PB173 Perl

05 Vstup a výstup

Roman Lacko <xlacko1@fi.muni.cz>

Obsah

Thanatológia

> bIlujDI' yI`cheghQo'; yI`Hegh()`!
>
> It is better to `die()` than to `return` in failure.
>
> — Klingon programming proverb, perldoc autodie

`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
: úspešný beh

1-255
: neúspený beh

`die`

```
die LIST
```

Ekvivalent `throw` v iných jazykoch.

Ak má zoznam 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 u tom, kde k úmrtiu došlo.

eval

```
eval EXPRESSION
eval BLOCK
```

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

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

if ($@) {
    say STDERR "Whoopsie: $@";
    exit 1;
}
```

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

`warn`

```
warn LIST
```

Podobné ako `die`, s drobnými rozdielmi:

- Výnimku nevyhodí, len vypíše na chybový výstup.
- Ak je `LIST` prázdny, použije `$@` alebo `"Something went wrong"`.

%SIG

Hash, ktorý obsahuje reakcie na signály.
Tie zatiaľ podobne rozoberať nebudeme.

V tomto hashi sa nachádzajú aj špeciálne kľúče `__DIE__` a `__WARN__`, ktoré spustia reakciu na `die` resp. `warn`.

```
$SIG{__DIE__} = sub ($ex) {
    say "About to die due to $ex";
};
```

Výnimku je odtiaľ možné znova vyhodiť pomocou `die`.

Spracovanie výnimky zo `$SIG{__DIE__}` je možné zrušiť len pomocou `goto`.

Návod na zomieranie

Kedy použijeme ktorý spôsob?

`exit`
: Chyba na najvyššej úrovni skriptu, napr. nesprávny počet argumentov.

`die`
: Chyba vo funkcii alebo metóde, na ktorú môže chcieť volajúci kód reagovať.

`warn`
: Keď niečo vyzerá podozrivo, ale v princípe môžeme pokračovať.

Odbočka: Moduly

`Carp`

```
use Carp;

carp "There is something fishy";
croak "The argument is undefined, you twat";
confess "I can't go on";
Carp::cluck "What is this?"     # Not imported by default
```

Varianty `die` a `warn` vhodné pre moduly, chybu zahlásia v kontexte volajúceho kódu.

Funkcie `confess` a `Carp::cluck` vypíšu *stack trace*.

Odbočka: Moduly

`Try::Tiny`

```
use Try::Tiny;

try {
    EXPRESSION;
} catch {
    # Exception is caught in ‹$_›.
    say "Caught $_";
};
```

Tento modul nie je v jadre Perlu.
Môžete ho však používať v domácich úlohách, ak bude treba.

Odbočka: Moduly a pragmy

`autodie`

```
use autodie;

chdir "/wonderland";    # Dies instead of returning false value
```

Mnohé funkcie namiesto vrátenia chybového kódu zavolajú `die`.

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

Vstup a výstup

Perl pri spustení otvorí jeden vstup a dva výstupy:

- `STDIN`
- `STDOUT`
- `STDERR`

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";
```

Otvorenie súboru

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

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

Režimy otvorenia pripomínajú shell:

`<`	čítanie
`>`	zápis
`>>`	pridávanie na koniec
`+<`	čítanie so zápisom
`+>`	zápis s čítaním
`+>>`	pridávanie s čítaním

Otvorenie súboru

Vždy kontrolujte `open`!

Použite `or die`

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

Premenná `$!` je ekvivalent `errno`.
Pozor na rozdielnu prioritu `||` a `or`.

Čítanie zo súboru

```
readline EXPRESSION
<EXPRESSION>
```

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

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

Ak sa výraz s `readline` alebo `<>` použije ako podmienka cyklu, testuje sa `defined`, nie logická hodnota. Prečo je to užitočné?

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

Diamond operator

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

Postupne otvorí všetky súbory v `@ARGV` a číta ich. Spracuje aj `STDIN`, ak:

- Práve skúmaný argument je `-`
- `@ARGV` je inicilálne prázdne.

Perl pridá aj automatické premenné:

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

Na spracovaní všetkých argumentov bude `@ARGV` prázdne.

Odbočka: `chomp`

```
chomp VARIABLE
chomp (LIST)
```

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

Presnejšie z konca vymenovaných premenných odstrání hodnotu `$/`.

```
while (defined(my $line = readline $fh)) {
    chomp $line;
    # ...
}
```

Zatvorenie súboru

```
close FILEHANDLE
```

Vyleje buffer súboru a zatvorí ho.

Perl zatvorí súbor automaticky, ak všetky premenné s odkazom ukončia svoju životnosť.

Ďalšie režimy open

```
open my $fh, '+>', undef
#   or die ...;
```

Vytvorí dočasný anonymný súbor.
Jediný zmysluplný režim je +>.

```
open my $fh, '<', \$scalar
#   or die ...;
```

In-memory file, zápis alebo čítanie do skalárnej premennej v pamäti.

Ďalšie režimy `open`

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

Otvorí `$fh` ako kópiu iného vstupu alebo výstupu.

Ďalšie režimy `open`

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

Spustí v ďalšom procese príkaz `COMMAND` a presmeruje mu vstup `|-` alebo výstup `-|`.

Open `open` s menším počtom argumentov

```
open FILEHANDLE, EXPR
```

- Režim a názov súboru v jednom argumente, napr. `"<file.txt"`.
- Ak režim nie je uvedený, implicitne `<`.

```
open BAREWORD
```

- Do `*{BAREWORD}{IO}` otvorí súbor s názvom `*{BAREWORD}{SCALAR}`.

> 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.

Bezpečná verzia `<>`

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

Čo urobí nasledujúci príkaz?

```
perl -Mv5.34 -e 'print while <>' 'ls $HOME |'
```

A čo ak `ls` nahradíme `rm -rf`?

Riešenie je operátor `<<>>`, ktorý sa chová ako `<>`,
ale používa `open` s troma argumentami. Druhý je vždy `<`.

Operátor `<<>>` chápe `-` ako názov súboru.