IO::File

```
use IO::File;

my $f = IO::File->open('/tmp/secrets', '<');
my $line = $f->getline();
$f->close;

my $g = IO::File->new_tmpfile;
$g->autoflush(1);
$g->print("Pozor, nepřítel naslouchá!\n");
...
```

IO::Socket

```
use IO::Socket::INET;

my $sk = IO::Socket::INET->new(
        Proto => 'tcp',
        PeerHost => 'kam.mff.cuni.cz',
        PeerPort => 80,
        Timeout => 60,
) or die;

$sk->print("GET / HTTP/1.1\r\n");
$sk->print("Host: kam.mff.cuni.cz\r\n");
$sk->print("Connection: close\r\n");
$sk->print("\r\n");
$sk->flush;

while (<$sk>) { print; }

$sk->close;
```

File::Spec

```
File::Spec->catfile('usr', 'share', 'games')
usr/share/games
File::Spec->splitpath('/usr/share/games')
("", '/usr/share/', 'games')
File::Spec->rel2abs('brum')
/nfs/home/mares/brum
File::Spec->abs2rel('/nfs/home/karry/xyzzy')
../karry/xyzzy
```

File::Find

```
use File::Find;

find({
        wanted => sub {
                print "Found $File::Find::name\n";
        },
        preprocess => sub {
                print "Entering $File::Find::dir\n";
                return sort @_;
        },
        postprocess => sub {
                print "Leaving $File::Find::dir\n";
        },
        bydepth => 1,
}, '/etc');
```

CPAN (Comprehensive Perl Archive Network)

http://www.cpan.org/
http://www.metacpan.org/

```
cpan
cpan> i Lingua::Romana::Perligata
cpan> install Lingua::Romana::Perligata

perl -Mlocal::lib -MCPAN -e shell

perl -Mlocal::lib
export PERL_LOCAL_LIB_ROOT="/home/mares/perl5";
export PATH="/home/mares/perl5/bin:$PATH";
...

cpanm MIME::Base64
cpanm -L ~/perl MIME::Base64
```

Moose – příklad na úvod

```
package Bear;
use Moose;
use namespace::autoclean;

extends 'Mammal';

has 'name' => ( is => 'rw', isa => 'Str', required => 1 );
has 'honey' => ( is => 'rw', isa => 'Num', default => 0 );

sub brum {
        my ($self) = @_;
        $self->voice('Brum!') if $self->honey >= 10;
}

__PACKAGE__->meta->make_immutable;


my $bear = Bear->new( name => 'brtník' );
$bear->brum;
```

Moose – atributy

```
has 'honey' => (
        is => 'rw',
        isa => 'Str',
        reader => 'get_honey',          # Přejmenování
        writer => 'set_honey',
        clearer => 'clear_honey',       # Jen na požádání
        predicate => 'has_honey',
        default => sub { rand(10) },
        lazy => 1,                      # Vznikne až při čtení
        init_arg => 'wealth',           # Argument konstruktoru
        trigger => \&honey_changed,
);
```

Moose – delegace

```
has 'claw' => (
        is => 'rw',
        isa => 'Bear::Claw',
        weak_ref => 1,
        handles => [ 'attack', 'clean' ],
);

has 'teeth' => (
        is => 'rw',
        isa => 'Bear::Teeth',
        handles = {
                clean_teeth => 'clean',
        },
);
```

Moose – modifikátory metod

```
override 'brum' => sub {
        print "Brum!\n";
};

before 'brum' => sub {
        my ($self) = @_;
        while ($self->honey < 10) {
                $self->eat;
        }
        # Návratová hodnota se ignoruje
};

after 'brum' => sub {
        print "DEBUG: Just brummed\n";
        # Návratová hodnota se ignoruje
};
```

Moose – další modifikátory metod

```
around 'brum' => sub {
        my $orig = shift;
        my $self = shift;
        if (rand(10) < 7) {
                return $self->$orig(@_);
        } else {
                print "Sleeping bears do not speak.\n";
                return;
        }
};
```

