PB173 Perl

05 Vstup a výstup

Roman Lacko xlacko1@fi.muni.cz

2023-03-12

Obsah

Thanatológia

Thanatológia

bIlujDI' yI‹ chegh› Qo'; yI‹ Hegh() ›!
It is better to ‹ die() › than to ‹ return › in failure.
— Klingon programming proverb in perldoc autodie

Thanatológia: ‹exit›

V niektorých prípadoch nestačí vrátiť ‹undef› ako indikáciu chyby
a je nutné ukončiť beh celého skriptu.

```
exit [EXPRESSION]
```

Ukončí skript s uvedeným chybovým kódom.
Bez argumentu znamená ‹exit 0›.

POSIX sémantika **0** pre úspešný a **1** pre neúspešný beh.

Nepoužívajte iné hodnoty než 0 - 255!

Zábavný historizmus
‹CORE::dump()›

Thanatológia: ‹die›

```
die [LIST]
```

Ekvivalent ‹throw› v iných jazykoch.

Ak má ‹LIST› viac než jeden prvok, spojí ich do jedného reťazca.
Prázdny zoznam sa nahradí ‹"Died"›.

Perl nemá ekvivalent štandardnej triedy pre výnimky,
použitie reťazcov je najjednoduchšie.

Ak text výnimky **ne**končí ‹\n›, Perl pridá aj informáciu, kde k úmrtiu došlo.

Thantológia: ‹warn›

```
warn [LIST]
```

Vypíše varovanie s textom v argumentoch.
Ak je ‹LIST› prázdny, použije ‹$@ // "Something went wrong"›.

> **!** Toto neslúži ako náhrada za logovanie.

Thanatológia: Návod na zomieranie

Kedy použijeme ktorý spôsob?

- `exit` Chyba na najvyššej úrovni skriptu, napr. nesprávny počet argumentov. Nedá sa zastaviť bez temnej mágie.
- `die` Chyba v knižnici alebo module. Dá šancu volajúcemu kódu reagovať.
- `warn` Keď niečo vyzerá podozrivo, ale v princípe môžeme pokračovať.

Môžeme chyby nejak ošetriť?

Thanatológia: ‹eval›

```
eval [EXPRESSION]
eval BLOCK
```

Vykoná Perl kód a zachytí prípadné chyby (napr. ‹die›) v premennej ‹$@›.

```
eval "say no more!";            # String eval
eval { do_a_backflip() };       # Block eval

if ($@) {
    say STDERR "Whoopsie: $@";
}
```

Pozor na ‹eval STRING›

Použitie ‹eval› na reťazci z neovereného zdroja môže mať nepriaznivé účinky na vaše mentálne zdravie a bezpečnosť programu.

Thanatológia: ‹%SIG›

Hash, ktorý obsahuje reakcie na signály.

Špeciálne kľúče ‹__DIE__› a ‹__WARN__› pre ‹die› resp. ‹warn›.

```
$SIG{__DIE__} = sub ($ex) {
    say "My $ex is going to kill me";
};
```

Výnimku je možné upraviť alebo obaliť, treba však znovu zavolať ‹die›.

Spracovanie výnimky v ‹$SIG{__DIE__}› je možné zrušiť len pomocou ‹goto›.

Thanatológia: ‹try›

Kľúčové slovo ‹try›
Táto vlastnosť je experimentálna od ‹v5.36›, môžete ju však používať.

```
use experimental qw{try};

try {
    # Dangerous zone
} catch ($ex) {
    # Handle the exception
} finally {         # Optional
    # Runs no matter what
}
```

V kombinácii s ‹do› vráti poslednú hodnotu.

Thanatológia: Modul ‹Carp›

```
use Carp;

croak "The argument is undefined, you twat";
carp "There is something fishy";
```

- Ekvivalenty ‹die› a ‹warn›, ktoré ohlásia výnimku z pohľadu volajúceho.

```
confess "I can't go on";
Carp::cluck "What is this?"      # Not exported by default
```

- Ako ‹croak› a ‹carp›, ale so *stack trace*.

