired Larry Wall to write Perl (or so it has been written)


99 99 Bottles of Beer | Start
https://www.99-bottles-of-beer.net
240%

99 Bottles of Beer

one program in **1500** variations

START | BROWSE LANGUAGES | SEARCH LANGUAGES | TOP LISTS | GUESTBOOK | SUBMIT NEW LANGUAGE

Team | Song Lyrics | History | Privacy

Welcome to 99 Bottles of Beer

This Website holds a collection of the Song **99 Bottles of Beer** programmed in different programming languages. Actually the song is represented in **1500** different programming languages and variations. For more detailed information refer to historic information.

All these little programs generate the lyrics to the song **99 Bottles of Beer** as an output. In case you do not know the song, you will find the lyrics to the song here.

Feel free to browse, to comment and to rate the different programming languages. In case your favourite programming language is missing, please submit your own piece of code. After a short review it will appear on the website.

For any comment, critic or praise concerning this website drop a message in our guestbook or contact one of the the team members.

Have a lot of fun,
Oliver, Gregor and Stefan

Start | Browse Languages | Search Languages | Top Lists | Guestbook | Submit new Language

http://99-bottles-of-beer.net/

99 bottles of beer on the wall, 99 bottles of beer.

Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.

Take one down and pass it around, 97 bottles of beer on the wall.

...

2 bottles of beer on the wall, 2 bottles of beer.

Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.

Take one down and pass it around, no more bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.

Go to the store and buy some more, 99 bottles of beer on the wall.

awk, its a language, really

```awk
#!/usr/bin/awk -f
        BEGIN{
       split( \
       "no mo"\
       "rexxN"\
       "o mor"\
       "exsxx"\
       "Take "\
      "one dow"\
     "n and pas"\
    "s it around"\
   ", xGo to the "\
  "store and buy s"\
  "ome more, x bot"\
  "tlex of beerx o"\
  "n the wall" , s,\
  "x"); for( i=99 ;\
  i>=0; i--){ s[0]=\
  s[2] = i ; print \
  s[2 + !(i) ] s[8]
\ s[4+ !(i-1)] s[9]
\ s[10]", " s[!(i)]\
  s[8] s[4+ !(i-1)]
\ s[9]".";i?s[0]--:\
  s[0] = 99; print \
  s[6+!i]s[!(s[0])]
\ s[8] s[4 +!(i-2)]
\ s[9]s[10] ".\n";}}
```

```awk
#!/usr/bin/awk -f
BEGIN {
   for(i = 99; i >= 0; i--) {
      print ubottle(i), "on the wall,", lbottle(i) "."
      print action(i), lbottle(inext(i)), "on the wall."
      print
   }
}
function ubottle(n) {
   return sprintf("%s bottle%s of beer", n ? n : "No more", n - 1 ? "s" : "")
}
function lbottle(n) {
   return sprintf("%s bottle%s of beer", n ? n : "no more", n - 1 ? "s" : "")
}
function action(n) {
   return sprintf("%s", n ? "Take one down and pass it around," : \
                            "Go to the store and buy some more,")
}
function inext(n) {
   return n ? n - 1 : 99
}
```

awk basics: arguments and variables

-F to set field separator (FS)

-v to set your own variables

e.g. -v x=5

BEGIN{}{}END{}

built–in variables:

NF = number of fields

NR = number of records

FS = field separator (space is default)

OFS = output field separator (space is default)

$1, $2, … = field values

awk basics: data types

numbers are bare

strings are "quoted"

scalars store a single value

```
e.g. x=42; x="text"
```

arrays store multiple key/value pairs

```
e.g. x[0]=42; x[5]="text"; x["y"]="xyz"
```

arrays can have multiple dimensions

```
e.g. x[0]["y"]=42; x[5]["z"]="text"; x["y"]["z"]="xyz"
```

awk basics: math

x = y	set
x + y	add
x++	add one
x - y	subtract
x--	subtract one
x * y	multiply
x / y	divide
x % y	modulus
x [+-*/%]= y	add/subtract/multiply/divide/modulus variable
x**y	to the power of

awk basics: if/else

condition:

```
    x==y; x!=y; x>=y; x<=y; x>y; x<y; x in y; x~/y/; x!~/y/
    (); &&; ||
if(condition){action}
if(condition){action}else{action}
if(condition){action}else if(condition){action}
if(condition){action}else if(condition){action}else{action}
```

