Some Ping Examples

```
ping -c4 www.linuxjournal.com

ping -c2 -b 149.153.100.255

ping -c2 224.0.0.2
```

Doing the Net::Ping Thing, 1 of 2

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

use strict;
use Net::Ping;

my $host = shift || 'localhost';

my @protos = qw( icmp tcp udp );
```

Doing the Net::Ping Thing, 2 of 2

```
foreach my $proto ( @protos )
{
    if ( ( $proto eq 'icmp' ) and ( $> ) )
    {
        print "multiping: 'icmp' only available to root.\n";
        next;
    }

    my $pinger = Net::Ping->new( $proto );

    if ( $pinger->ping( $host ) )
    {
        print "$host is alive to '$proto' pinging.\n";
    }
    else
    {
        print "$host did not respond to '$proto' pinging.\n";
    }

    $pinger->close;
}
```

Tracing Routes

traceroute www.linuxjournal.com

The CPAN `Net::Traceroute` module for Perl automates the processing of the results generated by `traceroute`. A general understanding of how `traceroute` works is important.

Getting Ready To Work With SNMP
The Simple Network Management Protocol

The `Net::SNMP` module from CPAN allows Perl programmers to interact with Managed Devices (or Agents). See the installation instructions on the next page.

If access to SNMP devices is not available on the local network (perhaps due to security concerns), an SNMP implementation can be installed in order to test any developed SNMP programs. Such an implementation is available at:

http://www.net-snmp.org

Download and install this technology into Linux.

Installing The Net::SNMP Module

```
gunzip Net-SNMP-3.60.tar.gz
tar xvf Net-SNMP-3.60.tar
cd Net-SNMP-3.6
perl Makefile.PL
make
make test
su
make install
<ctrl-D>
man Net::SNMP
perl -e 'use Net::SNMP'
```

udpstats: Working With Net::SNMP, 1 of 2

```
./udpstats 149.153.100.253 xxxxxx

Requesting 'udp' group data for: 149.153.100.253, xxxxxx, SNMPv2

UDP instance values:

1.3.6.1.2.1.7.1.0 => 18821809
1.3.6.1.2.1.7.2.0 => 5303516
1.3.6.1.2.1.7.3.0 => 1
1.3.6.1.2.1.7.4.0 => 10495825
```

udpstats: Working With Net::SNMP, 2 of 2

```
UDP table values:

1.3.6.1.2.1.7.5.1.1.149.153.1.253.496 => 149.153.1.253
1.3.6.1.2.1.7.5.1.1.149.153.1.253.520 => 149.153.1.253
1.3.6.1.2.1.7.5.1.1.149.153.1.253.1985 => 149.153.1.253
1.3.6.1.2.1.7.5.1.1.149.153.2.253.49 => 149.153.2.253
1.3.6.1.2.1.7.5.1.1.149.153.2.253.51505 => 149.153.2.253
1.3.6.1.2.1.7.5.1.1.149.153.2.253.56134 => 149.153.2.253
1.3.6.1.2.1.7.5.1.1.149.153.100.253.67 => 149.153.100.253
1.3.6.1.2.1.7.5.1.1.149.153.100.253.161 => 149.153.100.253
1.3.6.1.2.1.7.5.1.2.149.153.1.253.496 => 496
1.3.6.1.2.1.7.5.1.2.149.153.1.253.520 => 520
1.3.6.1.2.1.7.5.1.2.149.153.1.253.1985 => 1985
1.3.6.1.2.1.7.5.1.2.149.153.2.253.49 => 49
1.3.6.1.2.1.7.5.1.2.149.153.2.253.51505 => 51505
1.3.6.1.2.1.7.5.1.2.149.153.2.253.56134 => 56134
1.3.6.1.2.1.7.5.1.2.149.153.100.253.67 => 67
1.3.6.1.2.1.7.5.1.2.149.153.100.253.161 => 161
```

Working With Mnemonic Object Identifiers
The OIDs.pm Module

```
use OIDs qw( sysUpTime ipForwarding udpInDatagrams );

use OIDs qw( :interfaces );

use OIDs qw( :tcp :udp ifNumber );
```

The udpstats Source Code, 1 of 5

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

use strict;
use Socket;
use Net::SNMP qw( oid_lex_sort );
use OIDs qw( :udp );

my $snmp_host      = shift || 'localhost';
my $snmp_community = shift || 'public';
my $snmp_version   = shift || '2';
my $snmp_port      = shift || 161;
```

