```
#!/bin/bash

#set -x

################################################################################
################################################################################
#########################
# EXPLAINING THE LOGIC:
#
#
#
# Here, I am dividing my thought process into certain points:
#
#   (1) From input, find the mask value (say /24 or /28) and see how many
bits are available to be borrowed (say 8 or 4)
#
#   by subtracting the mask value from 32.
#
#   (2) Step 1 above when done with power of 2 tells the number of hosts
availabe (say 2^8=256 or 2^4=16)
#
# LOGIC OF ABOVE TWO (STEPS 1-2) IS IMPLEMENTED BETWEEN LINES 86-93
#
#
#
#   (3) Now Make Iterations on the numner of hosts from step 2 so that
they are divided into number of groups given.
#
#   This iteration is to be made one time less than the nuber of groups
(say 2 times if 3 groups are needed)
#
#   Maintain the number of hosts in an array that keeps growing till the
iteration runs.
#
# LOGIC OF STEP 3 IS IMPLEMENTED IN RECURSIVE FUNTION
host_range_calculator() FROM LINES 11-55.
#
#
#
#   (4) While reading the output and calculating host range, we will use
the array returned by funtion host_range_calculator() from step 3 and read
it in reverse order     #
# LOGIC OF STEP 4 IS IMPLEMENTED IN RECURSIVE FUNTION print_function()
FROM LINES 57-80
#
#
#
################################################################################
################################################################################
#########################

new_diff=0
#This recursive function implements logic mentioned in Step (3) mentioned
in heading
```

```
host_range_calculator(){
  if [ "$new_diff" -ne 0 ]
  then
    if [ "$new_diff" -gt 256 ] || [ "$new_diff" -lt 0 ]
    then
      echo "Host range exceeded in host range cal. Printing best possible
grouping"
    fi
  fi

  #Catching function arguments into variables
  division_cycles=$1
  no_of_div_iteration_needed=$2
  starting_no=$3
  ending_no=$4
  valid_bit_count=$5

  #Performing partition of primary group
  while [ "$division_cycles" -le "$no_of_div_iteration_needed" ]
   do
    #Performing first partition of first part of the present subgroup
    divided_group_host_count_gr1=$(( $ending_no/2 ))
    host_array[$index_count]=$divided_group_host_count_gr1
    index_count=$(( $index_count + 1 ))
    host_array[$index_count]=$divided_group_host_count_gr1
    group1_start=$starting_no
    group1_end=$(( $divided_group_host_count_gr1 - 1 ))
    group2_start=$divided_group_host_count_gr1
    group2_end=$valid_bit_count
    index_count=$(( $index_count + 1 ))
    division_cycles=$(( $division_cycles + 1 ))
    #Performing first partition of the second part of present subgroup
    if [ "$division_cycles" -le "$no_of_div_iteration_needed" ]
    then
      new_diff=$(( $group2_end-$group2_start ))
      #new_diff=$group2_start
      if [ "$new_diff" -gt 0 ]
      then
        divided_group_host_count_gr2=$(( $new_diff/2 ))
        host_array[$index_count]=$divided_group_host_count_gr2
        index_count=$(( $index_count + 1 ))
        host_array[$index_count]=$divided_group_host_count_gr2
        index_count=$(( $index_count + 1 ))
        division_cycles=$(( $division_cycles + 1 ))
      else
        echo "No further goups can be made. Proceeing with the best
possible grouping"
      fi
    fi
    #Performing second partition of the second part of present subgroup
    if [ "$division_cycles" -le "$no_of_div_iteration_needed" ]
    then
      if [ "$new_diff" -gt 0 ]
      then
```

```
        #new_diff=$(( $group2_start-$group1_start ))
        new_diff=$group2_start
        divided_group_host_count_gr1=$(( $new_diff/2 ))
        host_array[$index_count]=$divided_group_host_count_gr1
        index_count=$(( $index_count + 1 ))
        host_array[$index_count]=$divided_group_host_count_gr1
        division_cycles=$(( $division_cycles + 1 ))
        index_count=$(( $index_count + 1 ))
      else
        echo "No further goups can be made. Proceeing with the best
possible grouping"
      fi
    fi
    #Calling next cycle of partitions passing first subgroup as the
primary group
    if [ "$division_cycles" -le "$no_of_div_iteration_needed" ]
    then
      host_range_calculator $division_cycles $no_of_div_iteration_needed
$starting_no $divided_group_host_count_gr1 $valid_bit_count
    fi
   done
}


#This recursive function implements logic mentioned in Step (4) mentioned
in heading
print_function(){
  #Catching function arguments into variables
  starting_no=$1
  host_array_length=$2
  sub_group_length=$3
  oct1=$4
  oct2=$5
  oct3=$6
  pow=1
  iter=$host_array_length
  init=0

  #Iterate through host_array in reverse order upto the extent of desired
number of groups
  while [ "$iter" -gt "$(( $host_array_length - $sub_group_length ))" ]
  do
    current_bits=${host_array[$(( $iter - 1 ))]}
    #This loop finds the bits that can be borrowed.