Thanatológia: Modul ‹Try::Tiny›

Náhrada za kľúčové slovo ‹try› pred ‹v5.36›.

```
use Try::Tiny;

try {
    # Quelque chose de dangereux
} catch {                               # Optional
    # Exception en <$_>
};                                      # Note the <;>!
```

Tento modul nie je v jadre Perlu.

Control flow

Kľúčové slovo ‹try› je transparentné pre ‹return› a pod.;
‹Try::Tiny› simuluje ‹try› pomocou funkcií, takže ‹return› zachytí!

Thanatológia: Pragma ‹autodie›

Spôsobí, že niektoré ‹CORE::› funkcie namiesto benevolentnej chyby zavolajú ‹die›.

```
chdir "/mnt/narnia";    # Returns 0

use autodie;
chdir "/mnt/narnia";    # Dies
```

Nie je to modul, ale *lexikálna pragma* (ako ‹strict› a ‹warnings›).
Môžete ju vypnúť v bloku pomocou ‹no autodie›.

Vstup a výstup

Vstup a výstup

Perl pri spustení otvorí jeden vstup a dva výstupy:

- `STDIN` Štandardný vstup
- `STDOUT` Štandardný výstup
- `STDERR` Štandardný chybový výstup

Tieto slová sú *barewords*, tzn. nemajú *sigil*.

Funkcie `say`, `print` a `printf` môžu mať ako prvý argument výstup:

```
say "Default output";
say STDOUT "Standard output";           # Same thing
say STDERR "Standard error output";
```

Vstup a výstup: ‹open›

```
open FILEHANDLE, MODE, FILENAME
```

Otvorí súbor v zadanom režime a priradí ho do premennej.
Ak zlyhá, vráti ‹undef›.

Režimy otvorenia pripomínajú shell:

režim	‹open(2)›
‹<›	‹O_RDONLY›
‹>›	‹O_WRONLY | O_TRUNC | O_CREAT›
‹>>›	‹O_WRONLY | O_APPEND | O_CREAT›
‹+<›	‹O_RDWR›
‹+>›	‹O_RDWR | O_TRUNC | O_CREAT›
‹+>>›	‹O_RDWR | O_APPEND | O_CREAT›

Vstup a výstup: ‹open›

Vždy kontrolujte ‹open›!

Idióm ‹open … or die›

```
open my $handle, '<', $filename
    or die "$filename: $!";
```

Pozor na rozdielnu prioritu ‹||› a ‹or›.

Vstup a výstup: Anonymné súbory

```
open my $fh, '+>', undef
```

Vytvorí dočasný anonymný súbor. Jediný zmysluplný režim je ‹+›.

```
open my $fh, '<', \$scalar
```

In-memory file, zápis alebo čítanie do skalárnej premennej v pamäti.
Ekvivalent ‹memfd_create(2)›.

Vstup a výstup: Podprocesy

```
open my $fh, '|-', "COMMAND";
open my $fh, '-|', "COMMAND";
```

Spustí v ďalšom procese príkaz ‹COMMAND› a presmeruje ‹$fh› na vstup (‹|-›) alebo výstup do ‹$fh› (‹-|›).

Ekvivalent ‹popen(3)›.

Vstup a výstup: Dva argumenty pre ‹open›

```
open FILEHANDLE, EXPR
```

Režim a názov súboru v jednom argumente, napr. ‹"<file.txt"›.
Ak režim nie je uvedený, predpokladá sa ‹<›.

Často nájdete v starších kódoch.

Nepoužívajte
Preferujte výlučne variant s troma argumentami

Vstup a výstup: Diamond operator

```
readline [HANDLE]
<[HANDLE]>
```

V skalárnom kontexte prečíta jeden riadok, po poslednom vráti `undef`.
V zoznamovom kontexte vráti všetky riadky.

```
while (my $line = readline $fh) {       # ≈ <$fh>
    # Do something with ‹$line›.
}
```

To, čo Perl považuje za riadok, záleží od premennej `$/`.

