Inline Perl Packages ...

- And other handy code snippets
- https://github.com/daviddelikat/inlinePerlPackages

What are they?

Small pieces of code (with variations) that can be dropped into a script to add a feature.

Why use them?

- Sometimes the feature is so small that it doesn't warrant a full package.
- Some packages work best when customizations can be implemented.
- Sometimes its handy to integrate the package with the script.

Why?

- Work the bugs out of a snippet.
- Work features into a snippet.
- Pick the features you need for this script.

Some examples

- Stack
- Queue
- Tee
- Status
- Other handy code bits

Stack

```
{ package STACK;  # LIFO

# a simple stack object for processing elements

sub new { bless [ ], ( ref $_[0] || $_[0] ) }

sub top { $_[0][0] }

sub topa { $_[0][0] ||= [ ] }

sub toph { $_[0][0] ||= { } }

sub pop { shift @{$_[0]} }

sub push { unshift @{$_[0]}, $_[1] }

}
```

Queue

```
{ package QUEUE;   # FIFO
# a simple queue object for processing elements
our @ISA = qw/ STACK /;
sub push { push @{$_[0]}, $_[1] }
}
```

Queue – full

```
{ package QUEUE;   # FIFO
# a simple queue object for processing elements
sub new { bless [ ], ( ref $_[0] || $_[0] ) }
sub top { $_[0][0] }
sub topa { $_[0][0] ||= [ ] }
sub toph { $_[0][0] ||= { } }
sub pop { shift @{$_[0]} }
sub push { push @{$_[0]}, $_[1] }
}
```

TEE

```
{package TEE;
require Tie::Handle;
use IO::All;
my($f1,$f2);

BEGIN {
   @TEE::ISA = qw/ Tie::Handle /;
   open $f1, '>&STDOUT';
   $f2 = io($0 . '_' . time() . '.log');
}

sub TIEHANDLE { bless [ ], TEE; }
sub FILENO{ fileno $f1 }
sub CLOSE { undef $f1; undef $f2; }
sub WRITE { $_ && $_->print( $_[1] ) for ( $f1, $f2 ); }
}
tie *STDOUT, TEE;
tie *STDERR, TEE;
```

```
{package TEE;
require Tie::Handle;
use IO::All;
my($f1,$f2);

BEGIN {
    @TEE::ISA = qw/ Tie::Handle /;
    open $f1, '>&STDOUT';
    $f2 = io($0 . '_' . time() . '.log');
}

sub TIEHANDLE { bless [ ], TEE; }
sub FILENO{ fileno $f1 }
sub CLOSE { undef $f1; undef $f2; }
sub WRITE { $_ && $_->print( $_[1] ) for ( $f1, $f2 ); }
}
tie *STDOUT, TEE;
tie *STDERR, TEE;
```

Status

```
{package STATUS;
  use Guard;
  my @statusCodeRefs;
  sub push { shift; push @statusCodeRefs, shift; guard { pop @statusC
  sub print { print join( '; ',map { $_->() } @statusCodeRefs), "\n"; }
}
```

Status – custom

```
# slightly more complicated print method

  sub print { if( @statusCodeRefs ) {
            print join( ';', map { $_->() } @statusCodeRefs ), "\n";
        } else { print "running\n" } }
```

Status – example

```
sub workerfunction {
   my($numberToProcess,$counter) = (50,0);
   my $cleanup = STATUS->push( sub { 'label:' . int($counter /
$numberToProcess * 100) . '%' } );
   while( $counter ++ < $numberToProcess ) {
        STATUS->print;
   }
}
```

Other Handy Bits

GetOpts

```
use Getopt::Long;


$result = GetOptions (
        "option:s" => \my $optionalOption,  # value is optional(:)
        "length=i" => \my $integer,         # value is required(=)
        "verbose"  => \my $verboseFlag,
        "array=s" => \my @array,            # automatically handles repeats
        "hash=s"  => \my %hash,             # repeats of <key>=<value>
        "sub=s" => sub {
                        my($option,$value) = @_;
                        # do something...
                },
        'site|sites|s=s'     => sub { push @sites, split /,/, $_[1]; },
);
```

```
use Getopt::Long;

$result = GetOptions (
        "option:s" => \my $optionalOption,  # value is optional(:)
        "length=i" => \my $integer,         # value is required(=)
        "verbose"  => \my $verboseFlag,
        "array=s@" => \my $arrayref,    # handles repeats
        "set=s%"  => \my $hashref,        # <key>=<value>
        "run=s" => sub {
                    my($option,$value) = @_;
                    # do something...
                },
        'site|sites|s=s' => sub { push @sites, split /,/, $_[1]; },
);
```

The Last Thing

Sometimes you want to delay execution to the ~very~ last moment.

Procrastination...

- END { ... }
- Perhaps the simplest option
- Not the only option
- Not *always* the best option

Test...

```
perl <<'EOF'

use Guard;

print "start of script\n";

my $guard = guard { print "guard in sub\n" };

sub mySub { $guard; }

our $oguard = guard { print "our guard\n" };

my $mguard = guard { print "my guard\n" };

END { print "end block\n"; }

sub bSub { $mguard }

print "end of script\n";

EOF
```

```
perl <<'EOF'
use Guard;
print "start of script\n";
my $guard = guard { print "guard in sub\n" };
sub mySub { $guard; }
our $oguard = guard { print "our guard\n" };
my $mguard = guard { print "my guard\n" };
END { print "end block\n"; }
sub bSub { $mguard }
print "end of script\n";
EOF
```

Output

```
start of script
end of script
end block
my guard
our guard
guard in sub
```

The 'best' solution?

```
my $g=guard { cleanup() };
sub theGuard { $g }
```

There's more in git.

https://github.com/daviddelikat/inlinePerlPackages

- Send me your snippets
- Request a snippet you wish you had

Hey look! A QR code!

Questions?