Moose – konstruktory

```
sub BUILD {
        my ($self) = @_;
        if ($self->name =~ m{\b von \b}x) {
                $self->honey(10000);
        }
}

around BUILDARGS => sub {
        my $orig = shift;
        my $class = shift;
        if (@_ == 1 && !ref $_[0]) {
                return $class->orig( name => $_[0] );
        } else {
                return $class->orig(@_);
        }
};
```

Moose – „typový systém“

```
Bool
Str
Num
Int
Bear::Paw
Maybe[Int]
ArrayRef[Any]
ArrayRef[Bear::Paw]
Bear | Cat
```

Moose – validace argumentů

```
use MooseX::Params::Validate;

sub speak {
    my $self = shift;
    my %p = validated_hash(
        \@_,
        loudness => { isa => 'PositiveInt', default => 1 },
        utterance => { isa => 'Str | ArrayRef[Str]' },
    );
    ...
}

use MooseX::StrictConstructor;
```

Moose – definice vlastních typů

```
use Moose::Util::TypeConstraints;

subtype 'PositiveInt',
        as 'Int',
        where { $_ > 0 },
        message { "The number $_ must be positive" };

coerce 'PositiveInt',
        from 'Str',
        via { hex };

has 'counter' => {
        is => 'rw',
        isa => 'PositiveInt',
        coerce => 1,
};
```

Moose – role

```
package Breakable;

use Moose::Role;

has 'is_broken' => ( is => 'rw', isa => 'Bool' );
sub break {
        my $self = shift;
        print "Oops!\n";
        $self->is_broken(1);
}


package Pipe;
use Moose;
with 'Breakable';
...
$obj->does('Breakable') or die;
```

Moose – role podruhé

```
package Unreliable;

use Moose::Role;

requires 'break';

before qr{^handle_} => sub {
        my ($self) = @_;
        if (rand(10) < 3) {
                $self->break;
        }
};


package RealPipe;
use Moose;
extends 'Pipe';
with 'Unreliable';
```

Moose – metatřídy (viz Class::MOP)

```
my $meta = Bear->meta;

for my $attr ($meta->get_all_attributes) {
        print $attr->name;
        print ' [rw]' if $attr->is eq 'rw';
        print "\n";
}

$meta->make_mutable;
$meta->add_attribute('size' => ( is => 'rw', isa => 'Int' ));
$meta->make_immutable;

say join(" ", $meta->get_method_names);
say join(" ", $meta->get_all_method_names);
say join(" ", $meta->linearized_isa);
```

Moose – alternativy

`Mouse` – *"Moose minus the antlers"*

- Syntaxe kompatibilní s podmnožinou Moose
- Částečně kompatibilní metaprotokol

`Moo` – *"Minimalist Object Orientation"*

- Syntaxe se od Moose občas odchyluje
- Žádný metaprotokol …
- … ale pokud nahrajete Moose, metaprotokol automaticky vznikne

Standardní knihovna: system

```
system 'ls', '-al', '/etc';
if ($? < 0) {
        print "Nepovedlo se spustit: $!\n";
} elsif ($? & 127) {
        print "Skončilo na signál ", $? & 127, "\n";
        print "Core dumped\n" if $? & 128;
} elsif ($?) {
        print "Skončilo s kódem ", $? >> 8, "\n";
}
```

Standardní knihovna: fork

```
my $pid = fork;
if ($pid < 0) { die "fork failed: $!\n"; }
if ($pid > 0) {
        # Parent
        waitpid $pid, 0;
        print "Exit status: $?\n";
} else {
        # Child
        exec '/bin/ls', '/etc';
        die "Exec failed: $!\n";
}
```

Test::Simple, Test::More

```
use Test::Simple tests => 3;
ok( 1 == "1", "Jedničky se rovnaji" );
ok( 1 != 2, "Vesmír je v pořádku" );
```

```
1..3
ok 1 - Jedničky se rovnaji
ok 2 - Vesmír je v pořádku
# Looks like you planned 3 tests but ran 2.
```

```
use Test::More;
is( cos 0, 1, "Kosinus nuly" );
isa_ok( IO::File->new, 'IO::File' );
done_testing;
```

tie

```
use NDBM_File;
use Fcntl;

tie my %h, 'NDBM_File', 'file.db', O_RDWR|O_CREAT, 0666 or die;
print $h{'brum'}++, "\n";
untie %h;
```

