Cgi-bin

Ukažme si, jak můžeme perl použít pro psaní jednoduchých cgi-bin skriptů. Vytvoříme soubor ab.cgi:

```
#!/usr/bin/perl -w
print "Content type: text/plain\n\n";
print "Simple script.\n";
```

Musíme mu dát práva pro provádění (chmod +x ab.cgi), cesta na prvním řádku musí správně ukazovat na interpret perlu na WWW serveru, na kterém chceme skript provozovat. Na serveru musí být spouštění cgi-bin skriptů povoleno a skript musí mít buď korektní příponu (.cgi), nebo být uložen ve vyhrazeném adresáři. Tyto lokální detaily je vhodné konzultovat se správcem systému.

Nyní ověříme, zda jsme schopni ab.cgi spustit z příkazového řádku a zda dává (alespoň na první pohled) korektní výsledky. Server očekává data od skriptu na jeho standardním výstupu. Na začátku je hlavička či hlavičky, od zbytku jsou odděleny prázdným řádkem (proto \n\n). Hlavička Content type říká, jaká data na výstup posíláme.

Příklad, generující HTML výstup, je:

```
#!/usr/bin/perl -w
print "Content type: text/html\n\n";
print <<EOF;
<HTML><HEAD><TITLE>
Script ab.cgi
</TITLE></HEAD>
<BODY>
This is HTML body generated by script
</BODY></HTML>
EOF
```

Zápis <<EOF; znamená to samé jako v shellu, text vložený přímo do těla skriptu.

Jako jiný příklad vezměme mezi uživateli oblíbené počítadlo.

```
#!/usr/bin/perl
$LOCK_EX = 2;
$LOCK_UN = 8;
$counter_file = "/home/user/public_html/counter_file";
print "Content-type: text/plain", "\n\n";
open (FILE, "+< $counter_file") ||
                        die "Error opening counter file.\n";
flock (FILE, $LOCK_EX);
$count = <FILE>;
seek (FILE, 0, SEEK_SET);
print FILE ++$count;
close (FILE);
print "You are visitor number $count\n";
```

Na začátku nastavíme hodnoty několika užitečných proměnných, poté vytiskneme HTTP hlavičku. Potom ze souboru $counter_file přečteme starý počet přístupů ($count = <FILE>;). Soubor otevíráme v módu +<, což znamená čtení i zápis. Při práci se souborem

ho funkcí flock zamykáme, vyhrazujeme si k němu exkluzivní přístup ($LOCK_EX). Poté zapíšeme novou hodnotu $count, již zvýšenou o jedna. Nakonec vytiskneme sdělení.

Zde je nutno zdůraznit jednu věc. Http server a tedy i uživatelské skripty jsou obvykle spouštěny pod speciálním uživatelem, například www. Tento uživatel musí mít práva pro zápis do souboru počítadla (obecně musí mít právo operaci provést). Na většině systémů běžný uživatel nemůže udělat chown na jiného a tím změnit vlastnictví souboru na www, proto je nutno přidělit práva pro zápis světu nebo použít suid přístup.

V další části se podíváme, jak můžeme pro tvorbu složitějších cgi-bin skriptů s výhodou využít již hotových modulů CGI*, které už vytvořil někdo jiný. Předtím se ale seznámíme s principy a použitím modulů. Vážnějším zájemcům o tvorbu modulů doporučuji manuálové stránky perlmod, perlobj či perlbot, zde se omezíme hlavně na čistě uživatelský přístup.

Package, moduly, reference

Perl má jednoduchý způsob, jak zabránit tomu, aby si různé části rozsáhlého programu navzájem přepisovaly proměnné. Význam takového přístupu se ještě zvýší, pokud začneme používat části kódu, které napsal někdo jiný. Příkazem package definujeme a pojmenováváme prostor jmen. Na proměnnou, která přísluší do určitého package, se odkazujeme pomocí dvojité dvojtečky, například $Sport::swim je skalární proměnná $swim v package Sport. Pokud je název package před :: prázdný, uvažuje se defaultně package main. Jméno pole @::fields je to samé jako @main::fields. Pracujeme-li v daném modulu a neuvádíme-li před jmény proměnných název package, pracujeme se jmény z aktuálního package.

