MasterIT - Scripting Es2a

Perl

(Practical Extraction and Report Language)

"There's more than one way to do it." -- Larry Wall, Author of the Perl Programming Language

Sandro.Angius@lnf.infn.it

20/11/2002

eric.pl

- Perl puo' essere poco leggibile:

```
#!/usr/local/bin/perl5
$!=$;=$_+(++$_);
($:,$~,$/,$^,$*,$@)=$!=~/.(.)...(.)(.)....(.)..(.)..(.)/;
`$^$~$/$: $^$*$@$~ $_>&$;`;
```

myname.pl

- Oppure in caso di un attacco di egocentrismo:

```
#!/usr/local/bin/perl5
$!=$;=$_+(++$_);
($:,$@,$~,$/,$^,$*,$#)= $! =~ /.(.).(.).(.)(.)....(.)..(.).(.)./;
$!=1<<2;
($, , $.)= $! =~ /.(.)..................(.)../;
`$^$~$/$: $@$.$,${#}${*}${:} $_>&$;`;
```

“Traduzione” di eric.pl e myname.pl

- *cat eric_easy.pl*

```
#!/usr/local/bin/perl5
$g = 1;    $! = $h = 2;
#                              N o   suc  h   file   or   directory
($a,$b,$c,$d,$e,$f) = $! =~ /.(.)...(.)(.)....(.)..(.)..(.)/;
`$d$b$c$a $d$e$f$b $g>&$h`;
```

- *cat myname_easy.pl*

```
#!/usr/local/bin/perl5
$h = 1;    $! = $i = 2;
#                                 N o   s  uc  h   file   or   directory
($a,$b,$c,$d,$e,$f,$g) = $! =~ /.(.).(.).(.)(.)....(.)..(.).(.)./;
$! = 4;
#                  In terrupted system c all
($j,$k) = $! =~ /.(.)..................(.)../;
`$e$c$d$a $b$k$j$g$f$a $h>&$i`;
```

ese5.pl

- Versione Perl del programma ese5
- Dato un numero ip e il numero di bit della netmask viene tornato l'identificativo ip della rete di appartenenza.

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

# Conversione in Perl dell'esercizio 5, Referimenti:
# /afs/lnf.infn.it/project/master.it/doc/...
#       .../Scripting/EsScripts/ese5.{sh|csh|awk}
#       .../Scripting/Es1a_shellgrepsedawk.pdf

use strict;

my($numip, $nbmsk, @byteip, $subip,
   $netid, @bnetid, $netbc, @bnetbc);

if ( $#ARGV != 1 ) {
    print "Usage: $0 <Numero IP> <Numero bit maschera IP>\n";
    exit -1;
}

($numip, $nbmsk) = @ARGV;

if ( $numip !~ /^(\d{1,3}\.){3}\d{1,3}$/ ) {
    print "Numero IP non valido, deve essere A.B.C.D\n";
    exit -2;
}

if (($nbmsk !~ /\d+/) || ($nbmsk < 1) || ($nbmsk > 32)) {
    print "Il numero di bit maschera deve essere tra 1 e 32\n";
    exit -3;
}

@byteip = split('\.', $numip);
```

Ese5.pl - Pag. 1/2

```
foreach $subip (@byteip) {
    if ($subip > 255) {
        print "Numero IP non valido, $subip > 255 !!!\n";
        exit -4;
    }
}

my ($nid, $nbc) = &gnetid($numip, $nbmsk);
print $numip, " ---> ", $nid, " - ", $nbc, "\n";

sub gnetid {
    my($numip, $nbmsk) = @_;
    my($bnetid, $bnetbc);

    $nbmsk = 32 - $nbmsk; # Ci interessano i bit da azzerare
    $numip = (unpack("N", pack("C4", split('\.', $numip))) >> $nbmsk) << $nbmsk;

    # $bnetid: Indirizzo di rete
    $bnetid = join(".", unpack("C4", pack("N", $numip)));
    # Mettiamo a 1 i bit "host", $bnetbc: Indirizzo di broadcast
    $bnetbc = join(".", unpack("C4", pack("N", $numip | ((1 << $nbmsk) - 1))));
    ($bnetid, $bnetbc);
}
```

Ese5.pl - Pag. 2/2

ese5.pl: Subroutine gnetid($numip,$nbmsk)

