Perl Reference Card

(cl) 2006 Michael Goerz <goerz@physik.fu-berlin.de>
http://www.physik.fu-berlin.de/~goerz/
Information taken liberally from the perl documentation and various other sources.

1 Variable Types

1.1 Scalars and Strings

`chomp($str);`	discard trailing `\n`
`$v = chop($str);`	$v becomes trailing char
`eq, ne, lt, gt, le, ge, cmp`	string comparison
`$str = ‘‘0’’ x 4;`	`$str` is now ‘‘0000’’
`$v = index($str, $x);`	find index of `$x` in `$str`,
`$v = rindex($str, $x);`	starting from left or right
`$v = substr($str, $start, $len);`	extract substring
`$cnt = $sky =~ tr/0-9//;`	count the digits in `$sky`
`$str =~ tr/a-zA-Z/ /cs;`	change non-alphas to space
`$v = sprintf(‘‘%10s %08d’’, $s, $n);`	format string

Format String: `%[flags][0][width][.precision][mod]type`

types:

`c`	character
`d(i)`	signed decimal int
`e(E)`	scientific notation
`f`	decimal floating point
`g`	shorter `%e` or `%f`
`G`	shorter `%E` or `%f`
`o`	signed octal
`s`	string of chars
`u`	unsigned decimal int
`x`	unsigned hex int
`X`	unsigned hex int in capitals
`p`	address pointer
`n`	nothing printed

modifiers:

`h`	arg is short int
`l`	arg is long int/double
`L`	arg is long double

More: `chr, crypt, hex, lc, lcfirst, length, oct, ord, pack, q/STRING/, qq/STRING/, reverse, uc, ucfirst`

1.2 Arrays and Lists

`@a = (1..5);`	array initialization
`$i = @a;`	number of elements in `@a`
`($a, $b) = ($b, $a);`	swap `$a` and `$b`
`$x = $a[1];`	access to index 1
`$i = $#a;`	last index in `@a`
`push(@a, $s);`	appends `$s` to `@a`
`$a = pop(@a);`	removes last element
`chop(@a);`	remove last char (per el.)
`$a = shift(@a);`	removes first element
`reverse(@a);`	reverse `@a`
`@a = sort{$ela <=> $elb}(@a);`	sort numerically
`@a = split(/-/,$s);`	split string into `@a`
`$s = join(‘‘::’’ @c);`	join `@a` elements into string
`@a2 = @a[1,2,6..9];`	array slice
`@a2 = grep(!/^#/, @a);`	remove comments from `@a`

1.2 Hashes

`%h = ( k1 => ‘‘val1’’, k2 => 3);`	hash initialization
`$val = $map{k1};`	recall value
`@a = %h;`	array of keys and values
`%h = @a;`	create hash from array
`foreach $k (keys(%h)){...}`	iterate over list of keys
`foreach $v (values(%h)){...}`	iterate over list of values
`while (($k,$v) = each %h){...}`	iterate over key-value-pairs
`delete $h{k1};`	delete key
`exists $h{k1}`	does key exist?
`defined $h{k1}`	is key defined?

2 Basic Syntax

`($a, $b) = shift(@ARGV);`	read command line params
`sub p{ my $var = shift; ...}`	define subroutine
`p(‘‘bla’’);`	execute subroutine
`if(expr){} elsif {} else {}`	conditional
`unless (expr){}`	negative conditional
`while (expr){}`	while-loop
`until (expr){}`	until-loop
`do {} until (expr)`	postcheck until-loop
`for($i=1; $i<=10; $i++){}`	for-loop
`foreach $i (@list){}`	foreach-loop
`last, next, redo`	ends loop, skips to next, jumps to top
`eval {$a=$a/$b; }; warn $@ if $@;`	exception handling

3 References and Data Structures

`$aref = \@a;`	reference to array
`$aref = [ 1, "foo", undef, 13 ];`	anonymous array
`$href = { APR => 4, AUG => 8 };`	anonymous hash
`$el = @{$aref}[0];`	access element of array
`$aref2 = [@{$aref1}];`	copy array
`$href2 = {%{$href1}};`	copy hash
`if (ref($r) eq "HASH") {}`	checks if $r points to hash
`@a = ([1, 2],[3, 4]);`	2-dim array
`$i = $a[0][1];`	access 2-dim array
`%HoA = (fs=>["fred","barney"],` `sp=>["homer","marge"]);`	hash of arrays
`$name = $HoA{sp}[1];`	access to hash of arrays