Pokud definujeme package v samostatném souboru, který se jmenuje stejně a má příponu .pm, říkáme mu modul. Proměnné a funkce uvedené v daném modulu spadají do jednoho prostoru jmen. Modul může některé funkce exportovat tím, že je uvede v poli @EXPORT (ty se exportují defaultně), případně @EXPORT_OK (tyto na požádání).

Modul použijeme příkazem use. Defaultně si tím zpřístupníme všechny funkce, které modul exportuje. Importované funkce můžeme také explicitně zpřístupnit v seznamu za příkazem.

```
use Curses;
use Curses qw(initscr endwin);
```

První příkaz zpřístupní všechny funkce, které jsou v modulu Curses, druhý pouze funkce initscr a endwin.

Pro implementaci objektů se využívá fakt, že do skalární proměnné můžeme uložit kromě již zmíněných numerických a textových hodnot také referenci. Reference je v principu totéž, co ukazatel (pointer) v jazyce C. Je to adresa do paměti, spolu s typem, na který ukazuje. Adresu proměnné či funkce dostaneme pomocí backslashe. Například:

```
$i = 10;
$j = \$i;
print "$$j\n";
$$j += 16;
print "$$j\n";
```

vytiskne pod sebe 10 a 26. První řádek je obyčejné přiřazení do proměnné. Na druhém řádku uložíme do $j adresu proměnné $i. Ve třetím řádku tiskneme hodnotu, na kterou ukazuje `$j`. Znak dolar navíc zde znamená jednu úroveň dereference. Pokud bychom vytiskli pouze `$j`, byla by odpověď ve stylu `SCALAR(0x1001af8c)`. Poslední řádky ukazují, že s dereferencovanými proměnnými pracujeme zcela transparentně.

Pomocí referencí můžeme v perlu vytvářet složité datové struktury, jako pole polí, pole hashů, atd. Vždy to je vlastně pole či hash referencí na daný typ, kterým může být opět třeba pole. Reference si pamatují, kolikrát jsou data odkazována a pokud počet odkazů klesne na nulu, automaticky se uvolní alokovaná paměť. Tedy podobně jako v případě normálních polí a hashů se nestaráme o alokování dalších prvků, pokud se pole natahuje, a uvolňování paměti, pokud se zkracuje, i zde to perl dělá za nás.

Poznamenejme jen, že pokud chceme uložit adresu funkce či volat funkci dereferencí, je třeba vyznačit pomocí `&`, že jde o funkci. Při normálním volání funkce jejím jménem můžeme `&` před jméno funkce uvést také, ale není to nutné.

```
sub func1 {
        print "Func1 called\n";
        }
sub func2 {
        print "Func2 called\n";
        }
$ref = \&func1;
&$ref();
$ref = \&func2;
&$ref();
```

odpoví

```
Func1 called
Func2 called
```

Do proměnné `$ref` přiřazujeme adresu funkcí získanou operátorem \ a pak je dereferencí voláme. Vlastní hodnota `$ref` bude `CODE(0x1001b008)` či podobně.

Je možné definovat i anonymní funkce, které pak voláme pouze dereferencí:

```
sub func1 {
        print "Func1 called\n";
        }
$ref = \&func1;
&$ref();
$ref = sub { print "Anonymous function\n"; };
&$ref();
```

V první části voláme `func1`, ve druhé přiřazujeme do `$ref` přímo adresu anonymní funkce, se kterou potom pracujeme analogicky.

Třídou je v perlu takový modul, jehož funkce korektně pracují jako metody. Jde o to, že funkce musí podporovat mechanismus, kterým jsou při volání přiřazeny k určité instanci dat. Reference na datovou strukturu je jim předávána jako první parametr a je na metodách, aby ho správně zpracovaly. Ukažme si, jak by mohl vypadat konstruktor třídy. Konstruktory nemají povinné jméno, ale z historických důvodů se doporučuje používat `new`.

```
package Widget;
```



```
sub new
        {
        my $arg = shift;
        my $class = (ref $arg) || $arg;
        my $self = { };
        bless $self, $class;
        $self->init( @_ );
        $self;
        }
```

