Our First Perl Program

```
#! /usr/bin/perl -w

while (<>)
{
    print;
}
```

Our First Perl Program

```
perl first /etc/passwd

perl first /etc/passwd /etc/inittab .bash_profile
```

Perl's Default Variable

```
#! /usr/bin/perl -w

while ($_ = <>)
{
    print $_;
}
```

The Strange First Line Explained

```
chmod +x first

./first /etc/passwd /etc/inittab .bash_profile
```

One of Something: Scalars

```
$greeting = "Welcome to Perl!";
$score = 20;
$score = "Goal!";
$next = <>;
$_the_answer = 42;
$WhoWantsToBeA = 1_000_000;
```

One of Something: Scalars

```
$WhoWantsToBeA = 1_000_000;

$WhoWantsToBeA = 1000000;

$WhoWantsToBeA = 1,000,000;
```

A Collection of Somethings: Arrays and Lists

```
@networks = ('Ethernet', 'Token-Ring', 'Frame-Relay', 'ATM');

print "The size of the array is: ", ( $#networks + 1 ), "\n";
$size = @networks;
print "The size of the array is: $size\n";

---

The size of the array is: 4
The size of the array is: 4
```

A Collection of Somethings: Arrays and Lists

```
@networks = ( @networks, ( 'FDDI', 'Arcnet' ) );

print "$networks[1]\n";

print "@networks[1]\n";
```

A Collection of Somethings: Arrays and Lists

```
$WhoWantsToBeA = 1,000,000;

print "The size of the array is: ", scalar @networks, "\n";

@networks = ('Ethernet', 'Token-Ring', 'Frame-Relay', 'ATM');

@networks = qw(Ethernet Token-Ring Frame-Relay ATM);
```

Hashes

```
%net_mtus;

$net_mtus{'Ethernet'} = 1500;

%net_mtus = ( 'Token-Ring', 4464, 'PPP', 1500, 'ATM', 53 );
```

Hashes

```
$net_mtus{'Token-Ring'} = 4464;
$net_mtus{'PPP'} = 1500;
$net_mtus{'ATM'} = 53;

%net_mtus = ( 'Token-Ring' => 4464,
              'PPP'        => 1500,
              'ATM'        => 53 );
```

Hashes

```
$net_mtus{'SMDS'} = undef;

$net_mtus{SMDS} = undef;

%net_mtus = ();
```

References

```
%networks = ();
@ethernets = qw( Ethernet-II IEEE802.3 IEEE802.3-SNAP );

$e_ref = \@ethernets;

$networks{'Ethernet Standards'} = $e_ref;
```

References

```
print "The Ethernet standards are: ";
print "@{$networks{'Ethernet Standards'}}\n";

%networks = ();
@ethernets = qw( Ethernet-II IEEE802.3 IEEE802.3-SNAP );

$networks{'Ethernet Standards'} = \@ethernets;
```

References

```
%networks = ();

$networks{'Ethernet Standards'} =
        [ 'Ethernet-II', 'IEEE802.3', 'IEEE802.3-SNAP' ];
```

References

```
$scalar = 42;
$refs = \$scalar;
print 'Both $scalar and $refs have the value: ', ${$refs}, ".\n";

%hash = ( 'Name' => "Paul Barry",
          'Book' => "Programming the Network with Perl",
          'Year' => 2002 );

$array_of_hashes[0] = \%hash;

print "There's a great book called ";
print "${$array_of_hashes[0]}{'Book'} by\n";
print "${$array_of_hashes[0]}{'Name'}, published in ";
print "${$array_of_hashes[0]}{'Year'}.\n";
```

References

```
${$array_of_hashes[0]}{'Name'}

$array_of_hashes[0]->{'Name'}

$array_of_hashes[0]{'Name'}
```

References

```
$array_of_hashes[0] =
    { 'Name' => "Paul Barry",
      'Book' => "Programming the Network with Perl",
      'Year' => 2002 };

%hash = ( 'Name' => "Paul Barry",
          'Book' => "Programming the Network with Perl",
          'Year' => 2002 );

$array_of_hashes[0] = { %hash };
```

References

```
${$networks{'Ethernet Standards'}}[1];
$networks{'Ethernet Standards'}->[1];
$networks{'Ethernet Standards'}[1];
```

Built-In Variables

```
man perlvar
```

if

```
if ( $net eq 'Token-Ring' )
{
    print "The network is of the token passing variety.\n";
}
```

if

```
if ( $net eq 'Token-Ring' )
{
    print "The network is of the token passing variety.\n";
}
else
{
    print "The network is NOT a token-passer.\n";
}
```

if

```
if ( $net eq 'Token-Ring' )
{
    print "Your network is of the token-passing variety.\n";
}
elsif ( $net eq 'Ethernet' )
{
    print "Your network is of the CSMA/CD variety.\n";
}
elsif ( $net eq 'Frame-Relay' )
{
    print "Your network is of the ISDN variety.\n";
}
else    #  Assuming a value of 'ATM' ...
{
    print "Your network is of the cell-switching variety.\n";
}
```