Přetěžování operátorů letem světem

```
package Really::Big::Int;

use overload
        '+' => \&plus,
        '-' => \&minus,
        '""' => \&stringify;

sub minus {
        my ($self, $other, $swapped) = @_;
        ...
}

package TwoFaced;
sub new { return bless [ $_[1], $_[2] ]; }
use overload
        '0+' => sub { $_[0]->[0] },
        '""' => sub { $_[0]->[1] },
        fallback => 1;
```

POD (Plain Old Documentation)

```
=pod

=head1
Potrava tužkožrouta obecného

=over

=item tužky
Nejraději má obyčejné, ale propiskou nepohrdne.

=item fixky
Z těch upíjí inkoust.

=back

=cut
```

pod2html
pod2man

POD – formátovací kódy

```
I<kurzíva>
B<tučně>
C<kód>
L<Net::Ping> ... link
E<lt> ... pojmenovaný znak
F</etc/passwd>
```

Cairo

```
use Cairo;

my $surf = Cairo::ImageSurface->create('argb32', 1024, 768);
my $cr = Cairo::Context->create($surf);

$cr->rectangle(100, 100, 300, 100);
$cr->set_source_rgb(0.7, 0, 0.7);
$cr->fill;

$cr->rectangle(100, 100, 300, 100);
$cr->set_source_rgb(0, 0, 0);
$cr->set_line_width(2);
$cr->stroke;

$cr->show_page;
$surf->write_to_png('out.png');
```

DBI

```
use DBI;

my $dbh =
  DBI->connect("DBI:mysql:database=karryina;host=localhost",
               "karry", "tajneheslo", {AutoCommit => 0})
  or die("Připojení selhalo s chybou $DBI::err: $DBI::errstr");

unless ($dbh->do("DELETE FROM users WHERE status = -1"))
{
  say "Něco je hodně špatně.";
}

$dbh->commit();

$dbh->disconnect();
```

DBI

```
my $dbh = DBI->connect($dsn, "karry", "heslo",
                       {RaiseError => 1, AutoCommit => 0});

my $sql = qq{SELECT nazev_c, datum_c FROM clanky_c
             WHERE id_r = ? AND uid = ?};
my $sth = $dbh->prepare($sql);

$sth->execute($rubrika, $autor);
$sth->bind_col(1, \$nazev);
$sth->bind_col(2, \$datum);

while ($sth->fetch)
{
  say "$nazev ($datum)";
}
```

DBI

```
my $bears = $dbh->selectall_arrayref("SELECT name, honey
                                      FROM bears");

foreach my $bear (@$bears)
{
  say "Bear $bear->{name} has $bear->{honey} honey.";
}

my %bears_hash = $sth->fetchall_hashref("name");
my %bears_hash = $dbh->selectall_hashref($sql, "name");

say $bears_hash->{$name}->{honey};
```

Scalar::Util

```
use Scalar::Utils qw /reftype refaddr weaken/;

$addr = refaddr 42;
$addr = refaddr \$prom;

say reftype 42;
say reftype \$prom;

my $prom;
my $ref = \$prom;
weaken($ref);
my $kopie = $ref;
$kopie obsahuje silnou referenci

unweaken($ref);
```

Scalar::Util

```
$jedna = dualvar(1, "jedna");
say 10 + $jedna;
say "Dvacet " . $jedna;
```

```
11
Dvacet jedna
```

```
say "Yep" if isdual $jedna;
```

```
say "Yep" if looks_like_number 42;
```

List::Util

```
$max = reduce { $a > $b ? $a : $b } @cisla;

fce() if any { length > 10 } @strings; # all, none, notall
$max = max @cisla; # min, maxstr, minstr
$fst = first { $_ =~ qr/a+b+c+/ } @strings;
```