4 System Interaction

`system(‘‘cat $f | sort -u | $f.s’’);`	system call
`@a = readpipe(‘‘lsmod’’);`	catch output
`$today = ‘‘Today: ‘‘.`date`;`	catch output
`chroot(‘‘/home/user/’’);`	change root
`while (<*.c>) {}`	operate on all c-files
`unlink(‘‘/tmp/file’’);`	delete file
`if (-f ‘‘file.txt’’){...}`	file test

File Tests:

`-r, -w`	readable, writeable
`-x`	executable
`-e`	exists
`-f, -d, -l`	is file, directory, symlink
`-T, -B`	text file, binary file
`-M, -A`	mod/access age in days
`@stats = stat(‘‘filename’’);`	13-element list with status

More: `chmod, chown, chroot, fcntl, glob, ioctl, link, lstat, mkdir, opendir, readlink, rename, rmdir, symlink, umask, utime`

5 Input/Output

`open(INFILE,"in.txt") or die;`	open file for input
`open(INFILE, "<:utf8", "file");`	open file with encoding
`open(TMP, "+>", undef);`	open anonyomous temp file
`open(MEMORY,'>', \$var);`	open in-memory-file
`open(OUTFILE,">out.txt") or die;`	open output file
`open(LOGFILE,">>my.log") or die;`	open file for append
`open(ARTICLE, "caesar <$art |");`	read from process
`open(EXTRACT, "|sort >Tmp$$");`	write to process
`$line = <INFILE>;`	get next line
`@lines = <INFILE>;`	slurp infile
`foreach $line (<STDIN>){...}`	loop of lines from `STDIN`
`print STDERR "final warning.\n";`	print to `STDERR`
`close INFILE;`	close filehandle

More: `binmode, dbmopen, dbmclose, fileno, flock, format, getc, read, readdir, readline, rewinddir, seek, seekdir, select, syscall, sysreed, sysseek, tell, telldir,truncate, pack, unpack, vec`

6 Regular Expressions

`($var =~ /foo/)`	matches
`($var !~ /foo/)`	does not match
`m/pattern/igmsoxc`	matching pattern
`qr/pattern/imsox`	store regex in variable
`s/pattern/replacement/igmsoxe`	search and replace

Modifiers:

`i`	case-insensitive	`o`	compile once
`g`	global	`x`	extended
`m`	multiline	`c`	don’t reset pos (with `g`)
`s`	as single line (. matches `\n`)	`e`	evaluate replacement

Syntax:

`\`	escape
`.`	any single char
`^`	start of line
`$`	end of line
`*, *?`	0 or more times (greedy / nongreedy)
`+, +?`	1 or more times (greedy / nongreedy)
`?, ??`	0 or 1 times (greedy / nongreedy)
`\b`	word boundary (`\w` - `\W`)
`\B`	match except at word b.
`\A`	string start (with `/m`)
`\Z`	string end (before `\n`)
`\z`	absolute string end
`\G`	continue from prev `m//g`
`[...]`	character set
`(...)`	group, capture to `$1, $2`
`(?:...)`	group without capturing
`{n,m} , {n,m}?`	at least n times, at most m times
`{n,}  , {n,}?`	at least n times
`{n}  , {n}?`	exactly n times
`|`	or
`\1, \2`	text from nth group (`$1`, ...)

Escape Sequences:

`\a`	alarm (beep)	`\e`	escape
`\f`	formfeed	`\n`	newline
`\r`	carriage return	`\t`	tab
`\cx`	control-x	`\l`	lowercase next char
`\L`	lowercase until `\E`	`\U`	uppercase until `\E`
`\Q`	diable metachars until `\E`	`\E`	end case modifications

Character Classes:

`[amy]`	'a', 'm', or 'y'
`[f-j.-]`	range f-j, dot, and dash
`[^f-j]`	everything except range f-j
`\d`	digit `[0-9]`
`\D`	nondigit `[^0-9]`
`\w`	word char `[a-zA-Z0-9_]`
`\W`	nonword char
`\s`	whitepace `[ \t\n\r\f]`
`\S`	non-whitespace
`\C`	match a byte
`\pP`	match p-named unicode
`\p{...}`	match long-named unicode
`\PP`	match non-P
`\P{...}`	match non-P{...}
`\X`	match extended unicode

