

Perl クイックリファレンス

変数、正規表現、ファイル I/O、リファレンス、モジュールの基本

基本

Hello World

```
#!/usr/bin/perl
use strict;
use warnings;
print "Hello, World!\n";
say "Hello, World!"; # with use feature 'say';
```

Perl の実行

```
perl script.pl # run a file
perl -e 'print "hi\n"' # run inline
perl -ne 'print' file # process file line by line
```

コメントとドキュメント

```
# single-line comment
=pod
Multi-line POD documentation
=cut
```

変数

シジル

\$scalar	単一の値 (文字列、数値、リファレンス)
@array	スカラーの順序付きリスト
%hash	キーと値のペア
\$array[0]	単一の配列要素にアクセス (スカラーコンテキスト)
\$hash{key}	単一のハッシュ値にアクセス (スカラーコンテキスト)

スカラー変数

```
my $name = "Perl"; # string
my $version = 5.40; # number
my $count = 42; # integer
my $undef; # undefined (undef)
my $combined = "$name v$version"; # interpolation
```

コンテキスト

```
my @arr = (1, 2, 3);
my $count = @arr; # scalar context: 3
my @copy = @arr; # list context: (1, 2, 3)
my $len = scalar @arr; # force scalar context
```

特殊変数

\$_	デフォルト変数 (トピック)
@_	サブルーチンの引数
!	システムエラーメッセージ
\$@	eval からのエラー
\$0	プログラム名
@ARGV	コマンドライン引数
%ENV	環境変数

演算子

比較演算子

==, !=, <, >, <=, >=	数値比較
eq, ne, lt, gt, le, ge	文字列比較
<>	数値スペースシップ (-1、0、1 を返す)
cmp	文字列スペースシップ
 =~	正規表現マッチ/バインド
 ! =~	正規表現の否定マッチ

文字列演算子

```
my $full = "Hello . " . "World"; # concatenation
my $line = "-" x 40; # repetition
my $len = length($full); # 11
```

論理演算子

&& / and	論理 AND (低優先度: and)
 / or	論理 OR (低優先度: or)
//	定義済み-or (左辺が定義済みなら返す)
! / not	論理 NOT
? :	三項条件演算子

制御フロー

条件文

```
if ($x > 0) { print "positive\n"; }
elsif ($x == 0) { print "zero\n"; }
else { print "negative\n"; }
print "yes\n" if $condition; # postfix if
print "no\n" unless $condition; # postfix unless
```

ループ

```
for my $i (0..9) { print "$i\n"; }
foreach my $item (@array) { print "$item\n"; }
while ($line = <STDIN>) { chomp $line; }
until ($done) { last if check(); }
```

ループ制御

next	次の反復へスキップ (continue 相当)
last	ループを終了 (break 相当)
redo	現在の反復を再スタート
next LABEL	ラベル付きループの次の反復へスキップ
last LABEL	ラベル付きループを終了

Given / When

```
use feature 'switch';
given ($status) {
    when ("ok") { say "success"; }
    when ("error") { say "failed"; }
    default { say "unknown"; }
}
```

サブルーチン

基本的なサブルーチン

```
sub greet {
    my ($name) = @_;
    return "Hello, $name!";
}
my $msg = greet("Alice");
```

デフォルトと名前付きパラメータ

```
sub connect {
    my (%opts) = @_;
    my $host = $opts{host} // "localhost";
    my $port = $opts{port} // 5432;
    return "$host:$port";
}
connect(host => "db.example.com", port => 3306);
```

サブルーチンリファレンス

```
my $double = sub { return $_[0] * 2; };
print $double->(5); # 10
my @sorted = sort { $a <> $b } @nums;
```

プロトタイプとシグネチャ

```
use feature 'signatures';
sub add($a, $b) { return $a + $b; }
sub greet($name, $greeting = "Hello") {
    return "$greeting, $name!";
}
```

正規表現

マッチング

```
if ($str =~ /pattern/) { print "matched\n"; }
if ($str =~ /(\d+)/) { print "number: $1\n"; }
my @matches = ($str =~ /(\w+)/g); # all matches
```

置換

```
$str =~ s/old/new/; # first occurrence
$str =~ s/old/new/g; # global (all occurrences)
$str =~ s/\s+|\s+$//g; # trim whitespace
(my $clean = $str) =~ s/\W//g; # non-destructive copy
```

修飾子

/i	大文字小文字を区別しない
/g	グローバル (全マッチ)
/m	複数行 (^と\$が行境界にマッチ)
/s	単一行 (.が改行にマッチ)
/x	拡張 (空白とコメントを許可)