List::MoreUtils

```
$fst = first_index { $_ =~ qr/a+b+c+/ } @strings;

apply { $_ *= 2 } @pole;

before { length > 8 } @strings;
after_incl { length > 8 } @strings;

@a = ( 1, 2, 3, 4 );
@b = ( a, b, c, d );
@h = pairwise { ($a, $b) } @a, @b;
```

```
1, a, 2, b, 3, c, 4, d
```

```
say uniq (1, 2, 3, 1, 4, 3, 2, 0);
say scalar uniq (1, 2, 3, 1, 4, 3, 2, 0);
```

```
1 2 3 4 0
5
```

common::sense

- use utf8;
- use strict qw(vars subs);
- use feature qw(say state switch);
- use feature qw(unicode_strings unicode_eval current_sub fc evalbytes);
- no feature qw(array_base);
- no warnings;
- use warnings qw(FATAL closed threads internal debugging pack portable prototype inplace io pipe unpac malloc deprecated glob digit printf layer reserved taint closure semicolon);
- no warnings qw(exec newline unopened);

Getopt::Long

```
my $sep = "aaa";
my $depth = 3;
my $long = 1, $recursive = 0, $magic = 0;
my @dirs, $verbose, $date;

Getopt::Long::Configure("bundling");

$res = GetOptions ("depth=i" => \$depth, "sep=s" => \$sep,
                   "dirs=s" => \@dirs, "verbose" => \$verbose,
                   "date:s" => \$date, "l|long" => \$long,
                   "r" => \$recursive, "m" => \$magic);
```

```
./program --sep " + " --depth 6 --verbose \
          --date --dir /tmp/ --dir /aux/karry/ \
          soubor.txt -lrm
```

B::Deparse

```
perl -MO=Deparse,-x3 -e 'for ($i=0; $i<8; $i++) { print; }'
while ($i < 8) {
    print $_;
}
continue {
    ++$i
}
```

```
perl -MO=Deparse,-q -e 'print "abc $xyz def"'
print 'abc ' . $xyz . ' def';
```

```
use B::Deparse;
my $dep = B::Deparse->new("-x3", "-q");
$dep->coderef2text(sub { $_[0] =~ s{brum}{abc}g or die;  });
die unless $_[0] =~ s/brum/abc/g;
```

Carp

```
use Carp;

# warn user (from perspective of caller)
carp "string trimmed to 80 chars";

# die of errors (from perspective of caller)
croak "We're outta here!";

# die of errors with stack backtrace
confess "not implemented";

# cluck, longmess and shortmess not exported by default
use Carp qw(cluck longmess shortmess);
cluck "This is how we got here!";
$long_message   = longmess( "message from cluck()
                             or confess()" );
$short_message  = shortmess( "message from carp()
                              or croak()" );
```

LWP: Triviální příklad

```
use HTTP::Request;
use LWP::UserAgent;

my $req = HTTP::Request->new(GET => 'http://www.ucw.cz/');

my $ua = LWP::UserAgent->new;
my $res = $ua->request($req);   # -> HTTP::Response

print $res->decoded_content;
```

LWP: Doplňujeme detaily

```
use HTTP::Request;
use LWP::UserAgent;

my $req = HTTP::Request->new(GET => 'http://www.ucw.cz/');
$req->header('Accept-Charset' => 'utf-8');

my $ua = LWP::UserAgent->new;
$ua->agent('UrsaMaior/1.0');
$ua->from('mj@ucw.cz');
my $res = $ua->request($req);

$res->is_success or die;
print $res->code, "\n";
print $res->header('Content-Type'), "\n";
print $res->decoded_content;
```

WWW::Mechanize – mechanický uživatel webu

```
my $m = WWW::Mechanize->new(
        agent => 'UrsaMaior/1.0',
        autocheck => 1
);
my $page = $m->get('http://www.ucw.cz/');
print $page->decoded_content;

my $next = $m->follow_link( text_regex => qr{LibUCW} );

use HTML::TreeBuilder;
my $body = $next->decoded_content;
my $t = HTML::TreeBuilder->new_from_content($body);
my $title = $t->find_by_tag_name('title');
print $title->as_text;
```