The udpstats Source Code, 2 of 5

```
$snmp_host = inet_ntoa( scalar gethostbyname( $snmp_host ) );

print "Requesting 'udp' group data for: $snmp_host, ";
print "$snmp_community, SNMPv$snmp_version\n\n";

my ( $snmp_session, $snmp_error ) = Net::SNMP->session(
                         -hostname     => $snmp_host,
                         -community    => $snmp_community,
                         -version      => $snmp_version,
                         -port         => $snmp_port,
                         -debug        => 0 );

if ( !defined( $snmp_session ) )
{
    die "udpstats: an error occurred: ", $snmp_error, "\n";
}
```

The udpstats Source Code, 3 of 5

```
my @udpOIDs = ( udpInDatagrams, udpNoPorts,
                    udpInErrors, udpOutDatagrams );

my $responsePDU = $snmp_session->get_request( @udpOIDs );

if ( !defined( $responsePDU ) )
{
    warn "udpstats: ", $snmp_session->error, "\n";
}

print "UDP instance values:\n\n";

foreach my $resp ( oid_lex_sort ( keys %{ $responsePDU } ) )
{
    print "$resp => ", $responsePDU->{ $resp }, "\n";
}
```

The `udpstats` Source Code, 4 of 5

```
print "\nUDP table values:\n\n";

udp_get_table( $snmp_session, udpTable );

$snmp_session->close;
```

The udpstats Source Code, 5 of 5

```
sub udp_get_table {
    my ( $sess, $requestOID ) = @_;

    my $responsePDU = $sess->get_table( $requestOID );

    if ( !defined( $responsePDU ) )
    {
        print "udpstats: OID: ", $requestOID, " : ",
                    $sess->error, "\n";
    }

    foreach my $resp ( oid_lex_sort ( keys %{ $responsePDU } ) )
    {
        print "$resp => ", $responsePDU->{ $resp }, "\n";
    }
    print "\n";
}
```

The howlongup Program - 1 of 3

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

use strict;
use Socket;
use Net::SNMP;
use OIDs qw( sysDescr sysName sysUpTime );

my $snmp_host      = shift || 'localhost';
my $snmp_community = shift || 'public';
my $snmp_version   = shift || '2';
my $snmp_port      = shift || 161;

$snmp_host = inet_ntoa( scalar gethostbyname( $snmp_host ) );
```

The howlongup Program - 2 of 3

```
print "Requesting system data items for: $snmp_host, ";
print "$snmp_community, SNMPv$snmp_version\n\n";

my ( $snmp_session, $snmp_error ) = Net::SNMP->session(
                        -hostname     => $snmp_host,
                        -community    => $snmp_community,
                        -version      => $snmp_version,
                        -port         => $snmp_port,
                        -debug        => 0 );

if ( !defined( $snmp_session ) )
{
    die "howlongup: an error occurred: ", $snmp_error, "\n";
}
```

The howlongup Program - 3 of 3

```
my @sysOIDs = ( sysDescr, sysUpTime, sysName );

my $responsePDU = $snmp_session->get_request( @sysOIDs );

if ( !defined( $responsePDU ) )
{
    warn "howlongup: ", $snmp_session->error, "\n";
}

print "Descr. => ", $responsePDU->{ OIDs::sysDescr }, "\n";
print "Name   => ", $responsePDU->{ OIDs::sysName }, "\n";
print "UpTime => ", $responsePDU->{ OIDs::sysUpTime }, "\n\n";

$snmp_session->close;
```

The howlongup Program - Example Output

```
./howlongup 149.153.100.253 xxxxxx

Requesting system data items for: 149.153.100.253, xxxxxx, SNMPv2

Descr. => Cisco Internetwork Operating System Software
          IOS (tm) MSFC Software (C6MSFC-JSV-M), Version 12.1(6)E,
          EARLY DEPLOYMENT RELEASE SOFTWARE (fc3)
          TAC Support: http://www.cisco.com/cgi-bin/
                                    ibld/view.pl?i=support
          Copyright (c) 1986-2001 by cisco Systems, In
Name   => CAT6000_MSFC
UpTime => 82 days, 18:34:42.41
```

The udpstats2 Program - Example Output

```
Requesting 'udp' group data for: 149.153.100.253, xxxxxx, SNMPv2

UpTime => 82 days, 18:35:04.64

UDP instance values:

