Perl shortcuts Cheat Sheet

by SathyaNarayanan (Sathyanarayanan) via cheatography.com/136085/cs/28347/

Documentation

=pod
my $dog = ”hatchi”;
my $cat = “meow”;
=cut

Logical Operators

Logical	or, and, xor, not, ||, &&, !
Strings	eq, gt, lt, le, ge, cmp
Numbers	==, <,>,<=,>=, <=>

File Test Operators

-e exists
-r readable
-w writable
-d directory
-f file
-T Text file

Conditions

Unless
if { }
elsf{}
else{}
given() { when { } }

Regular expression

Whitespace	\s
word	\w
Digit	\d
Non digit	\D
Beginning	^

Regular expression (cont)

End	$

/\bFred\b/ matches whole word
Next LINE if $line=/^#/;

Quantifiers

min, max	\d{min, max}
7 to 11 digit	\d{7,11}
Only matches 7 digit	\d{7}

+, *, ? .
{} quantifiers applies to single character
() store in memory
s/(S+)s+(S+)/$2 $1/

LIST processing

print reverse sort map{lc} keys %hash
To parse a string 12:59:59 am
-my ($hour, $min, $sec, $ampm)=/(\d+): (\d+) : (\d+) * (\w+)/

Terms and List Operator

|| has lower precedence than chdir

chdir $foo || die #(chdir $foo) || die
chdir($foo) || die #(chdir $foo) || die
chdir +($foo) || die #(chdir $foo) || die

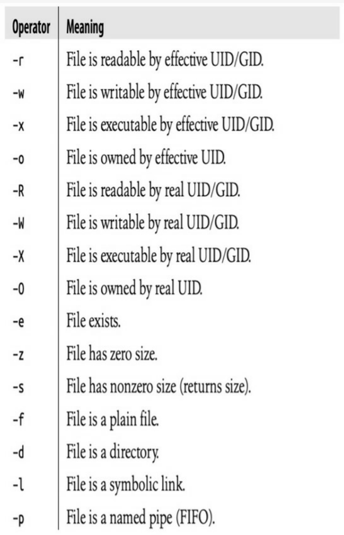
Evalvuation

print($foo, exit) # Obviously not what you want
print $foo, exit #nor this

These do the print before evaluating exit.

(print $foo), exit
or
print ($foo), exit

File test operator



Operator	Meaning
-r	File is readable by effective UID/GID.
-w	File is writable by effective UID/GID.
-x	File is executable by effective UID/GID.
-o	File is owned by effective UID.
-R	File is readable by real UID/GID.
-W	File is writable by real UID/GID.
-X	File is executable by real UID/GID.
-O	File is owned by real UID.
-e	File exists.
-z	File has zero size.
-s	File has nonzero size (returns size).
-f	File is a plain file.
-d	File is a directory.
-l	File is a symbolic link.
-p	File is a named pipe (FIFO).

Total seconds /day = 86400
5 Minutes = 300
12 hour = 300 *12* 12 = 43200
Total hour= 12 hr/86400 = 0.5
-M $finename > 0.5 #greated than 12 hours
-M $finename > 0.5 #recently modified file
&callfunction If int(-A $filename) == 90

Tricks

```
Sub setenv{
my ($key, $value)=@_;
$ENV{$key)=$value
unless $ENV{$key}
}
```

Initialization

```
BEGIN {
my @scale = ("A" .. "-
G");
my $note = -1;
sub next_pitch {
return $scale[ ($note
+= 1) %= @scale ] }; }
```

It will execute before your program starts

Handle reference

```
for $file(@names){
my $fh;
open($fh, $file) ||
next;
$filehandle{$file}-
=$fh;
}
```

Symbol table reference

```
$scalarref=
*foo{SCALAR} # \$foo
$arrayref=*ARGV{ARRAY}
# \@ARGV
$hashref = *ENV{HASH};
# \%ENV
$coderef=*handler{-
CODE} # \&handler
```

Symbol table reference (cont)

```
$gobref=*foo{GLOB}
$ioref=*STDIN{IO};
$format=*foo{FORMAT};
```

protection

```
use strict “refs”
```

block counterman the decree with
no strict “refs”

Symbolic reference example:

```
our $value = "global";
{
my $value = "private";
print "Inside, mine is
${value}, "; say "but
ours is ${"value"}.";
}
say "Outside, ${value}
is again ${"valu-
e"}.";
```

which prints:
Inside, mine is private, but ours is global. Outside, global is again global.