```
sub gnetid {

    my($numip, $nbmsk) = @_;

    my($bnetid, $bnetbc);

    $nbmsk = 32 - $nbmsk; # Ci interessano i bit da azzerare

    $numip = (unpack("N", pack("C4", split('\.', $numip))) >> $nbmsk) << $nbmsk;

    # $bnetid: Indirizzo di rete
    $bnetid = join(".", unpack("C4", pack("N", $numip)));

    # Mettiamo a 1 i bit "host", $bnetbc: Indirizzo di broadcast
    $bnetbc = join(".", unpack("C4", pack("N", $numip | ((1 << $nbmsk) - 1))));

    ($bnetid, $bnetbc); # Ritorna i risultati
}
```

ese5.pl: Subroutine gnetid($numip,$nbmsk)

```
>>> $numip = ((unpack("N", pack("C4", split('\.', $numip))) >> $nbmsk)) << $nbmsk;

        split('\.', $numip)          # Divide la stringa $numip e crea un vettore

     pack("C4",                 )    # "Impacca" il vettore in 4 byte contigui (MSB first)

   unpack("N",                    )  # Considera i 4 byte come un unico 32bit "N" (=> MSB first)

  (( ....      ) >>          $nbmsk)  # "Elimina" tramite shift a destra i bit di "hosts"

( ....              ) << $nbmsk      # E reinserisce lo stesso numero di bit "0" a destra
```

```
>>> $bnetid = join(".", unpack("C4", pack("N", $numip)));

        pack("N", $numip)            # Trasforma il "numero" $numip in un "N" (=> MSB first)

      unpack("C4",      ....)        # Separa i 4 byte in un vettore di 4 elementi

     join(".",                  )    # Trasforma il vettore in una stringa separata da "."
```

>>> Simile operazione viene eseguita per $bnetbc

Esempio di uso dei “Package”

- Definizione di un “Package” *nsc.pm* per la conversioni da numero in cifre a numero in lettere e viceversa.
- Il programma Per *testnscpackage.pl* cerca di convertire i paramatri che gli vengono passati con il package *nsc.pm*.

```
package nsc; # nsc.pm

use strict;

my(@da0a19) = ("zero", "uno",   "due",  "tre",  "quattro",
   "cinque", "sei",  "sette", "otto", "nove", "dieci",
   "undici", "dodici", "tredici", "quattordici", "quindici",
   "sedici", "diciassette", "diciotto",  "diciannove");

my(@da0ad9) = (@da0a19[0..12], "3dici", @da0a19[14..16],
               "dicias7", "dici8", "dician9");

my(@decine) = ("", "", "venti", "trenta", "quaranta",
 "cinquanta", "sessanta", "settanta", "ottanta", "novanta");

my(@decin3) = ($decine[2], "3nta", @decine[4..9]);

my(@mults)  = ("", "mille", "unmilione", "unmiliardo");

my(@multp)  = ("", "mila",  "milioni",   "miliardi",
               "miliardi", "dimiliardi");

my($PARTIALRES) = 0;

sub partialres {
    my($val) = @_; $PARTIALRES = ($val eq "1" ? 1 : 0);
}

sub s2nwchk {
    my($strnum) = @_;
    my($chks2n, $dummy) = &s2n($strnum);
    if ($chks2n ne "-1") {
        my($chkn2s) = &n2s($chks2n);
        if (($chkn2s ne "-1") && ($chkn2s eq lc $strnum)) {
            return $chks2n;
        }
        $dummy = $chks2n;
    }
    if ($PARTIALRES) { return (-1, $dummy); }
    return -1;
}
```