1.3.6.1.2.1.7.1.0 => 19079331
1.3.6.1.2.1.7.2.0 => 5356687
1.3.6.1.2.1.7.3.0 => 1
1.3.6.1.2.1.7.4.0 => 10707080

UDP table values:

1.3.6.1.2.1.7.5.1.1.149.153.1.253.496 => 149.153.1.253
1.3.6.1.2.1.7.5.1.1.149.153.1.253.520 => 149.153.1.253

    ...
```

The What's Up? Configuration Text File

```
149.153.100.10:public:0
gw.itcarlow.ie:wwwwww:0
149.153.100.253:xxxxxx:0
glasnost.itcarlow.ie:public:0
193.1.206.34:zzzzzz:0
```

The What's Up? Source Code - 1 of 6

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

use strict;
use Socket;
use Net::SNMP qw( ticks_to_time );
use OIDs qw( sysUpTime );

my $whatsupfile = shift || 'whatsup.txt';
my %managed_agents = ();
```

The What's Up? Source Code - 2 of 6

```
open WHATSUPFILE_IN, $whatsupfile
    or die "whatsup: could not open file: $whatsupfile.\n";

while ( <WHATSUPFILE_IN> )
{
    my ( $host, $comm, $ticks ) = split /:/;
    chomp( $ticks );
    $managed_agents{ $host } = [ $comm, $ticks ];
}

close WHATSUPFILE_IN;
```

The What's Up? Source Code - 3 of 6

```
print "\nThe whatsup results:\n\n";

foreach my $h ( keys %managed_agents )
{
    my $snmp_host      = inet_ntoa( scalar gethostbyname( $h ) );
    my $snmp_community = $managed_agents{ $h }->[0];

    my ( $snmp_session, $snmp_error ) = Net::SNMP->session(
                    -hostname     => $snmp_host,
                    -community    => $snmp_community,
                    -translate    => [ -timeticks => 0x0 ] );

    if ( !defined( $snmp_session ) )
    {
        warn "whatsup: an error occurred: ", $snmp_error, "\n";
        next;
    }
```

The What's Up? Source Code - 4 of 6

```
my $responsePDU = $snmp_session->get_request( sysUpTime );

if ( !defined( $responsePDU ) )
{
    warn "whatsup: ", $snmp_session->error, "\n";
}

my $sysTm = $responsePDU->{ OIDs::sysUpTime };

if ( $sysTm > $managed_agents{ $h }->[1] )
{
    print "\'$h\' is still up, with uptime reported as ",
        ticks_to_time( $sysTm ), "\n\n";
}
```

The What's Up? Source Code - 5 of 6

```
    else
    {
        print "WARNING: \'$h\' may have restarted:\n";
        print "\tit has a reported uptime of ",
            ticks_to_time( $sysTm ), "\n";
        print "\tbut previously reported uptime of ",
            ticks_to_time( $managed_agents{ $h }->[1] ),
                "\n\n";
    }

    $managed_agents{ $h }->[1] = $sysTm;

    $snmp_session->close;
}
```

The What's Up? Source Code - 6 of 6

```
open WHATSUPFILE_OUT, ">$whatsupfile"
    or die "whatsup: could not open file: $whatsupfile to write.\n";

foreach my $h ( keys %managed_agents )
{
    my $c = $managed_agents{ $h }->[0];
    my $t = $managed_agents{ $h }->[1];

    print WHATSUPFILE_OUT "$h:$c:$t\n";
}

close WHATSUPFILE_OUT;
```

Being More Careful

```
use OIDs.pm qw( sysDescr );
    ...
sub snmp_connect {
    return undef unless $#_ == 3;

    my $try = Net::SNMP->session(
                -hostname     => shift,
                -community    => shift,
                -version      => shift,
                -port         => shift
              );

    my $responsePDU = $try->get_request( sysDescr );

    return ( !defined( $responsePDU ) ? undef : $try );
}
    ...
my $snmp_session = snmp_connect( $snmp_host, $snmp_community,
                                        $snmp_version, $snmp_port );
if ( !defined( $snmp_session ) )
{
    die "Whoops: an error occurred: cannot establish session.\n";
}
```

Configuring Community Strings on an SNMP Agent

```
#####################################################
# snmpd.conf
#####################################################

syslocation "Paul Barry's Laptop"
sysservices 12
syscontact  paul.barry@itcarlow.ie

rwcommunity  private
rocommunity  public
```