Package variables are global variable

Variables

$days : simple scalar value
$days[28] : 29 elements
$days{“feb”} value from hash days
${day}. Equvalent to $day
$Dog;:day; from Dog package

Variables (cont)

$#days last index of array #days
$days->[28] 29 the element pointed to by reference
$days[0][2] multi dim array
$days{2000}{“feb”} mult dim hash
$days{200,”feb”} multi dimensional hash emulation
@days : array containing ($day[0], $day[1]….$day[N])
@days[3,4,5] :array containing ($day[3], $day[4], 4day[5])
@days[3..5]: array containing ($day[3], $day[4],$day[5])
@days{“Jan”, “Feb”}: hash containing ($days{“Jan”}, $days{“feb”})
$days{“feb”} equvalent to $days{feb}

Name space

Lexical scope: block
Symbol table : global
Tips:
$foo, @foo can be declared
In same block,
$foo[0] is different from $foo

CommandLine

Perl -E ‘say 10/3’;
Perl-Mbigrat. -E ‘sat 4/3. * 5/12’

Truth

String True except for “ “ and “0”
Number true except for 0
Any reference is true
Any undefined value is false.

Tips

$” is equal to space “ “
$temp=join($”, @array);
Or
$temp=join(“ “, @array);

Quote Construct

q//	' '
qq//	" "
qx/ /	Command exec
qw//	WordList
m//	Pattern match
s//	Substitute
y///	Translation
Qr//	Regular Expression

Auto Increment and Decrement

++$b{/(\w+)/}[0]}
# increment the element hash indexed by word in the default
search variable ($_)and return after increment.

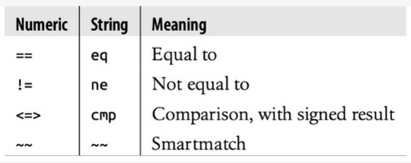
my $foo;
$foo= “99”; print ++ $foo ; #prints 100
$foo=”Az”; print ++ $foo ; #prints Ba

Binding operator

$string ! ~ /$pattern/
! ($string =~ /$pattern/)
not ($string =~ /$pattern/)

~ is after =

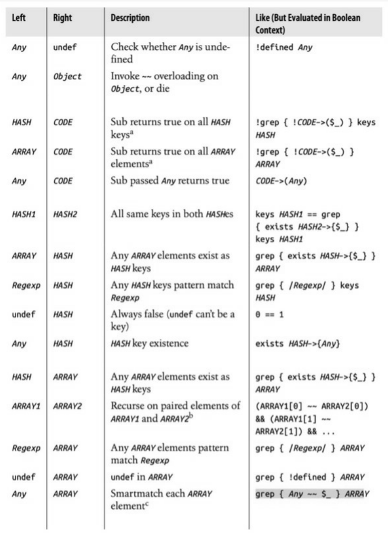
Equality Operator



Numeric	String	Meaning
==	eq	Equal to
!=	ne	Not equal to
<=>	cmp	Comparison, with signed result
~~	~~	Smartmatch

cmp : return -1 if left operand is less than right, 0 If it is equal, 1 if left operand is greater than right.