Na prvním řádku je jméno package. Pak definujeme funkci `new`. Její první argument si `shift`em uložíme. Do proměnné `$class` pak přiřadíme třídu, pro kterou je tato metoda volána. Na dalším řádku alokujeme anonymní hash a odkaz na něj přiřazujeme do proměnné `$self`. Tento hash tvoří datovou oblast instance, mohli bychom místo něho pochopitelně použít i normální pole či skalár. Funkcí `bless` svazujeme referenci `$self` s packagem `$class`, tím vytváříme příslušnost dat ke třídě. Pak zavoláme nějakou naši inicializační metodu a nakonec vrátíme referenci na nový objekt.

Ukažme si nyní, jak by vypadalo typické použití tohoto konstruktoru:

```
my $w = new Widget();
$w->resize(20, 5);
```

Na prvním řádku voláme metodu `new` a dáváme jí jako parametr jméno třídy. Volání by mohlo mít také tvar `$w = Widget->new()`. Na druhém řádku voláme metodu `resize` a reference `$w` je jí opět předána jako první parametr. V těle konstruktoru `new` testujeme, zda první parametr je reference, pokud ne, předpokládáme, že je to jméno třídy. Takovýto přístup nám dovoluje psát

```
$w = Widget->new();
$x = $w->new();
```

kdy na druhém řádku vytvoříme novou instanci stejné třídy jako je `$w`, a jakou třídu máme na mysli zjistíme právě pomocí `ref`. Jméno třídy `Widget` bychom do konstruktoru mohli napsat i přímo (`bless $self, Widget`), takhle ale umožňujeme použití této metody jako virtuálního konstruktoru i pro třídy, které z `Widget` budou dědit. Dědičnost ve třídě vyznačujeme pomocí pole `@ISA`, ve kterém uvedeme jména (rodičovských) tříd, ve kterých se mají hledat metody, pokud nejsou nalezeny v dané třídě.

Na závěr tohoto krátkého úvodu do tvorby modulů a objektů v perlu si dovolím ocitovat část manuálové stránky:

> Always use `-w` . Try to `use strict;` (or `use strict qw(...);`). Remember that you can add `no strict qw(...);` to individual blocks of code which need less strictness. Always use `-w`. Always use `-w`!

V dalším se již zaměříme na použití hotových modulů, které můžeme najít na CPANu.

Cgi-bin pomocí modulů

Moduly `CGI`, `CGI_Lite` a `CGI::*` jsou navrženy pro usnadnění tvorby cgi-bin skriptů. Jejich cíle jsou stejné, dát programátorovi jednoduché nástroje, pomocí nichž by mohl co nejsnáze udělat, co potřebuje.

Pokud použijeme modul `CGI`, pak příkaz `$q = new CGI;` projde vstup skriptu a vytvoří objekt, který bude obsahovat tato vstupní data. Voláním `$name = $q->param('name');` přečteme hodnotu argumentu `name`. Pokud má mít parametr více hodnot, můžeme je přečíst přiřazením do pole. Takto zpracujeme vstup.

Pro generování výstupního dokumentu, který bude vrácen prohlížeči, je k dispozici celá řada metod. Pokud je v proměnné `$q` objekt vytvořený voláním `new CGI`, potom například `$q->header;` vrátí jako řetězec HTTP hlavičku, kterou potom příkazem `print` vytiskneme. Metodě `header` můžeme dát i argumenty, kterými řekneme, jakého typu dokument je, například `image/jpeg`, případně další volby. Obdobně můžeme použít s instancí `$q` i `start_html` či `end_html`. Všechny metody vracejí řetězec, nezapisují přímo na výstup. O to se musíme postarat sami, nejčastěji pomocí `print`. Má to tu výhodu, že s textem můžeme (pokud pro to máme důvod) před tiskem dále manipulovat.