WWW::Mechanize – formuláře

```
$m->get('http://blind.jizdnirady.idnes.cz/ConnForm.asp?tt=z');
my $res = $m->submit_form(
        form_name => 'Spojeni',
        fields => {
                FromStn => 'Kosova Hora',
                ToStn => 'Olbramovice',
        },
        button => 'search',
);
print $res->decoded_content;
```

Hippo::HTML

```
use Hippo::HTML;    # Pozor, zatím není ve CPANu
my $t = Hippo::HTML->new(
  [ 'div',
    [ 'h1', 'Heading' ],
    [ 'p', { class => 'test' },
      'A paragraph of text with one word ',
      [ 'em', 'emphasized.' ],
    ],
    [ 'p', Hippo::HTML::Raw->new('&#9731;') ],
    [ 'table',
        [ 'tr', map { [ 'td', $_ ] } 1..10 ],
        [ 'tr', map { [ 'td', sqrt $_ ] } 1..10 ],
    ],
  ]
);
$t->render(\*STDOUT, 0);
```

English

```
use English;

say "$PROGRAM_NAME called with @ARG";

local $INPUT_RECORD_SEPARATOR = ' '; # I jako $RS

say "Svět je v pořádku" if ($$ == $PID and $$ == $PROCESS_ID);
```

Lingua::Romana::Perligata

```
use Lingua::Romana::Perligata;

maximum inquementum tum biguttam egresso scribe.
meo maximo vestibulo perlegamentum da.
da duo tum maximum conscribementa meis listis.

dum listis decapitamentum damentum nexto
    fac sic
        nextum tum novumversum scribe egresso.
        lista sic hoc recidementum nextum
            cis vannementa da listis.
    cis.
```

PSGI a Plack – úvod

```
my $app = sub {
        my ($env) = @_;
        return [
                '200',
                [ 'Content-Type' => 'text/plain' ],
                [ "Hello, world!\n" ],
        ];
};
```

```
REQUEST_METHOD          podle specifikace CGI
REQUEST_URI
QUERY_STRING
HTTP_...                hlavičky HTTP požadavku
psgi.version            verze protokolu
psgi.input ->read       tělo požadavku
psgi.error ->print      error log
```

PSGI – zpožděná odpověď

```
my $app = sub {
        my ($env) = @_;
        $env->{'psgi.streaming'} or die;
        return sub {
                my ($responder) = @_;
                ...
                $responder->([ 200, $headers, [$content] ]);
        };
};
```

PSGI – streamovaná odpověď

```
my $app = sub {
        my ($env) = @_;
        $env->{'psgi.streaming'} or die;
        return sub {
                my ($responder) = @_;
                ...
                my $writer = $responder->([ 200, $headers ]);
                ...
                $writer->write("Hello, world!\n");
                $writer->close;
        };
};
```

AnyEvent – I/O a časovače

```
use AnyEvent;

...

my $io = AnyEvent->io(
        fh => \*STDIN,
        poll => 'r',
        cb => sub { $name = <STDIN>; },
);

my $timer = AnyEvent->timer(
        after => 0.5,
        interval => 1,
        cb => sub { print "Tick tock\n"; },
);
```

AnyEvent – signály, procesy, podmínky

```
use AnyEvent;

my $sig = AnyEvent->signal(
        signal => 'INT',
        cb => \&shut_down,
);

my $child = AnyEvent->child(
        pid => $pid,
        cb => sub {
                my ($pid, $status) = @_;
                print "Process $pid exited, status=$status\n";
        },
);

my $cond = AnyEvent->condvar;
$cond->send;
$cond->recv;
```

AnyEvent – příklad

```
use AnyEvent;

my $done = AnyEvent->condvar;

my $name;
my $wait = AnyEvent->io(
        fh => \*STDIN,
        poll => 'r',
        cb => sub {
                $name = <STDIN>;
                $done->send;
        },
);

$done->recv;
undef $wait;
print "Received: $name\n";
```