smartwatch operator



Left	Right	Description	Like (But Evaluated in Boolean Context)
Any	undef	Check whether *Any* is undefined	!defined *Any*
Any	*Object*	Invoke ~~ overloading on *Object*, or die	
HASH	*CODE*	Sub returns true on all *HASH* keys[a]	!grep { !*CODE*->($_) } keys *HASH*
ARRAY	*CODE*	Sub returns true on all *ARRAY* elements[a]	!grep { !*CODE*->($_) } *ARRAY*
Any	*CODE*	Sub passed *Any* returns true	*CODE*->(*Any*)
HASH1	*HASH2*	All same keys in both *HASH*es	keys *HASH1* == grep { exists *HASH2*->{$_} } keys *HASH1*
ARRAY	*HASH*	Any *ARRAY* elements exist as *HASH* keys	grep { exists *HASH*->{$_} } *ARRAY*
Regexp	*HASH*	Any *HASH* keys pattern match *Regexp*	grep { /*Regexp*/ } keys *HASH*
undef	*HASH*	Always false (undef can't be a key)	0 == 1
Any	*HASH*	*HASH* key existence	exists *HASH*->{*Any*}
HASH	*ARRAY*	Any *ARRAY* elements exist as *HASH* keys	grep { exists *HASH*->{$_} } *ARRAY*
ARRAY1	*ARRAY2*	Recurse on paired elements of *ARRAY1* and *ARRAY2*[b]	(ARRAY1[0] ~~ ARRAY2[0]) && (ARRAY1[1] ~~ ARRAY2[1]) && ...
Regexp	*ARRAY*	Any *ARRAY* elements pattern match *Regexp*	grep { /*Regexp*/ } *ARRAY*
undef	*ARRAY*	undef in *ARRAY*	grep { !defined } *ARRAY*
Any	*ARRAY*	Smartmatch each *ARRAY* element[c]	grep { *Any* ~~ $_ } *ARRAY*

my %hash = (red => 1, blue => 2, green => 3, orange => 4, yellow => 5, purple => 6,
black => 7, grey => 8, white => 9);
my @array = qw(red blue green);
print "some element in hash" if @array ~~ %hash;

Reference

```
@tailings=popmany(\@a, \@a2, \@a3);
Sub popmany{
my @retlist=()
for my $ref (@_) {
push @retlist @$ref;
}
return @retlist;
}
```

If you want to pass more than one array and hashes to the function you might have to use pass by reference, it saves the time and space

Hash of Hash

```
%table= (
         'john' =>
{age =>47, eyes=>'brown', weight=>186},
         'Mary'=>{-
age=>23, eyes=>'hazel', weight=>128}
        );
print $table{john}-{age}
```

Prints 47

Operator: ref

Ref operator to determine what a reference is pointing. SCALAR, ARRAY, HASH, CODE, GLOB, REF, VSTRING, IO, LVALUE, FORMAT, REGEXP

Symbolic reference

```
$name="bam";
$$name=1; #sets $bam
$name->[0]=4;
 #set the first element of @bam
$name->{X}="Y" #sets the X element of Y
@$name=() #resets the @bam
&$name #calls the function bam
```

Status

$$ current process id
$? Exits status

Context

Scalar:
$x = funkshun()
$x[1] = funkshun()
$x{"ray"}= funkshun()
List:
@x= funkshun();
@x[1]= funkshun();
@ax{"ray"}= funkshun()
%x= funkshun()
($x, $y, $z)= funkshun()
($x)= funkshun()
my $x= funkshun() #scalar context
my ($x)= funkshun(). # list context
Boolean:
@x; gives number of elements
While(@x){
$file=Shift(@x);
unlink($file) || warn. "couldn't delete file";
}

List values and arrays

@stuff=("-one", "two", "three")	Assign entire list to array
$stuff=("one", "two", "three")	"three" is assigned to stuff
$stuff=@stuff;	$stuff is assigned to length of the array i.e 3
@releases= ("alpha", "beta", "gamma",)	Comma is allowed in the last.
$modificationtime=(stat($file))[9];	Modification time
$hexdigit=("a","b","c","d",)[$digit-10]	hexdigit
($day, $month, $year)=(localtime)[3, 4,5];	day/month/year
@days + 0;	implicit forces array into scalar
scalar(@days)	explicit r @days into scalar
@whatever=()	$#whatever=-1