```
# Da numero in cifre a numero in lettere (12 --> dodici)
# 0 <= $numero < 2^32
sub n2s {
    my($numero) = @_;
    my($strres) = "";
    my($tmpval, $unita, $cnt);
    if ($numero =~ /\D/) { return -1; }
    $numero =~ s/^0{2,}/0/g;
    $numero =~ s/^0(.)/$1/g;
    if ($numero < 20) {
        $strres = $da0a19[$numero];
    }
    elsif ($numero < 100) {
        $numero =~ /^(.+)(.)$/;
        $unita  = $2; $tmpval = $1;
        $strres = $decine[$tmpval];
        chop $strres if (($unita==1)||($unita==8));
        $strres .= $da0a19[$unita] if ($unita);
    }
    elsif ($numero < 1000) {
        $tmpval  = $numero % 100;
        $unita   = int($numero / 100);
        $strres  = &n2s($unita) x ($unita > 1)."cento";
        $strres .= &n2s($tmpval) if ($tmpval);
    }
    else {
        for ($cnt = 3; $cnt < 19; $cnt += 3) {
            if ($numero < (10**($cnt+3))) {
                $tmpval  = $numero % (10**$cnt);
                $unita   = int($numero / (10**$cnt));
                $strres  = ($unita > 1 ?
                     &n2s($unita) . @multp[$cnt / 3] :
                     @mults[$cnt / 3]);
                $strres .= &n2s($tmpval) if ($tmpval);
                last;
            }
        }
    }
    $strres;
}
```

```
# Da numero in lettere a numero in cifre (dodici --> 12)
# L'output e' corretto per stringhe corrette! (centocento --> 100100)
# Input ok tra "zero" e "novecentonovantanovemilanovecentonovantanove"

sub s2n{

    my($strnum) = @_;
    $strnum = lc $strnum;     my($oval)    = $strnum;
    my($cifra, $lchar);       my($cnt)     = 0;

    foreach $cifra (@da0ad9) {
        $strnum =~ s/$cifra/$cnt/g;
        $cnt++;
    }

    $cnt = 2;
    foreach $cifra (@decin3) {
        $cifra =~ /^(.+)(.)$/;
        $strnum =~ s/$1($2|\d)/${cnt}_$1/g;
        $cnt++;
    }

    $strnum=~ s/_(\d)/$1/g;                    # <decina>{1,8}
    $strnum=~ s/_.(\d)/$1/g;                   # <decina>{2-7,9}
    $strnum=~ s/_./0/g;                        # <decina>0
    $strnum=~ s/(\d)cento(\d*)/100*$1+"0$2"/eg;# {n}<centinaia>{0-99}
    $strnum=~ s/cento(\d*)/100+"0$1"/eg;       # 1{0-99}
    $strnum=~ s/mille(\d*)/1000+"0$1"/eg;      # 1{0-999}
    $strnum=~ s/(\d+)mila(\d*)/1000*$1+"0$2"/eg;# <migliaia>{0-999}

    if ($strnum =~ /\D/) {
        if ($PARTIALRES) {
            return (-1, $strnum);
        }
        else {
            return -1;
        }
    }
    $strnum;
}
```

(File: testnscpackage.pl)

```
#!/usr/local/bin/perl5 -w

use strict;
use nsc; # deve essere reperibile tramite @INC o nella
         # directory corrente

my($instr, $dnum, $pstr, $snum);