AnyEvent::Handle – část 1

```
use AnyEvent;
use AnyEvent::Handle;

my $done = AnyEvent->condvar;
my $h = AnyEvent::Handle->new(
        connect => [ 'localhost', 4242 ],
        on_error => sub {
                my ($h, $fatal, $msg) = @_;
                print "ERROR: $msg\n";   # $!
                $h->destroy;
                $done->send;
        },
        on_eof => sub {
                my ($h) = @_;
                $h->destroy;
                $done->send;
        },
```

AnyEvent::Handle – část 2

```
        on_read => sub {
                my ($h) = @_;
                $h->push_read( line => sub {
                        my ($h, $line) = @_;
                        $h->push_write("$line?\n");
                } );
        },
        timeout => 10,
);

$h->push_write("Hello, world!\n");
$done->recv;
```

DBIx::Class – mapování databáze

```
use DBIx::Class::Schema::Loader qw(make_schema_at);

make_schema_at(
        'Bear::DB',
        {
                debug => 1,
                dump_directory => './lib',
                components => [ 'InflateColumn::DateTime ],
        },
        [
                'dbi:Pg:dbname="foo"',
                'USER',
                'PASSWORD',
        ],
);
```

```
=> Bear::DB (schéma)
=> Bear::DB::Beehives (tabulka)
```

DBIx::Class – dotazy

```
use Bear::DB;

my $schema = Bear::DB->connect(
        'dbi:Pg;dbname="foo"',
        'USER',
        'PASSWORD',
);

my $hives = $schema->resultset('Beehives');
my $h = $hives->find(10);
if ($h->name eq "") {
        $h->name('Unnamed');
        $h->update;
}

my @h = $hives->search({ visibility => 1 });
my $c = $hives->count({ visibility => 1 });
my @w = $hives->search_literal('length(name) <= ?', 10);
```

Try::Tiny

```
my $stav;
try
{
  $stav = moje_nebezpecna_funkce();
} catch
  {
    $stav = -1;
  }
  finally
  {
    say "Máme to za sebou";
  }
```

Email::Simple – posílání e-mailu

```
use Email::Simple;

my $email = Email::Simple->create(
    header => [
      From => 'kostlivec@skrin',
      To => 'karry+hriste@karryanna.cz',
      Subject => 'Překvápko',
    ],
    body => 'Velmi jednoduchý mail',
  );

$email->header_set('X-Coin: $1');

print $email->as_string;
```

Email::MIME – posílání e-mailu

```
use Email::MIME;
use Email::Sender::Simple qw(sendmail);

my $email = Email::MIME->create(
  header_str => [
    From => 'letadylko@obluda',
    To => 'root@obluda',
    Subject => 'tcejbuS'
  ],
  attributes => {
    encoding => 'quoted-printable',
    charset => 'utf-8',
  },
  body_str => 'You should have deleted this before reading';
);

sendmail($email);
```

Email::Simple – parsování e-mailu

```
use Email::Simple;

my $email = Email::Simple->new($text);

my $from_header = $email->header("From");
my @received = $email->header("Received");

my $body = $email->body;
```

Email::MIME – parsování e-mailu

```
use Email::MIME;

my $email = Email::MIME->new($doruceny_mail);

my $subject = $email->header('Subject');
my $from = $email->header('From');
```

Digest

```
use Digest::MD5 qw(md5);
$digest = md5($message);
$digest_hex = Digest::MD5::md5_hex($message);
$digest_base64 = Digest::MD5::md5_base64($message);

$ctx = Digest->new("SHA1");
$ctx->add("brum brum");
say $ctx->hexdigest;
```

POSIX

```
use POSIX;

$dir = POSIX::opendir("/var");
@files = POSIX::readdir($dir);
POSIX::closedir($dir);

$errno = POSIX::errno(); # = $!

($sys, $node, $release, $version, $machine) = POSIX::uname();
($realtime, $user, $system, $cuser, $csystem) = POSIX::times();
```

Storable

```
use Storable qw(store retrieve);

my %pocty_dni = { leden => 31, unor => 28, brezen => 31 };

story(\%pocty_dni, 'kalendar')
  or die "Nedaří se mi uložit počty dní";

my $ref = retrieve('kalendar');

say $ref->{leden};
```

```
31
```