List values and arrays (cont)

undef(@whatever)	recover its memory
scalar(@whatever)	$# whatever + 1;

Multiplicative operator

print “-“ x 80	prints row of dashes.
my @ones = (1) x 80;	list of 80 1’s;
@ones = (5) x @ones	#set all elements to 5

Array, Hash initialization:

my %hash	@hash(@keys) = (“ “) x @keys;
my @keys	@keys=qw(perl doc yyy);

Result:
$hash{perl}=””;
$hash{doc}=””;
$hash{yyy}=””

Short circuit operator

$a && $b or $a and $b
$a || $b or $a or $b
$a // $b #define OR

$pid=fok() // die ‘’can’t fork $!’’
$value = $hash{value} // “DEFAULT”

Conditional operator

my $a = $ok ? $a: $b
my @a=$ok? @a:@b;
($a_or_b ? $a:$b)=$c

miscellaneous

State : state variable is never initialized and restricted to scalar vairables.
use v5.18;
sub next_count{
state $counter=0;
return ++$counter;
}
print (next_count());
print (next_count());
print (next_count());
Output: 1,2,3
Our: access the global variable
sub check_warehouse {
our @Current_Inventory;
my $widget;
foreach $widget (@Current_Inventory) {
say "I have a $widget in stock today."; }
}
our keyword changes the global variable
local: it changes the variable value locally even if it uses the global name and it will not change the gloal variable value.
Subroutines

miscellaneous (cont)

To declare named subroutine without defining
Sub Name Sub Name Proto Sub Name Proto attr

Tricks2

```
@reflist = ( \$s, \@a,
\%h)
        Is equal to
@reflist=\($s, @a, %h)
@reflist=\(@x);
 Is equal to
@reflist=map(\$_}@x;
@reflist = \fx();
Is equal to
@reflist=map{\$_} fx();
@reflist=\(fx(), fy(),
fz());
Is equal to
@reflist=map{\$_} fx(),
fy(), fz();
@reflist=\localtime();
#ref to each of nine
time elements
$lastref = \localtime()
#ref to whether it’s
daylight saving time
```

variable as a a variable name

```
$$ref is scalar value of
$ref refers to
@$ref is array values of
$ref refers to
%$ref is hash values of
$ref referes to
$foo = “three humps”;
$scalarref = \$foo;
$camel_model= $$scalarref;

push(@$arrayref, $filename);
$$arrayref[0]=”Janualry”
; set the first element
of @$arrayref;
@$arrayref[4…6] = qw
/may June July/;
%$hashref =
(KEY=>”RING”,
BIRD=>”Sing”)
$$hashref{KEY}=”VALUE”;
@$hashref{“KEY1”,
“KEY2”}=(“VAL1”,
“VAL2”);
$refrefref=\\\”hody”;
print ($$$$refrefref);
```

Sigil => $ (dollar means sigil)

Hashes

```
my %map=(“aa”, 1,
“bb”, 2);
  or
my %map=(aa=>1,
bb=>2,);
 or
$map{aa}=1;
$map{bb}=2;
Reference:
 $map={ “aa”. => 1,
“bb”=>2 }
```

Dereference:
%m=%$map;
print($m{aa});
Or
print($map->{aa});

Type glob and file handles

*foo contains values of $foo, @foo, %foo, $foo,
Typeglob stores entire symbol table entry
foo=bar;
It makes everything named foo. synonymous to every corresponding thing name bar
*foo=\$bar;
Makes an foo an alias of bar but doesn’t make it for @foo an alias for @bar.