Vstup a výstup: Diamond operator

```
while (my $line = <>) {
    # ...
}
```

Postupne otvorí všetky súbory v `@ARGV` a prečíta ich.
Spracuje aj `STDIN` pre argument `-`, alebo ak je `@ARGV` prázdne.

Perl pridá aj automatické premenné:

- `$ARGV` Názov práve spracovávaného súboru
- `ARGV` *File handle* pre otvorený súbor

Po spracovaní všetkých argumentov bude `@ARGV` prázdne.

Vstup a výstup: Diamond operator

```
while (my $line = <>) {
    # ...
}
```

Čo ak súbor obsahuje prázdny riadok?

Perl pre `<>` vo `while` testuje `defined` namiesto pravdivostnej hodnoty. Konštrukcia hore je ekvivalentná

```
while (defined(my $line = <>)) {
    # ...
}
```

Vstup a výstup: Diamond operator

Operátor `<>` volá na pozadí ekvivalent `open ARGV, $ARGV`.

Čo urobí nasledujúci príkaz?

```
perl -Mv5.36 -E 'print while <>' 'ls $HOME |'
```

A čo ak `ls` nahradíme `rm -rf`?

Vstup a výstup: Double diamond operator

Operátor `<<>>` sa správa rovnako ako `<>`,
ale používa `open` s troma argumentami. Druhý je **vždy** `<>`.

Operátor `<<>>` chápe `-` ako súbor s názvom `-`, nie `STDIN`.

Vstup a výstup: ‹getc›

```
getc [FILEHANDLE]
```

Prečíta znak zo štandardného vstupu alebo ‹FILEHANDLE›.
Po poslednom znaku ďalšie volanie vráti ‹undef›.

Vstup a výstup: ‹eof›

```
eof FILEHANDLE
eof ()
eof
```

Zistí, či **nasledujúca** I/O operácia skončí s *End of File*.

Bez argumentu testuje posledný použitý súbor.
Argument ‹()› (prázdny zoznam) testuje koniec ‹<>› a ‹<<>>›.

Ak je to možné, využite radšej návratové kódy vstupných funkcií.

Vstup a výstup: ‹chomp›

```
chomp LVALUE
chomp (LIST)
```

Funkcia ‹readline› ponecháva na konci riadka ‹\n›.
Táto funkcia to odstráni.

```
while (my $line = <$fh>) {
    chomp $line;
    # ...
}
```

Premenná ‹$/›
‹chomp› presnejšie z konca hodnôt odstráni hodnotu ‹$/›.

Vstup a výstup: Pozícia v súbore

```
seek FILEHANDLE, POSITION, WHENCE
tell [FILEHANDLE]
```

Nastaví resp. vráti pozíciu v súbore.
Hodnoty ‹WHENCE› rovnaké ako pre ‹seek(2)›, ale dajú sa získať symbolicky:

```
use Fcntl ':seek';
seek $fh, 0, SEEK_SET;
```

Vstup a výstup: Výpis

```
print FILEHANDLE LIST
printf FILEHANDLE LIST
say FILEHANDLE LIST
```

‹FILEHANDLE› môže byť *Bareword* alebo skalár.
Pre čokoľvek zložitejšie je nutné výraz obaliť v ‹{}›:

```
say $handles[0] "Hello";        # Syntax error
say $handles[0], "Hello";       # *main::STDOUTHello
say { $handles[0] } "Hello";    # Hello
```

Vstup a výstup: ‹close›

```
close FILEHANDLE
```

Vyleje buffer súboru a zatvorí ho.

Automatický deštruktor
Perl zavolá ‹close› automaticky hneď keď zanikne posledná referencia na otvorený súbor.

Kódovanie vstupu a výstupu

Kódovanie vstupu a výstupu

Vo východzom stave Perl ukladá reťazce ako *byte strings* (ASCII).

Súbor môže mať na sebe naviazané disciplíny (alebo *layers*),
ktoré menia kódovanie reťazcov pri I/O operáciách:

```
use open IN => ':encoding(UTF-8)', OUT => ':raw';
use open ':std', ':encoding(ISO-8859-2)';
```

Lexikálna pragma, ktorá mení východzie disciplíny pre `open`.
`:std` mení disciplíny aj pre vstup a výstup.