Pro vložení formulářů, vstupních editovatelných prvků, menu, tlačítek a dalších prvků dokumentu použijeme volání odpovídajících funkcí. Jejich seznam zde nebudeme vypisovat, je možno ho najít v manuálových stránkách. Uvedeme zde pouze příklad programu (nedělá nic skutečně užitečného), ve kterém některé předvedeme.

```
#!/usr/bin/perl -w
use CGI;
$q = new CGI;
print $q->header, $q->start_html("Form");
print $q->startform, "Your name", "<BR>",
                                $q->textfield('name'), "<P>";
print $q->radio_group('-name' => 'siblings',
        '-values' => ['none', "brother", "sister", "more of any" ],
        '-default' => "none");
print $q->submit('Action', 'Yes');
print "<P> Variables: <P>", $q->dump;
print $q->endform;
print $q->end_html;
```

Názvy metod jsou zřejmé, proto se zastavíme jen u pár věcí, týkajících se perlu. První je hash, použitý jaké argument při volání metody `radio_group`. Před šipkou (`=>`) je vždy uveden klíč, za ní pak hodnota. Klíče i hodnoty jsme uzavřeli do apostrofů či uvozovek, aby perl neměl problémy s odlišením těchto řetězců od klíčových slov. Položky asociativního pole můžeme také zapsat jako normální seznam. Liché prvky jsou pak klíče a sudé hodnoty. Tento zápis pomocí šipek je ale přece jenom přehlednější.

Druhým jevem je anonymní pole u argumentu `-values`, vytvořené pomocí hranatých závorek. Připomeňme, že anonymní hash vytvoříme pomocí závorek složených.

Modul `CGI_Lite` pracuje obdobně, stejně jako moduly `CGI::*`. Vždy je vytvořen objekt, který obsahuje základní stavovou informaci, a na základě této je pak zpětně generován výstupní dokument. Použití modulů je pro cgi-bin skripty výhodné z několika důvodů: tím, že moduly fungují ve svém vlastním prostoru jmen, se snižuje šance, že vytvoříme kolidující názvy proměnných. Moduly jsou také psány tak, aby implementovaly co nejpřesněji normy i reálný svět, a proto mají ošetřeny i situace, o kterých by nás při programování vůbec nenapadlo přemýšlet. A dalším důvodem je, že zvolený přístup (hodnoty parametrů jsou uloženy mezi daty objektu CGI) dovoluje snadno vytvářet dokumenty, které budou již mít formuláře přednaplněny hodnotami z posledního volání.

LWP: Library modules for WWW access in Perl

Tyto moduly implementují užitečné funkce pro práci s WWW. Jsou zde funkce, které je možno použít pro přímé stáhnutí HTML stránky ze vzdáleného serveru, třída pro zpracování HTML dokumentů, procházení URL, implementace nejrůznějších protokolů.

Například toto je způsob, jak vytisknout dokument, uložený kdekoli na síti:

```
use LWP::Simple;
print get "http://www/";
```

Pokud bychom chtěli vytisknout pouze čistý text bez HTML značek, můžeme napsat třeba:

```
use LWP::Simple;
use HTML::Parse;
print parse_html(get "http://www/")->format;
```

Z manuálové stránky je příklad, jak stáhnout nejnovější verzi perlu

```
$ perl -MLWP::Simple -e '
getstore "ftp://sunsite.mff.cuni.cz/Languages/Perl/CPAN/src/latest.tar.gz",
        "perl.tar.gz"; '
```

Zmíníme se i o složitějším modulu `LWP::UserAgent`. Objektově orientovaným způsobem si zadefinujeme vlastnosti našeho WWW klienta (jméno, protokoly, timeouty, proxy), a potom mu předáváme požadavky pomocí třídy `HTTP::Request`. Například:

```
use LWP::UserAgent;
$client = new LWP::UserAgent();
$client->agent("Microsoft Exploder 0.99");
$request = new HTTP::Request('GET', 'http://www.fi.muni.cz/');
$request->header("Accept" => "text/html");
$answer = $client->request($request);
print $answer->content if $answer->is_success();
```

Můžeme samozřejmě použít i metodu POST, předávat hesla pro přístup k dokumentům s autorizací, zpracovávat přijímaná data námi definovanou funkcí nebo procesem, již při stahování, atp. Doporučuji si projít `man lwpcook`, LWP cookbook.