The Ternary Conditional Operator

```
if ( ($today eq 'Sat') or ($today eq 'Sun') )
{
    $do = "Play";
}
else
{
    $do = "Work";
}

$do = ( ($today eq 'Sat') or ($today eq 'Sun') ) ? "Play" : "Work";
```

while

```
while ( ($name, $value) = each %net_mtus )
{
    print "$name has the value: $value\n";
}
```

for

```
for ($i = 0; $i <= $#networks; $i++)
{
    print "$networks[$i]\n";
}
```

unless

```
unless ( $net eq 'Token-Ring' )
{
    print "The network is NOT of the token passing variety.\n";
}
```

foreach

```
foreach $i (@networks)
{
    print "$i\n";
}
```

foreach

```
foreach $name ( keys %net_mtus )
{
    print "$name has the value: $net_mtus{$name}\n";
}
```

foreach

```
$size = scalar keys %net_mtus;

print "The size of the hash is: $size\n";

foreach $name ( sort ( keys %net_mtus ) )
{
    print "$name has the value: $net_mtus{$name}\n";
}
```

do

```
$res = do
       {
           $a = 5;
           $b = 3;
           $a * $b;
       };
print $res;

do 'es1';
```

eval

```
$some_code = '$a = 5; $b = 3; $res = $a * $b; print $res;';
eval { $some_code };
```

eval

```
eval
{
    print "This is Apollo 13.  Mission Log.\n";
    print "We are half-way to the moon.\n";

    die "Houston, we have a problem!\n";
};
if ($@)
{
    print "Message from Apollo 13 - $@";
    print "Let's bring them home safely ... ";
}
else
{
    print "Maybe 13 is not that unlucky after-all ... ";
}
```

Statement Modifiers

```
print "Hello World" if defined( $should_we );

print "Hello World" unless $today eq 'Saturday';

print while (<>);
```

Statement Modifiers

```
do
{
    # Do something ...
}
while  # some condition is true;

do
{
    # Do something ...
}
until  # some condition is true;
```

Subroutines

```
sub simple {
    print "Hello from the simple subroutine!\n";
}

simple;
simple();
&simple;
&simple();
```

Processing Parameters

```
simple( "Hey!  Print this!\n" );

sub simple {
    $print_what = $_[0];

    print $print_what;
}

sub simple {
    print "@_";
}
```

Processing Parameters

```
simple( "On line one", "On line two" );

sub simple {
    print $_[0], "\n";
    print $_[1], "\n";
}

sub simple {
    print shift, "\n";
    print shift, "\n";
}
```

Returning Results

```
sub simple {
    print shift, "\n";
    my $count++;
    print shift, "\n";
    $count++;

    return( $count );
}
```

I Want An Array

```
sub return_hash_keys {
    %a_hash = @_;

    return ( wantarray ? keys %a_hash : scalar keys %a_hash );
}

%net_mtus = ( 'Token-Ring' => 4464,
              'PPP'        => 1500,
              'ATM'        => 53 );

@keys = return_hash_keys( %net_mtus );
print "Hash keys: @keys\n";

$size = return_hash_keys( %net_mtus );
print "Hash size: $size\n";
```

In-Built Subroutines

```
man perlfunc

perldoc -f print
```

References to Subroutines

```
if ($opt_u)
{
    $packet_handler = \&udp_both_packet;
}
else
{
    $packet_handler = \&tcp_both_packet;
}

&$packet_handler;
```

Perl I/O

```
$line = <STDIN>;

$line = <>;

print STDOUT "Writing to standard output, usually the screen.\n";

print "Writing to standard output, usually the screen.\n";

print STDERR "Writing to standard error.\n";

warn "Writing to standard error.\n";
```

Perl I/O

```
open MYINFILE, "readme.txt" or die "Could not open: $!";

$a_line = <MYINFILE>;
@entire_file = <MYINFILE>;

close MYINFILE;
```

Perl I/O

```
open MYOUTFILE, ">readme.txt" or die "Could not open: $!";

print MYOUTFILE "Writing to readme.txt\n";

close MYOUTFILE;
```

Perl I/O

```
open MYOUTFILE, ">>readme.txt" or die "Could not append: $!";

open MYOUTFILE, "+<readme.txt" or die "Could not read/write: $!";

open MYOUTFILE, "+>readme.txt" or die "Could not write/read: $!";
```

Variable Interpolation

```
print 'Both $scalar and $refs have the value: ', ${$refs}, ".\n";

        ...

print "There's a great book called ";
print "${$array_of_hashes[0]}{'Book'} by\n";

$mynet = "Ethernet";
$yournet = "Token-Ring";

print '$mynet is incompatible with $yournet\n';
print "\n$mynet is incompatible with $yournet\n";

---

$mynet is incompatible with $yournet\n
Ethernet is incompatible with Token-Ring
```

Packages, Modules and Objects

```
$ns = 30;

package MyNameSpace;

$ns = 100;
print 'The value of $ns is: ', $ns, "\n";

package ::main;

print 'The value of $ns is: ', $ns, "\n";

package MyNameSpace;

print 'The value of $ns is: ', $ns, "\n";
---
The value of $ns is: 100
The value of $ns is: 30
The value of $ns is: 100
----
use MyNameSpace;
```

Objects

```
use ReallyDead;

$my_cool = ReallyDead->new( First => 'Elvis', Last => 'Presley' );

print "$my_cool->{First} $my_cool->{Last} really is dead.\n";
```

```
Elvis Presley really is dead.
```

```
$my_cool = new ReallyDead First => 'Elvis', Last => 'Presley';
```