Perl Text-processing language

START HERE

`perl -e '<code>'`
Run code from the command line

`perl -i.bak -pe '...' <file>`
Edit in place, keep backup

`perl <script.pl>`
Run a script

`perl -lane 'print $F[0]' <file>`
Print first column

`perl -ne 'print if /<re>/' <file>`
Grep lines

`perl -c <script.pl>`
Syntax check only

`perl -pe 's/<a>/<b>/g' <file>`
Substitute on every line

`perldoc -f <function>`
Look up a function

Input files or STDIN → `-n` / `-p` loop → **$_** current line → `-a` / `-F` → **@F** split fields → your code → **Output** print or -i →

CONCEPTS

Start scripts with use strict; use warnings; (or use v5.36;) and declare every variable with my.

Sigils

Sigil	Holds	Element access
`$x`	scalar	$x
`@a`	array	$a[0]
`%h`	hash	$h{key}
`&f`	sub	f(...)
`\@a`	reference	$r→[0]

Only double-quoted strings interpolate variables and escapes; single quotes are literal.

Context decides results: an array in scalar context gives its length, in list context its elements.

ONE-LINERS

`perl -e '<code>'`
Run code given on the command line

`perl -E '<code>'`
Same, with say and modern features on

`perl -ne '<code>'`
Run code once per input line

`perl -pe '<code>'`
Per line, then print $_

`perl -i[.bak] -pe '<code>' <file>`
Edit files in place
Without a suffix no backup is kept. Test without -i first.

`perl -l`
Chomp input lines, add \n to print

`perl -a -F<sep>`
Split each line into @F

`perl -0777 -ne '<code>'`
Slurp whole file as one string

`perl -M<Module> -e '<code>'`
Load a module first

`perl -c <script.pl>`
Check syntax without running

`perl -pe 'BEGIN{...} END{...}'`
Run setup or summary around the loop

`perl -Mutf8 -CS -e '<code>'`
Treat source and I/O as UTF-8

`$.`
Current input line number

ONE-LINERS (cont.)

`$0 / @ARGV / %ENV`
Script name, arguments, environment

VARIABLES AND DATA

`my $x = <value>;`
Declare a lexical scalar

`my @a = (<1>, <2>, <3>);`
Declare an array

`my %h = (<k> => <v>);`
Declare a hash

`$a[0] / $a[-1]`
First / last element

`scalar @a / $#a`
Element count / last index

`push @a, <x> / pop @a`
Add / remove at the end

`unshift @a, <x> / shift @a`
Add / remove at the start

`@a[<1>,<2>] / @h{<k1>,<k2>}`
Array / hash slice

`$h{<k>}`
Hash element

`keys %h / values %h`
All keys / all values

`exists $h{<k>} / delete $h{<k>}`
Test for / remove a key

`sort { $a <=> $b } @a`
Numeric sort

`sort { $a cmp $b } @a`
String sort

`map { <expr> } @a`
Transform each element

`grep { <cond> } @a`
Keep matching elements

`join(<sep>, @a) / split(/<re>/, $s)`
Join / split

`\@a / \%h / sub {...}`
Reference to array / hash / code

`$r→[0] / $r→{<k>} / @$r`
Dereference

`{ <k> => [1,2] }`
Anonymous nested structure

`defined $x`
True if not undef

STRINGS AND OPERATORS

`"Hi $name @{[ f() ]}"`
Interpolate variables and expressions

STRINGS AND OPERATORS (cont.)

`$a . $b / $s x <n>`
Concatenate / repeat

`== != < > <=>`
Numeric compare

`eq ne lt gt cmp`
String compare

`&& || // !`
Logical and, or, defined-or, not

`$x //= <default>`
Set only if undef

`length $s / uc $s / lc $s`
Length, upper, lower

`substr($s, <off>, <len>)`
Extract or replace a substring

`index($s, <t>) / rindex($s, <t>)`
Find substring position

`sprintf("%-10s %5.2f", ...)`
Format values

`chomp $s / chop $s`
Strip newline / last char

REGEX

`$s =~ /<re>/`
Test for a match

`$s =~ s/<re>/<new>/g`
Substitute all

`($a,$b) = $s =~ /(\d+)-(\d+)/`
Capture into variables

`$1 $2 $&`
Capture groups, whole match

`my @all = $s =~ /<re>/g`
All matches in list context

`/i /x /m /s`
Ignore case, extended, multiline, dot-all

`\d \w \s \b`
Digit, word, space, boundary

CONTROL FLOW AND SUBS

`if (<c>) {...} elsif (<c>) {...} else {...}`
Conditional

`<stmt> if <c>; / <stmt> unless <c>;`
Postfix conditional

`for my $x (@a) {...}`
Loop over a list

`while (<c>) {...}`
Loop on a condition

`next / last / redo`
Skip / exit / repeat iteration

CONTROL FLOW AND SUBS (cont.)

`sub <name> { my ($a,$b) = @_; return ...; }`
Define a sub

`die "<msg>\n" / warn "<msg>"`
Raise error / warn

`eval { ... }; if ($@) {...}`
Catch exceptions

FILES AND I/O

`open(my $fh, '<', $f) or die "$!";`
Open for reading

`open(my $fh, '>', $f) / '>>'`
Open for write / append

`while (my $l = <$fh>) {...}`
Read line by line

`print $fh "text\n";`
Write to a handle

`-e -f -d -r -s $path`
File tests: exists, file, dir, readable, size

MODULES AND CPAN

`use <Module>;`
Load a module

`cpanm <Module>`
Install from CPAN

`perldoc <Module>`
Read module docs

`perldoc -f <func> / -q <faq>`
Function docs / FAQ search

`use Data::Dumper; print Dumper($x);`
Dump a data structure

RECIPES

Sum a column

```
perl -lane '$s += $F[2]; END { print $s }' <file>
```

Rename across many files

```
perl -i.bak -pe 's/\bfoo\b/bar/g' *.txt
```

Word frequency

```
perl -lne '$c{$_}++ for /\w+/g; END { print "$c{$_} $_" for sort keys %c }' <file>
```

Minimal script skeleton

```
use v5.36;
open(my $fh, '<', shift) or die $!;
while (<$fh>) {
    print if /error/i;
}
```