Posix:

`[:alnum]`	alphanumeric
`[:alpha]`	alphabetic
`[:ascii:]`	any ASCII char
`[:blank:]`	whitespace `[ \t]`
`[:cntrl:]`	control characters
`[:digit:]`	digits
`[:graph:]`	alphanum + punctuation
`[:lower:]`	lowercase chars
`[:print:]`	alphanum, punct, space
`[:punct:]`	punctuation
`[:space:]`	whitespace `[\s\ck]`
`[:upper:]`	uppercase chars
`[:word:]`	alphanum + '_'
`[:xdigit:]`	hex digit
`[:^digit:]`	non-digit

Extended Constructs

`(?#text)`	comment
`(?imxs-imsx:...)`	enable/disable option
`(?=...)`	positive lookahead
`(?!...)`	negative lookahead
`(?<=...)`	positive lookbehind
`(?<!...)`	negatie lookbehind
`(?>...)`	prohibit backtracking
`(?{ code })`	embedded code
`(??{ code })`	dynamic regex
`(?(cond)yes|no)`	cond corres. to capt. parens
`(?(cond)yes)`	cond corres. to lookaround

Variables

`$&`	entire matched string
`$` `	evt. prior to matched string
`$'`	evt. after matched string
`$1, $2 ...`	nth captured expression
`$+`	last parenth. pattern match
`$^N`	most recently closed capt.
`$^R`	result of last `(?{...})`
`@-`	offsets of starts of groups
`@+`	offsets of ends of groups

7 Object-Oriented Perl and Modules

Defining a new class:

```
package Person;
use strict;
sub new { #constructor, any name is fine
    my $class = shift;
    my $self  = {};
    $self->{NAME}   = undef; # field
    $self->{"_CENSUS"} = \$Census; # class data
    ++ ${ $self->{"_CENSUS"} };
    bless ($self, $class);
    return $self;
}
sub name { #method
    my $self = shift;
    if (@_) { $self->{NAME} = shift }
    return $self->{NAME};
}
sub DESTROY { #destructor
    my $self = shift;-- ${ $self->{"_CENSUS"} }; }
1;  # so the require or use succeeds
```

Using the class:

```
use Person;
$him = Person->new();
$him->name("Jason");
printf "There's someone named %s.\n", $him->name;
use Data::Dumper; print Dumper($him); # debug printout
```

Installing Modules: `perl -MCPAN -e shell;`

8 One-Liners

`-0`	(zero) specify the input record separator
`-a`	split data into an array named @F
`-F`	specify pattern for -a to use when splitting
`-i`	edit files in place
`-n`	run through all the @ARGV arguments as files, using <>
`-p`	same as -n, but will also print the contents of $_

Examples:

1. just lines 15 to 17, efficiently
   `perl -ne 'print if $. >= 15; exit if $. >= 17;'`
2. just lines NOT between line 10 and 20
   `perl -ne 'print unless 10 .. 20'`
3. lines between START and END
   `perl -ne 'print if /^START$/ .. /^END$/'`
4. in-place edit of *.c files changing all foo to bar
   `perl -pi.bak -e 's/\bfoo\b/bar/g' *.c`
5. delete first 10 lines
   `perl -i.old -ne 'print unless 1 .. 10' foo.txt`
6. change all the isolated oldvar occurrences to newvar
   `perl -i.old -pe 's{\boldvar\b}{newvar}g' *.[chy]`
7. printing each line in reverse order
   `perl -e 'print reverse <>' file1 file2 file3 ....`
8. find palindromes in the /usr/dict/words dictionary file
   `perl -lne '$_ = lc $_; print if $_ eq reverse' /usr/dict/words`
9. command-line that reverses all the bytes in a file
   `perl -0777e 'print scalar reverse <>' f1 f2 f3`
10. word wrap between 50 and 72 chars
    `perl -p000e 'tr/ \t\n\r/ /; s/(.{50,72})\s/$1\n/g;$_."\n"x2'`
11. strip and remove double spaces
    `perl -pe '$_ = " $_ "; tr/ \t/ /s; $_ = substr($_,1,-1)'`
12. move '*.txt.out' to '*.out'
    `perl -e '($n = $_) =~ s/\.txt(\.out)$/$1/ and not -e $n and rename $_, $n for @ARGV' *`