Backtic operator

$perl_info=qx(ps $$); #thats perl

$k = `ls`; Interpreted in shell and store the result in $k
$shell_info=qx’ps. $$’; #thats shell

Line Input Operator

While($_=defined(<STDIN>)){
print } or
print $_ while defined($_=<-STDIN>);
$one_line=<MYFILE>
@alllines=<MYFILE>

Filename Globbing operator

```
@files =<*.xml>
    To
Filehandle in a
variable
<$foo>
@files=glob(*”.xml”);
while(glob *.c){
chmod 0644 $_
}
@files=gob(“$dir/*.
[ch]”);
```

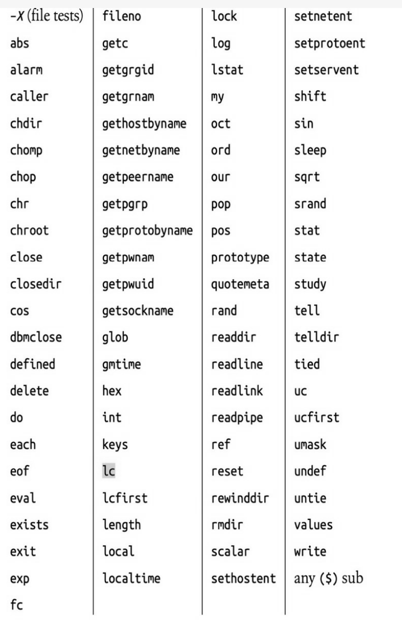
Statement

Loop Control:
last, next, redo
LINE:while(<STDIN>){
last LINE if ~/^$/;
eval { }; print ($@) #captures error

Statement (cont)

Scoped variables :

Named Unary operators



-X (file tests)	fileno	lock	setnetent
abs	getc	log	setprotoent
alarm	getgrgid	lstat	setservent
caller	getgrnam	my	shift
chdir	gethostbyname	oct	sin
chomp	getnetbyname	ord	sleep
chop	getpeername	our	sqrt
chr	getpgrp	pop	srand
chroot	getprotobyname	pos	stat
close	getpwnam	prototype	state
closedir	getpwuid	quotemeta	study
cos	getsockname	rand	tell
dbmclose	glob	readdir	telldir
defined	gmtime	readline	tied
delete	hex	readlink	uc
do	int	readpipe	ucfirst
each	keys	ref	umask
eof	lc	reset	undef
eval	lcfirst	rewinddir	untie
exists	length	rmdir	values
exit	local	scalar	write
exp	localtime	sethostent	any ($) sub
fc			

To declare and define named sub routine

Sub Name Block
Sub Name Proto Block
Sub Name Proto attr Block

Subroutines without names:
save return values at compile time, but also returns value
$subref = sub Block;

To import subroutines

use MODULE qw(Name1 Name2 Name3)

call subroutine

Name(LIST)	& is optional with parameter or Name LIST
Name	exposes current @_ to that subroutine

Indirect call

&$subref(LIST)
&$subref->(LIST),
&$subref

Semantics

```
Sub razzle{
print “Ok”
}
```

Default parameter @_;
Which is local array
$_[0]., $_[1] are the first two elements of array

Inline constant function

```
sub FLAG_FOO () { 1 <<
8 }
sub FLAG_BAR () { 1 <<
9 }
sub FLAG_MASK () {
FLAG_FOO | FLAG_BAR }
sub OPT_GLARCH(){
{ (0x1B58 & FLAG_MASK)
== 0 } {
{ return 23 }
{ return 42 }
```

Inline constant function (cont)

```
}
```

To return hash memory

```
Sub hashem{ +{@_} }
Sub hashem{ return{@_}}
```

Anonymous subroutine composer

$coderef = sub {print “Boink!\n”};

Semicolon is to terminate $coderef;
Sub { } > code inside is not executed immediately, it just generates reference to code.

References

References are like & in C
Anonymous hash with braces
$hashref={
“adam” => “eve”,
“clde”=>$bonnie
}

Hash of array

```
%table= (
'john' =>[47, 'brown',186],
'mary '=>[23, 'hazel', 128]
);
print @{$table{john}}[0];
```

prints 47