nsc::partialres(1); # vogliamo anche la conversione parziale

foreach $instr (@ARGV) {
    ($dnum,$pstr)= nsc::s2nwchk($instr);
    $snum = nsc::n2s($instr);
    if (($dnum < 0) && ($snum eq "-1")) {
        print "L'argomento >$instr< non e' un numero ";
        print "(Risultato parziale: $pstr)\n";;
    }
    else {
        if ($snum eq "-1") {
            $snum = $dnum;
        }
        print "$instr ---> $snum\n";
    }
}
```

text.pl

- Dato un file di testo in input contare le parole (di minimo 2 caratteri), i numeri e gli indirizzi email contenuti nel testo.
- Per le sole parole riportare le occorrenze di ogni singola parola e la loro distribuzione per lunghezza.

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

use strict;

my $MAILMATCH = '^\w+(\.\w+)*\@\w+(\.\w+)+$';
my $NUMBMATCH = '^\d+$';
my %email   = ();  my $tote = 0;
my %numbers = ();  my $totn = 0;
my %words   = ();  my $totw = 0;
my %unknown = ();  my $totu = 0;
my %dlenw   = ();
my ($line, $word, $lenw);

while (<STDIN>) {

    s/[^\w@.]+/ /g; # Solo alfanumerici _ @ .
    s/^\s+//g;      # Via gli spazi/tab iniziali
    s/\s+$//g;      # Via anche quelli finali

    $line = lc $_;  # Tutto in minuscolo!

    while ($line ne "") {
        $line =~ /^(\S+)\s*(.*)$/; # Tronca in due
        $word = $1;                # Token in analisi
        $line = $2;                # Resto della linea

        if ($word =~ /$MAILMATCH/) { # E' un email?
            $tote++;                 # Incrementa Totale
            $email{$word}++;         # Aggiorna Dizionario
            next;                    # Prossimo Token
        }
        if ($word =~ /[@._]/) {
            $word =~ s/[@._]/ /g;    # Elimina "@" e "."
            $line = $word . ' ' . $line; # Ricomponi
            $line =~ s/^\s+//g;
            $line =~ s/\s+$//g;
            next;                    # Prossimo Token
        }
```



```
        if ($word =~ /$NUMBMATCH/) { # E' un numero?
            $totn++;                 # Incrementa Totale
            $word =~ s/^0+(.)/$1/g;  # Via "0" non signif.
            $numbers{$word}++;       # Aggiorna Dizionario
            next;                    # Prossimo Token
        }
        if ($word =~ /^[a-z]{2,}$/) {# Parola > 2 chars?
            $totw++;                 # Incrementa Totale
            $words{$word}++;         # Aggiorna Dizionario
            $lenw = length($word);   # Lunghezza Parola
            $dlenw{$lenw}++;         # Aggiorna Statistica
            next;                    # Prossimo Token
        }
        $totu++;                     # Incrementa "unknown"
        $unknown{$word}++;           # Aggiorna Dizionario
    }
}

print " ", "=" x 25, " Results ", "=" x 26, "\n\n";
print " EMAIL ADDRESSES SUMMARY:\n\n";
print "  Index  Freq    Email Record\n\n" if ($tote);
&PrintResult(\%email, "email addresses", $tote);
print " WORDS SUMMARY:\n\n";
print "  Index  Freq    Word\n\n" if ($totw);
&PrintResult(\%words, "words (>2 chars)", $totw);
print " WORDS LENGTHS SUMMARY:\n\n";
print "  Index  Freq    Word length\n\n" if ($totw);
&PrintResult(\%dlenw, "words lengths", $totw);
print " NATURAL NUMBERS SUMMARY:\n\n";
print "  Index  Freq    Number Record\n\n" if ($totn);
&PrintResult(\%numbers, "natural numbers", $totn);
print " UNDEFINED RECORDS SUMMARY:\n\n";
print "  Index  Freq Undefined Record\n\n" if ($totu);
&PrintResult(\%unknown, "undefined record", $totu);

exit;
```

```
sub H2HofL { # Hash to Hash of List

    my($hashin, $hashout) = @_;
    my($key, $value);

    while (($key, $value) = each %$hashin) {
        push @{$$hashout{$value}}, $key;
    }
}

sub PrintResult {

    my($hashin, $msg, $tot) = @_;
    my(%freq)   = ();
    my($cnt, $numf, $word);

    &H2HofL($hashin, \%freq);

    $cnt = 0;
    foreach $numf (sort { $b <=> $a } keys %freq){
        # foreach $numf (reverse sort { $a <=> $b } keys %freq){
        foreach $word ( sort @{ $freq{$numf} } ) {
            printf("%7d %7d    %s\n", ++$cnt, $numf, $word);
        }
    }

    print  " ", "=" x 60, "\n";
    printf " %d different %s on total of %d\n", $cnt, $msg, $tot;
    print  " ", "=" x 60, "\n\n\n";
}
```

Bibliografia

- Perl Programmers Reference Guide
  (una copia si trova in /afs/lnf.infn.it/system/doc/perl/perlman.pdf)

- Copia degli scripts si trova in:
  /afs/lnf.infn.it/project/master.it/doc/Scripting/EsScripts/