awk

Matt Might

University of Utah

matt.might.net

No class on Tuesday!

Homework

- Released next Tues
- Check out from git
- Turn in: private repo

`winterfell.ucombinator.org`

Today

- A guided tour of awk
- Relational scripting
- Pinch of emacs, vim

Next Thursday

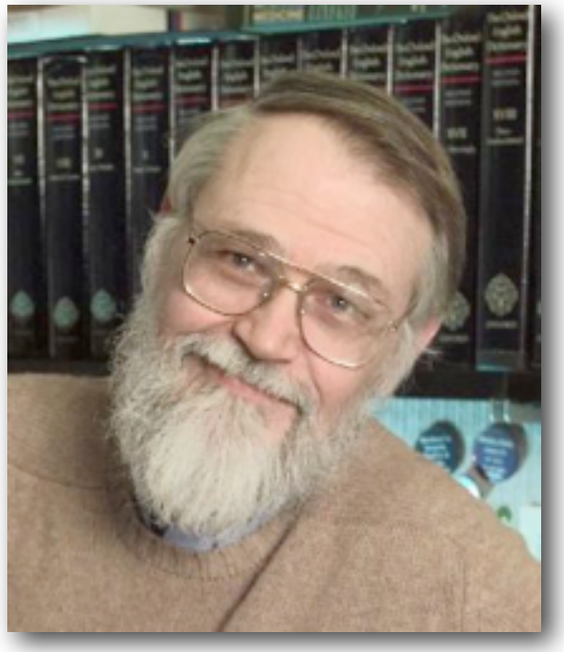
- Lexical analysis
- New tool: `lex`

History

- `sh` - 1971
- `sed` - 1973
- AWK - 1977

Aho

Weinberger



Kernighan

How C sees files

$$byte_1 byte_2 byte_3 byte_4 ...$$

How sed sees files

$line_1$

$line_2$

$line_3$

$line_4$

...

How awk sees files

$record_1$

$record_2$

$record_3$

$record_4$

...

$field_1$ $field_2$ $field_3$ ...

```
awk -F separator
```

```
awk -F :
```

$$field_1:field_2:field_3:...$$

```
awk -F ,
```

$$field_1, field_2, field_3, ...$$

The Unix Way

- Many *ad hoc* databases
- One line = one record
- Delimiters: `,` `:` `\t` `=`

Some one liners

See: http://www.pement.org/awk/awk1line.txt

```
awk '1'
```

```
awk '1;1'
```

```
awk 'NR % 2'
```

```
awk '1;{print ""}'
```

```
awk '{print NR "\t" $0}'
```

```
awk 'END{print NR}'
```

```
awk 'NF > 4'
```

awk programs

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

awk rules

```
pattern
```

```
pattern { statements }
```

awk patterns

/regex/

expression

pat_1, pat_2

BEGIN

END

awk statements

```
if (expression) statement [ else statement ]
while (expression) statement
for (expression; expression; expression) statement
for (var in array) statement
do statement while (expression)
break
continue
{ [ statement ... ] }
expression
print [ expression-list ] [ > expression ]
printf format [ , expression-list ] [ > expression ]
return [ expression ]
next
nextfile
delete array[expression]
delete array
exit [ expression ]
```

for (*var* **in** *array*) *statement*

print [*expression*-list] [> *expression*]

next
nextfile
delete array[*expression*]
delete array

for (*var* **in** *array*) *statement*

print [*expression*-list] [> *expression*]

next

nextfile

delete *array*[*expression*]

delete *array*

awk expressions

```
lit
var
var[expr]
expr binop expr
preop expr
expr postop
expr expr                    # oops!
var(exprlist)
(expr)
```

Special variables

Name	Meaning
`$0`	current line
`$`n	field n
`FILENAME`	current file
`NR`	# of records thus far
`NF`	# of fields this line
`RS`	record separator
`OFS`	output field separator
`ORS`	output record separator

$\$n$ is not a variable!

```
$(10) $i $(a[i])
```

awk arrays

Associative

Not fully first class!

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

BEGIN {
  arr[0] = 1 ;
  print 0, arr[0] ;    # prints 0 1
  modify_array(arr) ; # ok
  print 0, arr[0] ;    # prints 0 2
  brr = arr ;          # error
  exit ;
}

function modify_array(array) {
  for (k in array) {
    array[k]++ ;
  }
}
```

awk functions

```
function name(arg1,...,argN) { statements }
```

Oops! No local variables!

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

{ print f($0); }

function f(n,    i) {
  i = 3 ;
  return $i ;
}
```

awk examples

```
awk '1'
```

```
awk '{print $2, $1}'
```

```
awk 'END{print}'
```

```
awk 'NR % 2'
```

```
awk '/foo/ && /bar/ { print }'
```

```
awk -F : '$3 == 500 { print $1 }' /etc/passwd
```

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

/^#/ { next ; }

{ users[$3] = $1 ; }

END {
 max = 0 ;
 for (i in users) {
  if ((i+0) > (max+0))
   max = i ;
 }
 print users[max];
}
```

```
awk '!a[$0]++'
```

```
$ awk '{ print gensub(/\/\/[ ]?(.*)/, "/* \\1 */", "g" ) }'
```

Design principles

Target a niche.

Anti-patterns

Mix variables and arrays.

Class warfare with arrays.

Juxtapose to concatenate.

Ignore local variables.

emacs and vim

vim	emacs
/*pattern*	C-M-s *pattern* ret
:s/*pattern*/*replace*/*args*	M-x replace-regexp RET *pattern* RET *replace* RET
:r!*command*	M-1 M-! *command*
:'<'>!*command*	M-1 M-| *command* RET

Just for fun:

Relational shell scripting

Can shell scripts have principles?

Yes! Relations!

A relation is set of tuples.

A record is a tuple.

A file is a set of tuples!

{(Bob,32), (Judy, 31)}

Bob	32
Judy	31

Name	Age
Bob	32
Judy	31

Bob,31
Judy,32

Relational algebra

- union :: `cat`
- selection :: `sed, grep, awk`
- projection :: `cut, awk`
- renaming :: `awk`
- difference :: `diff, comm`
- Cartesian product :: ???

Relational algebra == SQL!

Union

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

cat *A* *B* | sort | uniq

Selection

$$\sigma_\phi(A) = \{x \mid x \in A \text{ and } \phi(x) \text{ is true } \}$$

$$\texttt{awk '}\varphi\texttt{'}\ \ A$$

Projection

$$\pi_{i_1,\ldots,i_n}(R) = \{(x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_n) \in R\}$$

$$\texttt{cut -f } i_1, \ldots, i_n \; A$$

Rename?

```
awk '{ print $2, $1}' A
```

Difference

$$A - B = \{x \mid x \in A \text{ but } x \notin B\}$$

comm -23 <(sort *A*) <(sort *B*)

Cartesian product

$$A \times B = \{(a_1, \ldots, a_n, b_1, \ldots, b_m) \mid (a_1, \ldots, a_n) \in A \text{ and } (b_1, \ldots, b_m) \in B\}$$

```
BEGIN { first=1 ; i = 0 ; j = 0 ; }

{ if (first)  a[i++] = $0 ;
  else        b[j++] = $0 ; }

ENDFILE {
  if (first) first = 0;
}

END {
  for (n = 0; n < i; n++)
   for (m = 0; m < j; m++)
     print a[n], b[m] ;
}
```

Relationally complete!

Challenge

- Read blog post.
- Try out example.

Reminder

- Homework next Tues
  - Check out from `git`
  - Turn in: private repo

`winterfell.ucombinator.org`

Questions?