awk basics: loops

```
while(condition){action}
for(initialization; condition; increment){action}
    e.g. for(x=5; x>0; x--){action}
```

loop keywords:

break	end the loop immediately
continue	skip to the next loop iteration
next	skip to the next input line

awk basics: useful keywords…

asort	sort an array
delete	delete an element from an array
gensub	RE search and replace
in	test if index exists in array
length	length of a string
match	match a string with a RE
patsplit	split a string into an array using a RE

awk basics: …useful keywords

print	print to stdout
printf	print to stdout with formatting
rand	return a random number (set seed with srand)
split	split a string into an array using a separator
substr	extract a string portion
tolower	convert to lowercase
toupper	convert to uppercase

awk examples

```
awk -F'\t' '{print $3}'
awk -F'\t' '{print $3,$5}'
awk -F'\t' 'BEGIN{OFS="\t"}{print $3,$5}'
awk -F'\t' 'BEGIN{x=0}{x+=$3}END{print x}'
awk -F'\t' '{if($3>9){print $0}}'
awk -F'\t' '{if($3>9&&$5=="x"){print $0}}'
```

perl

‘Swiss-Army chainsaw’ (Henry Spencer)

- i.e. powerful but inelegant
- reclaimed as a complement by Perl enthusiasts

1987: Perl 1.0 released by Larry Wall

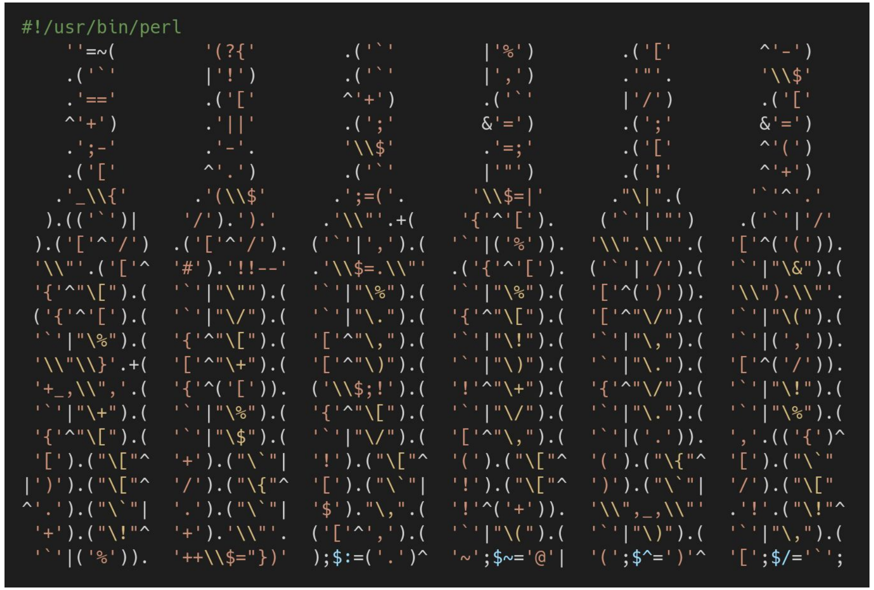
- combines the ‘best’ of sed, awk, C, and sh
- multiple ways to do (al)most everything

1989: Perl 3.0 released (GPL)