Z dalších zajímavých modulů pro práci na Webu zmiňme například `MIME::Base64` nebo `MIME::Quotedprint`, které implementují MIME standard, `HTTP::*`, díky kterým získáme jednotný přístup k HTTP požadavkům, a `HTML::*`, které umožňují analyzovat HTML strukturu dokumentu nebo ho formátovat například do Postscriptu či jako čistý text.

perl/Tk

Tk znamená Toolkit a je to prostředek pro tvorbu X-ových aplikací. Všeobecně je známější ze spojení tcl/Tk, jeho funkce jsou ale díky modulu Tk přístupné i z perlu. Můžeme tedy vytvářet grafické X-aplikace přímo, bez nějakého dalšího nástroje. Mnoho informací najdeme v manuálových stránkách, často kladené otázky týkající se perlu/Tk jsou zodpovězeny na adrese `http://w4.lns.cornell.edu/~pvhp/ptk/ptkTOC.html`, dokumentace widgetů a funkcí na `/~pvhp/ptk/doc/`.

Základním pojmem je widget, čili přípravek. Obvykle se projevuje jako obdélníková část obrazovky, která má v sobě nápis, barvy, rámeček, chová se jako tlačítko, menu, a podobně. V perlu/Tk jsou k dispozici tyto primitivní Tk widgety:

> Button, Canvas, Checkbutton, Entry, Frame, Label, Listbox, Menu, Menubutton, Message, Radiobutton, Scale, Scrollbar, Text, Toplevel.

Aplikace, používající Tk widgety, je skládá do stromu, ve kterém pak existují vztahy rodič/potomek. Pomocí objektových přístupů perlu můžeme vytvářet i kompozitní widgety (například dialogové okno pro výběr souboru může být z hlediska použití jediný objekt), měnit defaultní chování nebo je rozšiřovat o další metody.

Ukažme si typické Hello world:

```
#!/usr/bin/perl -w
use strict;
use Tk;
my $main = new MainWindow;
$main->Label(-text => "Hello, world")->pack();
$main->Button(-text => "End", -command => \&exit)->pack();
MainLoop;
```

Do proměnné `$main` přiřadíme nově vytvořený objekt hlavního okna aplikace (`MainWindow`). Potom voláme funkce `Label` a `Button`, s uvedením `$main` jako rodiče. Chceme, aby hlavní okno obsahovalo jak nápis, tak tlačítko. Při tvorbě nápisu a tlačítka předáváme argumenty, kterými říkáme, jak budou vypadat. Specifikujeme vypisovaný text a u tlačítka také akci, která se má provést, pokud ho stiskneme (klikneme na něm) myší. Poté takto vytvořené objekty zabalíme pomocí `pack`. Tím je předáme do správy takzvanému geometry manageru, který se stará o to, aby byly objekty pěkně poskládané. Nakonec voláme funkci `MainLoop`, která provede vlastní zobrazení oken na obrazovace a potom začne zpracovávat události a bude je posílat správným přípravkům v aplikaci.

To, že jsme uvedli parametr specifikující text pouze změnilo defaultní nastavení, které je žádný text. Většina parametrů má nejakou standardní hodnotu. Tak například geometry manager packer skládá widgety pod sebe, od shora dolů. Pokud bychom ale při volání `pack` na nápis přidali parametr `-side => 'left'`, umístil by se nápis k levému okraji a tlačítko napravo od něho.

Argument `-command` u tlačítka má jako hodnotu adresu funkce `exit`. Mohli jsme specifikovat i jakoukoli naši uživatelskou funkci, nebo napsat anonymní funkci:

```
$main->Button(-text => "End",
        -command => sub {print "Button pressed\n"; exit })->pack();
```

Všechny widgety mají společnou nějakou množinu standardních voleb, jednotlivé objekty si navíc mohou definovat další. Standardní jsou:

activebackground activeborderwidth activeforeground anchor background bitmap borderwidth cursor disabledforeground exportselection font foreground geometry highlightbackground highlightcolor highlightthickness image insertbackground insertborderwidth insertofftime insertontime insertwidth jump justify orient padx pady relief repeatdelay repeatinterval selectbackground selectborderwidth selectforeground setgrid takefocus text textvariable troughcolor underline wraplength xscrollcommand yscrollcommand.