よくあるパターン

\d, \D	数字 / 非数字
\w, \W	単語文字 / 非単語文字
\s, \S	空白 / 非空白
\b	単語境界
(?: ...)	非キャプチャグループ
(?<name> ...)	名前付きキャプチャ (\${name} でアクセス)

ファイル I/O

オープンと読み込み

```
open(my $fh, '<', 'data.txt') or die "Cannot open: $!";
while (my $line = <$fh>) {
    chomp $line;
    print "$line\n";
}
close($fh);
```

書き込みと追記

```
open(my $fh, '>', 'out.txt') or die "Cannot open: $!";
print $fh "Hello\n";
close($fh);
open(my $fh, '>>', 'log.txt') or die "Cannot open: $!";
print $fh "entry\n";
close($fh);
```

ファイル全体を読み込む

```
use File::Slurp;
my $content = read_file('data.txt');
my @lines = read_file('data.txt', chomp => 1);
```

Perl クイックリファレンス

ファイルテスト

-e \$path	ファイルが存在する
-f \$path	通常のファイルである
-d \$path	ディレクトリである
-r / -w / -x	読み取り可 / 書き込み可 / 実行可
-s \$path	ファイルサイズ (バイト、空の場合は 0)
-z \$path	ファイルサイズがゼロ

配列とハッシュ

配列

```
my @arr = (1, 2, 3, 4, 5);
push @arr, 6;           # append
my $last = pop @arr;   # remove last
my $first = shift @arr; # remove first
unshift @arr, 0;       # prepend
my @slice = @arr[1..3]; # slice
```

配列関数

scalar @arr	要素数
push / pop	末尾に追加/末尾から削除
shift / unshift	先頭から削除/先頭に追加
splice(@a, 2, 1)	インデックス 2 の要素を 1 つ削除
sort @arr	アルファベット順でソート
reverse @arr	逆順にする
grep { /pat/ } @arr	パターンでフィルタリング
map { \$_ * 2 } @arr	各要素を変換
join(',', @arr)	文字列に結合

ハッシュ

```
my %user = (name => "Alice", age => 30);
$user{email} = "a@b.com"; # add pair
delete $user{age};        # remove pair
my @keys = keys %user;
my @vals = values %user;
```

ハッシュの反復

```
while (my ($k, $v) = each %hash) {
    print "$k => $v\n";
}
for my $key (sort keys %hash) {
    print "$key: $hash{$key}\n";
}
```

リファレンス

リファレンスの作成

```
my $scalar_ref = \$name;
my $array_ref  = \@arr;
my $hash_ref   = \%hash;
my $anon_arr   = [1, 2, 3]; # anonymous array ref
my $anon_hash  = {a => 1, b => 2}; # anonymous hash ref
```

デリファレンス

```
print $$scalar_ref; # dereference scalar
print $array_ref->[0]; # arrow notation
print $hash_ref->{key}; # arrow notation
my @copy = @$array_ref; # dereference to array
my %copy = %$hash_ref; # dereference to hash
```

複雑なデータ構造

```
my @users = (
    { name => "Alice", age => 30 },
    { name => "Bob",   age => 25 },
);
print $users[0]->{name}; # "Alice"
```

ref()関数

ref(\$r) eq 'SCALAR'	スカラーへのリファレンス
ref(\$r) eq 'ARRAY'	配列へのリファレンス
ref(\$r) eq 'HASH'	ハッシュへのリファレンス
ref(\$r) eq 'CODE'	サブルーチンへのリファレンス

モジュール

モジュールの使用

```
use strict;
use warnings;
use List::Util qw(sum max min);
use File::Basename;
use Cwd qw(abs_path);
```

モジュールの作成

```
# MyModule.pm
package MyModule;
use Exporter 'import';
our @EXPORT_OK = qw(helper);
sub helper { return "help"; }
1; # module must return true
```

よく使うコアモジュール

List::Util	sum、max、min、reduce、any、all
File::Basename	basename、dirname、fileparse
File::Path	make_path、remove_tree
Getopt::Long	コマンドラインオプションのパーズ
JSON	encode_json、decode_json
LWP::Simple	get(\$url) — シンプルな HTTP クライアント
Data::Dumper	データ構造のデバッグダンプ
Carp	croak、confess — より良いエラーメッセージ

CPAN

```
cpan install Module::Name # install from CPAN
cpanm Module::Name       # cpanminus (faster)
perldoc Module::Name     # read module docs
```