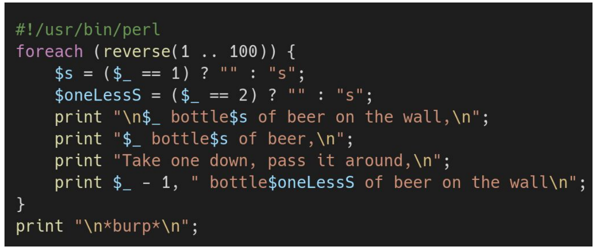
1994: Perl 5.0 released

1995: CPAN founded

perl (and 99 bottles of beer)



```perl
#!/usr/bin/perl
    ''=~(         '(?{'        .('`'        |'%')        .('['         ^'-')
    .('`'         |'!')        .('`'        |',')        .'"'.         '\\$'
    .'=='         .('['        ^'+')        .('`'        |'/')         .('['
    ^'+')         .'||'        .(';'        &'=')        .(';'         &'=')
    .';-'         .'-'.        '\\$'        .'=;'        .('['         ^'(')
    .('['         ^'.')        .('`'        |'"')        .('!'         ^'+')
   .'_\\{'       .'(\\$'      .';=('.      '\\$=|'      ."\|".(       '`'^'.'
  ).(('`')|     '/').').'    .'\\"'.+(    '{'^'[').    ('`'|'"')     .('`'|'/'
 ).('['^'/')  .('['^'/').  ('`'|',').(  '`'|('%')).  '\\".\\"'.(  '['^('(')).
 '\\"'.('['^  '#').'!!--'  .'\\$=.\\"'  .('{'^'[').  ('`'|'/').(  '`'|"\&").(
 '{'^"\[").(  '`'|"\"").(  '`'|"\%").(  '`'|"\%").(  '['^(')')).  '\\").\\"'.
 ('{'^'[').(  '`'|"\/").(  '`'|"\.").(  '{'^"\[").(  '['^"\/").(  '`'|"\(").(
 '`'|"\%").(  '{'^"\[").(  '['^"\,").(  '`'|"\!").(  '`'|"\,").(  '`'|(',')).
 '\\"\\}'.+(  '['^"\+").(  '['^"\)").(  '`'|"\)").(  '`'|"\.").(  '['^('/')).
 '+_,\\",'.(  '{'^('[')).  ('\\$;!').(  '!'^"\+").(  '{'^"\/").(  '`'|"\!").(
 '`'|"\+").(  '`'|"\%").(  '{'^"\[").(  '`'|"\/").(  '`'|"\.").(  '`'|"\%").(
 '{'^"\[").(  '`'|"\$").(  '`'|"\/").(  '['^"\,").(  '`'|('.')).  ','.(('{')^
 '[').("\["^  '+').("\`"|  '!').("\["^  '(').("\["^  '(').("\{"^  '[').("\`"
|')').("\["^  '/').("\{"^  '[').("\`"|  '!').("\["^  ')').("\`"|  '/').("\["
^'.').("\`"|  '.').("\`"|  '$')."\,".(  '!'^('+')).  '\\",_,\\"'  .'!'.("\!"^
 '+').("\!"^  '+').'\\"'.  ('['^',').(  '`'|"\(").(  '`'|"\)").(  '`'|"\,").(
 '`'|('%')).  '++\\$="})'  );$:=('.')^  '~';$~='@'|  '(';$^=')'^  '[';$/='`';
```



```perl
#!/usr/bin/perl
foreach (reverse(1 .. 100)) {
    $s = ($_ == 1) ? "" : "s";
    $oneLessS = ($_ == 2) ? "" : "s";
    print "\n$_ bottle$s of beer on the wall,\n";
    print "$_ bottle$s of beer,\n";
    print "Take one down, pass it around,\n";
    print $_ - 1, " bottle$oneLessS of beer on the wall\n";
}
print "\n*burp*\n";
```

perl basics: one–liners…

-e '...' == execute the code within quotes

-p == print after processing each input chunk

-n == do not print after processing each input chunk

-0777 == read the input all at once

-i.old == edit file in–place (makes a copy file.old)

-l == chomp() each chunk [remove \n | \r | \r\n]

-a == split(/ /, $_) each line into @F

-Fx to use x instead of <space>

usually: perl -pe || perl -ne || perl -lane || perl -077 -le

perl basics: …one–liners

@ARGV == arguments used to start Perl

@F == input chunk split by the splitting scalar

    default = <space> (-F flag)

$_ == input chunk

$a, $b == used in sort()

$1, $2, … == used in regular expressions

perl basics: syntax

lines end with semicolons;

comments start with an octothorpe #

double quotes are processed before the next action

single quotes are literal (no processing)

slashes escape special characters \

any pair of characters can be used for quotes

‘my’ provides variable scoping

indices start from zero

perl basics: variables

$scalar: a single number, string, or reference (pointers)

    size given by length($scalar)