Existuje množina standardních Tk metod:

after appname bind bindtags configure destroy exit option options setPalette update winfo wm.

Například metodou `bind` můžeme widgetu přiřadit akci, která se provede při výskytu definované události či posloupnosti událostí. Typy událostí mohou být:

ButtonPress (Button) Expose Map ButtonRelease FocusIn Motion Circulate FocusOut Property Colormap Gravity Reparent Configure KeyPress (Key) Unmap Destroy KeyRelease Visibility Enter Leave.

Jednotlivé widgety mohou mít i své vlastní metody. Například widget `Text`, který vytvoří okno pro editování má mimo jiné metody `insert` pro vložení dalšího textu, `mark*` pro práci se značkami v textu, `xview` a `yview` pro komunikaci se scrollbary, atd.

Místo geometry manageru packer můžeme použít i volání `grid`, který skládá přípravky do rastru a vytvoříme s ním například tabulku. Máme přístup i k dalším Tk třídám, konkrétně pro obrázky jsou to bitmap, image a photo. Obrázek zeměkoule dostaneme programem:

```
#!/usr/bin/perl -w
use strict;
use Tk;
my $main = new MainWindow;
my $c = $main->Canvas()->pack();
my $img = $c->Photo('IMG',
        -file => Tk->findINC("demos/images/earth.gif"));
$c->create("image", 10, 10, -anchor =>  "nw", -image => $img);
MainLoop;
```

Pro další náměty doporučuji přečíst dokumentaci.

S perlem/Tk se standardně nainstaluje i demonstrační aplikace `widget`, která možnosti widgetů ukazuje. Ke každému typu či postupu je ukázán příklad, včetně zdrojového textu, který můžeme ihned editovat a znovu spouštět, takže ihed vidíme, jak se provedené změny projeví. Jiná demonstrační aplikace je `tkweb`, což je jednoduchý WWW prohlížeč. Jeho krása spočívá především v tom, že se můžeme ve zdrojových textech (celý `tkweb` má asi 3 kB) podívat, jak se co dělá v reálném světě.

Standardní moduly

Toto jsou moduly, které jsou ve standardní distribuci, tedy k dispozici všude, kde je korektně nainstalován perl.

AnyDBM_File
: Přístup k násobným DBM.

AutoLoader
: Nahrává funkce na požádání.

AutoSplit
: Rozděluje package pro použití s autoloadingem.

Benchmark
: Benchmark pro dobu běhu programu.

Carp
: Chybová hlášení se zohledněním volající funkce.

Config
: Lokální konfigurace perlu.

Cwd
: Název aktuálního adresáře.

DB_File
: Interface k Berkeley DB.

Devel::SelfStubber
: Vytváření částí pro selfloading.

DynaLoader
: Dynamické načítání C knihoven do perl kódu.

English
: Umožňuje používat dlouhé anglické názvy pro předdefinované proměnné.

Env
: Importuje proměnné prostředí.

Exporter
: Modul pro řízení importu a exportu jmen.

ExtUtils::Liblist
: Zjišťuje, jaké knihovny použít a jak.

ExtUtils::MakeMaker
: Vytváří Makefile pro instalaci dalších modulů.

ExtUtils::Manifest
: Nástroje pro práci se souborem MANIFEST.

ExtUtils::Mkbootstrap
: Vytváří zaváděcí soubor pro DynaLoader.

ExtUtils::Miniperl
: Odlehčená verze.

Fcntl
: Používá deklarace z `Fcntl.h`.

File::Basename
: Pro zpracování jmen souborů.

File::CheckTree
: Pro testy nad adresářovým stromem.

File::Find
: Procházení adresářového stromu.

FileHandle
: Objektový přístup k filehandlům.

File::Path
: Vytváření a rušení adresářů.

Getopt::Long
: Zpracování rozšířených argumentů příkazové řádky.

Getopt::Std
: Zpracování jednoduchých (jednoznakových) argumentů.

I18N::Collate
: Porovnávání podle locales.

IPC::Open2
: Otevření procesu pro čtení i zápis.

IPC::Open3
: Otevření procesu pro čtení, zápis a chyby.

Net::Ping
: Ping.