Podobný efekt má `-C` prepínač pre `perl` (`man perlrun`),
preferujte však explicitné nastavenie kódovania v skripte.

Nepoužívajte disciplínu `:utf8`!
Používa ju interne Perl, nie je určená na vstup a výstup.

Ďalšie možnosti

```
open my $fh, '<:encoding(UTF-8)', $filename;
```

Aplikuje disciplínu na práve otváraný súbor.

```
binmode FILEHANDLE, DISCIPLINE;
```

Aplikuje disciplínu na už otvorený súbor.
Používajte, len ak nemáte ‹open› pod kontrolou.

Často používané disciplíny

`:raw` Binárne dáta.

`:encoding(ENCODING)`
Nastaví kódovanie vstupu alebo výstupu na `ENCODING`.
Typicky ako `:encoding(UTF-8)`.

`:crlf` Obvykle na Windows, konvertuje výstupné `\n` na `\r\n` a naopak pri vstupe.

Disciplíny je možné na seba skladať.

Podrobnosti viď https://metacpan.org/pod/PerlIO.

Kódovanie vstupu a výstupu

```
use utf8;
```

Pragma, ktorá deklaruje, že reťazce priamo v kóde sú kódované v UTF-8.

```
use Encode;

my $chars = Encode::decode('UTF-8', $bytes);
my $bytes = Encode::encode('UTF-8', $chars);
```

Explicitne dekóduje alebo zakóduje reťazec.
https://metacpan.org/pod/Encode

Niektoré moduly očakávajú parametre v *byte strings*, iné reťazce znakov. Čítajte manuál.

Kódovanie vstupu a výstupu (zhrnutie)

Ako sa v tom má človek vyznať?
Kódovanie robí problém v takmer každom jazyku.

Praktické zásady:

- Používajte `utf8` pragmu.
- Argumenty dekódujte čo najskôr, ak ich potrebujete.
- Vstup dekódujte čo najskôr, výstup čo najneskôr (`open`, `binmode`).

A Million Billion Squiggly Characters
https://www.youtube.com/watch?v=TmTeXcEixEg
Prednáška od R. Signes (dokument, komédia, 2016)

Špeciality

‹open›: Duplikácia deskriptora

```
open my $fh, '>&', HANDLE;
open my $fh, '<&', HANDLE;
open my $fh, '<&=', DESCRIPTOR;
...
```

Otvorí ‹$fh› ako kópiu iného vstupu alebo výstupu.

Ekvivalent ‹dup2(3)›.

‹open›: Jeden argument

```
open BAREWORD
```

Do ‹*{BAREWORD}{IO}› otvorí súbor s názvom ‹*{BAREWORD}{SCALAR}›
v režime na čítanie.

Pozor
Použitie ‹open› týmto spôsobom je označované za historizmus.
Môžete však na to naraziť ešte aj v dnešných kódoch.
Sami vždy preferujte verziu s troma argumentami.

Binárne IO: ‹read›

```
read FILEHANDLE, SCALAR, LENGTH, [OFFSET]
```

Prečíta do ‹SCALAR› zadaný počet **znakov**.
Pomocou ‹OFFSET› je možné zmeniť pozíciu v skalári.

Vráti počet prečítaných znakov, 0 na konci súboru
a ‹undef› pri chybe.

> ‹**write**›**?**
> Perl má ‹write›, ale táto funkcia robí niečo iné.
> ‹man perlform›

Binárne IO: ‹pack›, ‹unpack›

```
pack TEMPLATE, LIST
unpack TEMPLATE, [EXPRESSION]
```

Prevedie zoznam hodnôt na reťazec **bajtov** podľa zadanej šablóny a naopak.

```
pack "a*xl", 'abc', 1024;

# 0x61 0x62 0x63 0x00 0x00 0x04 0x00 0x00
# ----- a* -----   x  ------- l ---------
```

Vid' ‹perldoc -f pack›