Rocky Enterprise Linux 9.2 Manual Pages on command 'states.1'

$ man states.1

STATES(1) STATES STATES(1)

NAME

states - awk alike text processing tool

SYNOPSIS

states [-hvV] [-D var=val] [-f file] [-o outputfile] [-p path] [-s startstate] [-W level] [filename ...]

DESCRIPTION

States is an awk-alike text processing tool with some state machine ex? tensions. It is designed for program source code highlighting and to similar tasks where state information helps input processing.

At a single point of time, States is in one state, each quite similar to awk's work environment, they have regular expressions which are matched from the input and actions which are executed when a match is found. From the action blocks, states can perform state transitions; it can move to another state from which the processing is continued. State transitions are recorded so states can return to the calling state once the current state has finished.

The biggest difference between states and awk, besides state machine

extensions, is that states is not line-oriented. It matches regular expression tokens from the input and once a match is processed, it con? tinues processing from the current position, not from the beginning of the next input line.

OPTIONS

-D var=val, --define=var=val

Define variable var to have string value val. Command line definitions overwrite variable definitions found from the con? fig file.

-f file, --file=file

Read state definitions from file file. As a default, states tries to read state definitions from file states.st in the cur? rent working directory.

-h, --help

Print short help message and exit.

-o file, --output=file

Save output to file file instead of printing it to stdout.

-p path, --path=path

Set the load path to path. The load path defaults to the di? rectory, from which the state definitions file is loaded.

-s state, --state=state

Start execution from state state. This definition overwrites start state resolved from the start block.

-v, --verbose

Increase the program verbosity.

-V, --version

Print states version and exit.

-W level, --warning=level

Set the warning level to level. Possible values for level are:

light light warnings (default)

all all warnings

STATES PROGRAM FILES

States program files can contain on start block, startrules and

namerules blocks to specify the initial state, state definitions and expressions.

The start block is the main() of the states program, it is executed on script startup for each input file and it can perform any initializa? tion the script needs. It normally also calls the check_startrules() and check_namerules() primitives which resolve the initial state from the input file name or the data found from the beginning of the input file. Here is a sample start block which initializes two variables and does the standard start state resolving:

```
start
{
  a = 1;
  msg = "Hello, world!";
  check_startrules ();
  check_namerules ();
}
```

Once the start block is processed, the input processing is continued from the initial state.

The initial state is resolved by the information found from the startrules and namerules blocks. Both blocks contain regular expres? sion - symbol pairs, when the regular expression is matched from the name of from the beginning of the input file, the initial state is named by the corresponding symbol. For example, the following start and name rules can distinguish C and Fortran files:

```
namerules
{
  /\.(c|h)$/    c;
  /\.[fF]$/     fortran;
}
startrules
{
  /-\*- [cC] -\*-/      c;
  /-\*- fortran -\*-/   fortran;
```

}

If these rules are used with the previously shown start block, states first check the beginning of input file. If it has string -*- c -*-, the file is assumed to contain C code and the processing is started from state called c. If the beginning of the input file has string -*- fortran -*-, the initial state is fortran. If none of the start rules matched, the name of the input file is matched with the namerules. If the name ends to suffix c or C, we go to state c. If the suffix is f or F, the initial state is fortran.

If both start and name rules failed to resolve the start state, states just copies its input to output unmodified.

The start state can also be specified from the command line with option -s, --state.

State definitions have the following syntax:

state { expr {statements} ... }

where expr is: a regular expression, special expression or symbol and statements is a list of statements. When the expression expr is matched from the input, the statement block is executed. The statement block can call states' primitives, user-defined subroutines, call other states, etc. Once the block is executed, the input processing is con? tinued from the current intput position (which might have been changed if the statement block called other states).

Special expressions BEGIN and END can be used in the place of expr. Expression BEGIN matches the beginning of the state, its block is called when the state is entered. Expression END matches the end of the state, its block is executed when states leaves the state.

If expr is a symbol, its value is looked up from the global environment and if it is a regular expression, it is matched to the input, other? wise that rule is ignored.

The states program file can also have top-level expressions, they are evaluated after the program file is parsed but before any input files are processed or the start block is evaluated.

call (symbol)

Move to state symbol and continue input file processing from that state. Function returns whatever the symbol state's ter? minating return statement returned.

calln (name)

Like call but the argument name is evaluated and its value must be string. For example, this function can be used to call a state which name is stored to a variable.

check_namerules ()

Try to resolve start state from namerules rules. Function re? turns 1 if start state was resolved or 0 otherwise.

check_startrules ()

Try to resolve start state from startrules rules. Function re? turns 1 if start state was resolved or 0 otherwise.

concat (str, ...)

Concanate argument strings and return result as a new string.

float (any)

Convert argument to a floating point number.

getenv (str)

Get value of environment variable str. Returns an empty string if variable var is undefined.

int (any)

Convert argument to an integer number.

length (item, ...)

Count the length of argument strings or lists.

list (any, ...)

Create a new list which contains items any, ...

panic (any, ...)

Report a non-recoverable error and exit with status 1. Func? tion never returns.

print (any, ...)

Convert arguments to strings and print them to the output.

range (source, start, end)

Return a sub-range of source starting from position start (in?
clusively) to end (exclusively). Argument source can be string
or list.

regexp (string)

Convert string string to a new regular expression.

regexp_syntax (char, syntax)

Modify regular expression character syntaxes by assigning new
syntax syntax for character char. Possible values for syntax
are:

'w' character is a word constituent

' ' character isn't a word constituent

regmatch (string, regexp)

Check if string string matches regular expression regexp.
Functions returns a boolean success status and sets sub-expres?
sion registers $n.

regsub (string, regexp, subst)

Search regular expression regexp from string string and replace
the matching substring with string subst. Returns the result?
ing string. The substitution string subst can contain $n ref?
erences to the n:th parenthesized sup-expression.

regsuball (string, regexp, subst)

Like regsub but replace all matches of regular expression reg?
exp from string string with string subst.

require_state (symbol)

Check that the state symbol is defined. If the required state
is undefined, the function tries to autoload it. If the load?
ing fails, the program will terminate with an error message.

split (regexp, string)

Split string string to list considering matches of regular rex?
pression regexp as item separator.

sprintf (fmt, ...)

Format arguments according to fmt and return result as a
string.

strcmp (str1, str2)

Perform a case-sensitive comparision for strings str1 and str2.

Function returns a value that is:

-1 string str1 is less than str2

0 strings are equal

1 string str1 is greater than str2

string (any)

Convert argument to string.

strncmp (str1, str2, num)

Perform a case-sensitive comparision for strings str1 and str2

comparing at maximum num characters.

substring (str, start, end)

Return a substring of string str starting from position start

(inclusively) to end (exclusively).

BUILTIN VARIABLES

$. current input line number

$n the n:th parenthesized regular expression sub-expression from

the latest state regular expression or from the regmatch primi?

tive

$` everything before the matched regular rexpression. This is us?

able when used with the regmatch primitive; the contents of

this variable is undefined when used in action blocks to refer

the data before the block's regular expression.

$B an alias for $`

argv list of input file names

filename

name of the current input file

program name of the program (usually states)

version program version string

FILES

/usr/share/enscript/hl/*.st enscript's states definitions

SEE ALSO

awk(1), enscript(1)

AUTHOR

Markku Rossi <mtr@iki.fi> <http://www.iki.fi/~mtr/>

GNU Enscript WWW home page: <http://www.iki.fi/~mtr/genscript/>

STATES Oct 23, 1998 STATES(1)