POSIX
: Zpřístupňuje IEEE Std 1003.1.

SelfLoader
: Nahrávání funkcí na požádání.

Safe
: Rozšíření pro bezpečné programy.

Socket
: Zpřístupňuje deklarace a struktury ze `socket.h`.

Test::Harness
: Standardní perlovské testy se statistikou.

Text::Abbrev
: Vytváří tabulky zkratek ze seznamu.

Pragma moduly

Speciálními případy modulů jsou pragma moduly. Pomocí nich můžeme ovlivnit překlad našeho programu. Mají účinek v rámci bloku a můžeme je i vypnout příkazem `no`.

Zde je seznam takovýchto modulů:

diagnostics
: Tímto požadujeme více diagnostických výstupů.

integer
: Tato část kódu se bude překládat v celočíselné aritmetice. Default je double.

less
: Říkáme překladači, že něčeho chceme méně. V současné verzi není implementováno.

overload
: Aktivujeme přetěžování operátorů.

sigtrap
: Umožňujeme zpětný průchod zásobníkem při nenadálých signálech.

strict
: Zpřísňujeme požadavky na čistotu programátorských obratů.

subs
: Předeklarováváme jména funkcí.

Kategorie modulů

Zde je seznam kategorií, do nichž jsou moduly na CPANu zařazovány. Seznam za názvem jsou podadresáře, které pro danou kategorii existují. Moduly v adresářích se mohou jmenovat jinak, seznam zde uvádím spíše jako přehled toho, co všechno už udělal někdo jiný.

02 Základní perlovské moduly, nástroje pro tvorbu rozšíření a dokumentace
: Alias, AutoLoader, B, Carp, Config, DynaLoader, English, Exporter, Filter, O, Opcode, Pod, Safe, SelfLoader, Symbol, UNIVERSAL, diagnostics, integer, less, lib, overload, sigtrap, strict, subs, vars

03 Vývojové prostředky
: AutoSplit, Benchmark, DataFlow, Devel, ExtUtils, FindBin, Include, Make, Test, Usage

04 Interface k operačním systémům
: B, BSD, Env, Fcntl, Ioctl, OS2, POSIX, Proc, Quota, SGI, Shell, Sys

05 Sítě, zařízení (modemy) a komunikace mezi procesy
: DCE, IPC, Net, Parallel, Ptty, SNMP, Socket

06 Datové typy a nástroje pro práci s nimi
: Array, Class, Data, Date, FreezeThaw, Math, Object, PDL, Ref, Set, Sort, Statistics, Storable, Tie, Time

07 Databázová rozhraní
: AnyDBM_File, DBD, DBI, DBZ_File, DB_File, MLDBM, Msql, Oraperl, Pg, Sybase,

SDBM_File, Xbase

08 Znaková a grafická uživatelská rozhraní
: Cdk, Curses, PV, Sx, Term, Tk

09 Práce s jinými jazyky a jejich emulace
: Language, Tcl

10 Jména souboru, souborové systémy, zamykání souborů
: Cwd, File

11 Práce s řetězci, zpracování textu, prohledávání
: Font, SGMLS, Search, String, Text

12 Volby, argumenty, parametry, konfigurační soubory
: ConfigReader, Getopt, IniConf, Resources

14 Autentikace, bezpečnost, kryptování
: Crypt, DES, Des, MD5, PGP, SHA

15 World Wide Web, HTML, HTTP, CGI, MIME
: Apache, C, CGI, CGI_Lite, FCGI, HTML, HTTP, HTTPD, LWP, MIME, URI, WWW

16 Práce se servery a daemony
: EventServer, Server

18 Obrázky, pixmapy, bitmapy, grafy
: GD, Image, OpenGL, PDL, PGPLOT, VRML

19 Mail, net news
: Mail, NNML, News

20 Řízení toku (callback, exception)
: AtExit, Callback, Religion

21 Filehandly, adresáře a práce se vstupy a výstupy
: DirHandle, FileCache, FileHandle, IO, Log, SelectSaver

23 Nezatříděné moduly
: ARS, Agent, Archie, Bio, Business, CPAN, Gann, Logfile, Neural, Penguin, Remedy, Roman, SyslogScan, Wais