@array: lists of numbers, letters, strings, or references

    accessed by index position e.g. $array[0]

    size given by $#array

%hash: lists of numbers, letters, strings, references

    accessed by a key (a unique value) e.g. $hash{'key'}

    size given by scalar(keys(%hash))

perl basics: array tricks…

create an array explicitly using ()

e.g. @array = (0, 3, 5)

to convert a string into an array, use split()

e.g. split(/ /, 'this is a string'); ### [this] [is] [a] [string]

e.g. split(/i|s/, 'this is a string'); ### [th] [ ] [a] [tr] [ng]

to convert an array into a string, use join()

e.g. join(' ', ('x', 'y', 'z')); ### x y z

e.g. join(' | ', ('x', 'y', 'z')); ### x | y | z

perl basics: …array tricks

to access a subset (slice) of the array, use index numbers

```
e.g. @x = @y[1..2]; @x = @y[3, 2, 0];
```

to add to an array, use push()

```
e.g. push(@x, $y);
```

to sort an array, use sort()

```
e.g. alphabetic: @x = sort({$a cmp $b} @y);
e.g. numeric: @x = sort({$a <=> $b} @y);
```

perl basics: hash tricks…

create a hash explicitly using (=>)

e.g. %hash = ('key0'=>'value0', 'key1'=>'value1', 'key2'=>'value2');

to delete an element from a hash, use delete()

e.g. delete($hash{'key'});

to test if an element is present in a hash, use exists()

e.g. exists($hash{'key'})

to extract the keys from a hash, use keys()

e.g. @array = keys(%hash);

e.g. $number = keys(%hash);

perl basics: …hash tricks

to extract the values from a hash, use values()

```
e.g. @array = values(%hash);
```

to sort hash keys by their corresponding values, use sort()

```
e.g. alphabetically: sort({$hash{$a} cmp $hash{$b}} keys(%hash));
e.g. numerically: sort({$hash{$a} <=> $hash{$b}} keys(%hash));
```

to make arrays with unique values, use a hash

```
e.g. @unique = keys %{{ map {$_ => 1} @array}};
```

to count the number of occurrences, use a hash

```
e.g. $hash{$_}++ for @array;
```

perl basics: math

$x = $y	set
$x + $y	add
$x++	add one
$x - $y	subtract
$x--	subtract one
$x * $y	multiply
$x / $y	divide
$x % $y	modulus
$x [+-*/%]= $y	add/subtract/multiply/divide/modulus variable
$x**$y	to the power of

perl basics: text

`$x = $y`	set
`$x .= $y`	append
`$x . $y`	concatenate
`$x x $y`	repetition
`$x =~ s/y/z/`	replace
`$x =~ tr/0/1/`	replace

perl basics: if/else

condition:

```
        $x==$y; $x!=$y; $x>=$y; $x<=$y; $x>$y; $x<$y
        $x=~m/y/; $x!~m/y/; $x eq $y; $x ne $y
        (); &&; ||
if(condition){action}
if(condition){action}else{action}
if(condition){action}elseif(condition){action}
if(condition){action}elseif(condition){action}else{action}
```

perl basics: loops

while(condition){action}

for(initialization; condition; increment){action}

    e.g. for($x=5; $x>0; $x--){action}

loop keywords:

last	end the loop immediately
next	skip to the next loop iteration

perl basics: useful keywords

length	length of a string
lc	converts strings (or character) to lowercase
print	prints strings (default to STDOUT)
rand	return a random number (set seed with srand)
reverse	reverses the order of elements strings and arrays
sprintf	formats data (numbers)
substr	extracts a string from another string
uc	converts strings to UPPERCASE

awk => perl examples

```
awk -F'\t' '{print $3}'
perl -F'\t' -lane '{print $F[2]}'
awk -F'\t' 'BEGIN{OFS="\t"}{print $3,$5}'
perl -F'\t' -lane '{print $F[2]."\t".$F[4]}'
awk -F'\t' 'BEGIN{x=0}{x+=$3}END{print x}'
perl -F'\t' -lane 'BEGIN{$x=0}{$x+=$F[2]}END{print $x}'
awk -F'\t' '{if($3>9){print $0}}'
perl -F'\t' -lane '{if($F[2]>9){print $_}}'
awk -F'\t' '{if($3>9&&$5=="x"){print $0}}'
perl -F'\t' -lane '{if($F[2]>9&&$F[4]eq"x"){print join("\t",@F)}}'
```