Perl cheat sheet

Brian McGill - Jan 9, 2000

Datatypes & variables

3 data types: $scalar (may be float number or string) @array %hash
access arrays: @var → $var[n] or $var[n][m] where n,m are 0-based @var[3,5,9] $#var (gives size)
access hash: %var → $var{'red'} or $var{red} @var{'red','blue'}
create array: @var=(1,3,9) @var=(1 .. 4) **create hash:** %h={red=>1, blue=>2}
referencing/pointers: to take pointer: $ptr=\$var to deref pointer: $$ptr
constants: 123,123.4,123.4E5,0xff,0377 'abc' (literal) "abc" (vars and \n substituted)
`command` (interpret command) <<filename (insert file contents)

Operators

precedence is decreasing from left to right

->	++	**	\	=~	*	+	<<	function	<	==	&	|	&&	||	..	?:	=	not	and	or
	--		!	!~	/	-	>>		>	!=		^			…		+=			xor
			~		%	.			<=	<=>							-=			
			+		x				>=	eq							*=			
			-						lt gt	ne							/=			
									le ge	cmp							%=			

** is exponentiation x is string replication . is string concatenation eq/lt etc is string comparison

Flow control

block:= { statement; statement; …}
expr1 **if** expr2; expr1 || expr2; expr1 ? expr2 : expr3;
if (expr) block [[**elsif** (expr) block …] **else** block]
[label:] **while** (expr) block [**continue** block]
[label:] **until** (expr) block [**continue** block]
[label:] **for** (expr; expr; expr) block
[label:] **foreach** var list block [**continue** block]
do block **while** expr;
do block **until** expr;
goto label;
last [label]; exits loop (skips continue)
next [label]; skips rest of loop including continue & starts next iteration
redo [label]; restarts w/o evaluating the conditonal or continue

sub name { local($arg1,$arg2,…)=@_; statements;} local automatically creates local vars & parses out @_
&name(arg1,arg2,arg3) calls subroutine
@aryout=map {block} @aryin or @aryout=map expr, @aryin
foreach [$var] (list) { block w/ var or $_ }

Strings

substr(str,start[,len])	takes subset	lc(str), uc(str) **nb:** no arrays
index(str,substr[,start])	searches for a substring	ucfirst(uc(str)), lcfirst(lc(str))
chomp(str)	removes trailing \n	chop(str) removes last char
$name =~ /^(.*)\b\s*$/; $name=$1	strips trailing spaces	chomp(str) removes last \n
$name =~ s/\s+//g	removes white space	length(str)
@a=split [/pattern/[,string[,limit]]]	splits (drops trailing null unless limit<0)	use eq, ne, gt for comparisons
$s=join 'delim',@a	repack split	"" eval $n, \n '' don't
val=shift @array; push @array val/list; val=pop @array;		. concats x n duplicates

Debugging

perl –d program		l [min+#|min-max]	list
t	stack trace	/pattern/ search	
s	step in	?pattern?	back search
n	step over	b line|subroutine	break
<cr>	repeat	D	clear all breaks
r	return from sub	W expr	watch
c [line|sub]	continue	x expr	display

Files

open(filehandle,"*filename") [||die("Cannot open file");] where * may be: [>] for read, "> for write, '>>' for append
close filehandle
STDIN is a predefined filehandle
$line=<filehandle>; reads 1 line
@lines=<filehandle>; reads all lines
<filehandle>; reads 1 line into the var $_
<> reads 1 line into the var $_ from the files listed on the perl command line
print [filehandle] expr1, expr2, …; **nb:** no comma after filehandle
print [filehandle] <<LABEL line of text line of text …. LABEL (prints multiline)
printf [filehandle] "fmtstr",var1,var2, …; as per C formats
system("command","arg1","arg2") runs a program @results=`command arg1 arg2`; gets output

Regular expressions

/regexp/	compares $_ to the expression and returns a boolean (e.g. print $_ if /regexp/;)
$str =~ /regexp/	identical but use anystring, not just $_
$str !~ /regexp/	as above but returns true if no match
/regexp/g	returns array of all matches
s/pattern/substitution/	goes through $_ looking for pattern and replaces it with substitution
$str =~ s/pat/subst/	as above but for any string
/regexp/i	case insensitive match
/regexp/g	matches repeatedly
s/pattern/nextext/g	search and replace globally
/regexp/x	allow whitespace and #comments in the pattern (which now must be \ to use)

regexp/ms m causes multiline eval (^ & $ match newlines) vs s causing singleline (. matches newline)
/reg(exp)/;$1 subexpressions grouped in parenthesis can be matched in later statements by $1, $2, etc

Anchors	Characters		Quantifiers (of preceding)	
$ matches end of line	.	any character	*	0 or more
^ matches beginning of line	\n \r \t	control codes	+	1 or more
	[adg-l]	one of character	?	0 or 1
	[^abc]	any character not in list	{n}	exactly n times
	abc	characters in order	{n,}	at least n times
	red|blue|green	one of ordered characters	{n,m}	between n & m times
	\w	word (alphanumeric & _)	{,m}	at most m times
	\W	not word (alphanum & _)	quant?	modifiers quantifier to
	\s \S	white space (non)		not be "greedy" i.e.
	\d \D	digit (non)		match a minimum # of
	\b \B	word boundary (non)		times –e.g. *?

Global variables

$_ cur argument $! OS error $@ eval error $^O OS name @ARGV command line @INC (includes)
Filehandles: STDIN STDERR STDOUT REGS: $n $` $& $' (pre/match/post) $+ last parens

Canonical program

```
open(FILE,"perltest.txt") || die("Cannot open file. $!\n");
open(FILEOUT, "<myout.txt");
<FILE>; #skip header line
while ($ln=<FILE>) {
        next if $ln !~ /\S/; #skip blank lines
        @flds=split(/\s+/,$ln);  #/,/ to split by comma delim
        @flds=map(ucfirst,map(lc,@flds));
        $fldct=$#flds;
        if ($flds[0] eq "special") {
                print "Special record\n";
                next;
        }
        print join(',',@flds),"\n";  #print FILEOUT join(',',@flds),"\n"; #no ,
}
close(FILE);
```