Setting MIB-II Data

```
my $newLocation = "Paul Barry's Desktop"

my $respPDU = $snmp_session->set_request( sysLocation,
                                                OCTET_STRING,
                                                    $newLocation );

if ( !defined( $responsePDU ) )
{
    warn "set_request did not work: ", $snmp_session->error, "\n";
}
```

iproutedata - IP Router Mapping - 1 of 2

```
sub iptables_get_table {
    my ( $sess, $requestOID ) = @_;

    my $responsePDU = $sess->get_table( $requestOID );

    if ( !defined( $responsePDU ) )
    {
        print "iproutedata: OID: ", $requestOID, " : ",
                    $sess->error, "\n";
    }

    foreach my $resp ( oid_lex_sort ( keys %{ $responsePDU } ) )
    {
        my $oid_pat = OIDs::ipRouteDest;
        if ( $resp =~ /^$oid_pat\./ )
        {
            print "Dest:    $resp => ",
                        $responsePDU->{ $resp }, "\n";
        }
```

iproutedata - IP Router Mapping - 2 of 2

```
        $oid_pat = OIDs::ipRouteNextHop;
        if ( $resp =~ /^$oid_pat\./ )
        {
            print "NextHop: $resp => ",
                        $responsePDU->{ $resp }, "\n";
        }

        $oid_pat = OIDs::ipRouteType;
        if ( $resp =~ /^$oid_pat\./ )
        {
            print "Type:    $resp => ",
                        $responsePDU->{ $resp }, "\n";
        }
    }
    print "\n";
}

iptables_get_table( $snmp_session, ipRouteTable );
```

ipdetermine - IP Router Mapping - 1 of 4

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

use strict;
use Socket;
use Net::SNMP;
use OIDs qw( sysDescr
             ipRouteTable ipRouteDest
             ipRouteType ipRouteNextHop );

sub snmp_connect {
    return undef unless $#_ == 3;

    my $try = Net::SNMP->session(
                  -hostname     => shift,
                  -community    => shift,
                  -version      => shift,
                  -port         => shift
              );

    my $responsePDU = $try->get_request( sysDescr );

    return ( !defined( $responsePDU ) ? undef : $try );
}
```

ipdetermine - IP Router Mapping - 2 of 4

```
my $snmp_host      = shift || 'localhost';
my $snmp_community = shift || 'public';
my $snmp_version   = '2';
my $snmp_port      = 161;

$snmp_host = inet_ntoa( scalar gethostbyname( $snmp_host ) );

print "Determining 'ip' next hop data for: $snmp_host, ";
print "$snmp_community, SNMPv$snmp_version\n\n";

my $snmp_session = snmp_connect( $snmp_host, $snmp_community,
                                      $snmp_version, $snmp_port );

if ( !defined( $snmp_session ) )
{
    die "ipdetermine: an error occurred: cannot " .
                "establish session.\n";
}
```

ipdetermine - IP Router Mapping - 3 of 4

```
my $responsePDU = $snmp_session->get_table( ipRouteTable );

if ( !defined( $responsePDU ) )
{
    print "ipdetermine: OID: ", ipRouteTable, " : ";
    print $snmp_session->error, "\n";
}

my @dests = ();

foreach my $resp ( keys %{ $responsePDU } )
{
    my $oid_pat = ipRouteDest;
    if ( $resp =~ /^$oid_pat\./ )
    {
        @dests = ( @dests, $responsePDU->{ $resp } );
    }
}
```

ipdetermine - IP Router Mapping - 4 of 4

```
my %unique_next_hops = ();

foreach my $dest ( @dests )
{
    my $oid_type = ipRouteType . ".$dest";

    if ( $responsePDU->{ $oid_type } eq '4' )
    {
        my $oid_nexthop = ipRouteNextHop . ".$dest";
        $unique_next_hops{ $responsePDU->{ $oid_nexthop } }++;
    }
}

foreach my $unique ( keys %unique_next_hops )
{
    print "Next hop: $unique\n";
}

$snmp_session->close;
```

IP Router Mapping - Example Output

```
./ipdetermine 149.153.100.10 public

Determining 'ip' next hop data for: 149.153.100.10, public, SNMPv2

Next hop: 149.153.100.12
Next hop: 149.153.1.2
Next hop: 149.153.2.22
Next hop: 149.153.2.100
```