    #These borrowed bits derives the new mask (Say /25 or /26 or /29)
    while [ "$pow" -lt "$current_bits" ]
    do
      init=$(( $init + 1 ))
      pow=$(( 2**$init ))
    done
    #current_mask is the final value of mask derived here
    current_mask=$(( 32 - $init ))
    host_count_bit_keeper=$(( $host_count_bit_keeper + $current_bits ))
    if [ "$host_count_bit_keeper" -le 256 ]
```

```
    then
     echo "subnet=$oct1.$oct2.$oct3.$starting_no/$current_mask
network=$oct1.$oct2.$oct3.$starting_no broadcast=$oct1.$oct2.$oct3.$((
$starting_no + $current_bits - 1 )) gateway=$oct1.$oct2.$oct3.$((
$starting_no + 1 )) hosts=$(( $current_bits - 3 ))"
    else
     remaining_bits=$(( $valid_bit_count - $starting_no ))
     if [ $starting_no -ge 256 ]
     then
       echo "***** WARNING: No more partition possible, remaining
$remaining_bits hosts will be wasted *****"
       exit 1
     fi
     echo "***** WARNING: Host range exceeded. Printing best possible
grouping considering the remaining hosts out of available bits *****"
     remaining_bits=$(( $valid_bit_count - $starting_no ))
     echo "subnet=$oct1.$oct2.$oct3.$starting_no/$current_mask
network=$oct1.$oct2.$oct3.$starting_no broadcast=$oct1.$oct2.$oct3.256
gateway=$oct1.$oct2.$oct3.$(($starting_no + 1)) hosts=$(($remaining_bits -
3))"
     exit 1
   fi
   starting_no=$(( $starting_no+$current_bits ))
   iter=$(( $iter - 1 ))
  done
}

#Split first parameter with slash '/'. This gives the IP and the subnet
IFS='/' read ip mask <<< "$1"
#Split IP with dots '.' This gives all four octets in the IP given
IFS='.' read oct1 oct2 oct3 oct4 <<< "$ip"

#Doing basic checks
if [ "$#" -ne 2 ]
then
  echo "Need atleast 2 arguments, $# given"
  echo "USAGE: ./subnetter.sh <IP/CIDR Mask> <No of groups needed>"
  echo "EXAMPLE: ./subnetter.sh 192.168.0.0/24 4"
  exit 1
fi
if [ "$mask" -lt 24 ]
then
  echo 'Subnet Mask Not valid.\n Please enter a subnet mask smaller than
or equal to 24'
  exit 1
else
  #On the basis of mask given, finding the valid bits as mentioned in step
(1) in Heading
  valid_bits=$(( 32 - $mask ))
  valid_bit_count=$(( 2**$valid_bits ))
fi

#Initialize variables
division_cycles=1
```

```
no_of_div_iteration_needed=$(( $2 - 1 ))
starting_no=$oct4
ending_no=$valid_bit_count
index_count=0

#Calling host_range_calculator(). This funtion return the array with host
groups after group division (As mentioned in Step (3) in the Heading
above"
host_range_calculator $division_cycles $no_of_div_iteration_needed
$starting_no $ending_no $valid_bit_count
#Above funtion returns host_array containing group of hosts. We are
finding its lenth in the next line
host_array_length=${#host_array[@]}

#This funtion does the printing task needed for output (As explained in
Step (4) in the heading above
#It reads array host_array in reverse order only to the extent of the
number of groups needed
print_function $oct4 $host_array_length $2 $oct1 $oct2 $oct3
```