///BUILDWORDS///

```
#! /bin/bash
grep -E "<td>.+</td>" | #grab everything that is surrounded by
<td> \
tags
sed "1~2d" | #Remove every other line
sed "s/<[^>]*>//g" | #remove the <td> tags
tr [:upper:] [:lower:] | #convert all uppercase to lowercase
tr "\`" "\'"  | #transform ` to '
sed -E "s/,\s|\s/\n/g" | #replace , with newline and remove
whitespa\
ces
grep "^[pk\' mnwlhaeiou]\{1,\}$" | #The line starts with any one
of \
those characters, And those characters may appear at least 1
time, A\
nd the strong must also end with one of those characters
sed '/ ^$/d' | #removes blank lines
sort -u
```

///SHUF.PY///

```
import random, sys
from optparse import OptionParser
import argparse
import string

class shufP:
    def __init__(self,filename):
        f = open(filename, 'r')
        self.lines = f.readlines()
        f.close()

def shuff(indexes, numlines, inputLines):
    shuffIndex = random.sample(indexes , numlines)
    for index in shuffIndex:
        line = inputLines[index]
        line = str(line)
        if line.endswith('\n'):
            sys.stdout.write(line)
        else:
            sys.stdout.write(line + '\n')

def shuffRep(numlines, inputLines):
    if numlines == -1:
        while True:
            line = str(random.choice(inputLines))
            if line.endswith('\n'):
                sys.stdout.write(line)
            else:
                sys.stdout.write(line + '\n')
    else:
        for i in range(0,numlines):
            line = str(random.choice(inputLines))
            if line.endswith('\n'):
                sys.stdout.write(line)
            else:
                sys.stdout.write(line + '\n')

def zeroArg():
    try:
        lines = sys.stdin.readlines()
        return lines
    except IOError as e:
        errno, strerror = e.args
        parser.error("I/O error({0}): {1}".format(errno,
strerror))

def main():
    version_msg = "%prog 2.0"
    usage_msg = """%prog [OPTION] .... FILE
    shuffles input by outputting a random permutation of its
input lines/
    Each output permuation is equally likely."""
    parser = OptionParser(version=version_msg, usage=usage_msg)
    parser.add_option("-i", "--input-range", action="store",
                      dest="inputRange", default="",
                      help="treat numbers LO-HI in range as
input lines")
    parser.add_option("-n", "--head-count", action="store",
dest="count",
                      help="output at most COUNT lines")
    parser.add_option("-r", "--repeat", action="store_true",
                      dest="repeat", default=False,
                      help="output lines can be repeated")
    options, args = parser.parse_args(sys.argv[1:])
    if len(args) > 1:
        parser.error("wrong number of operands")

    if len(args) == 0:
        input_file = "-"
    else:
        input_file = args[0]
    indexes = []
    lines = []
    if options.inputRange != "":
        if "-" in options.inputRange:
            lines = []
            a, b = options.inputRange.split('-')
            try:
                a, b = int(a), int(b)
            except:
                parser.error("invalid input range " +
options.inputRange)
            lines.extend(range(a,b+1))
            numlines = len(lines)
        else:
            parser.error("invalid input range " +
options.inputRange)
    else:
        try:
            if input_file == '-':
                lines = zeroArg()
                numlines = len(lines)
            else:
                generator = shufP(input_file)
                numlines = len(open(input_file).readlines())
                lines = generator.lines
        except IOError as e:
            errno, strerror = e.args
            parser.error("I/O error({0}): {1}".
                         format(errno, strerror))
    for i in range (0,numlines):
            indexes.append(i)
    if options.count != None:
        try:
            options.count = int(options.count)
        except:
            parser.error("invalid line count " +
str(options.count))
        if options.count < 0:
            parser.error("invalid line count " +
str(options.count))
        if options.count > numlines and not options.repeat:
            options.count = numlines
        numlines = options.count

    if options.repeat:
        if options.count == None:
            shuffRep(-1,lines)
        else:
            shuffRep(numlines, lines)
    else:
        shuff(indexes, numlines, lines)
if __name__ == "__main__":
    main()
```

///SAMELN.SH///

```
#!/bin/bash

dir=$1

ret=`ls $dir | sort`
hidden=`ls -a $dir | grep '^\.' | sort`
let size=0
declare -a FDarr

restore="$IFS" #sets word splitting definition (ie word
boundary)
IFS=$'\n'

for fileH in $hidden
do
    if [ -L "$dir/$fileH" ]
    then
    :
    elif [ ! -f "$dir/$fileH" ]
    then
    :
    elif [ ! -r "$dir/$fileH" ]
    then
    echo "$fileH is not readable"
    else
    FDarr[$size]="$dir/$fileH"
    let size=size+1
    fi
done

for file in $ret
do
    if [ -L "$dir/$file" ]
    then
    :
```

```
    elif [ ! -f "$dir/$file" ]
    then
    :
    elif [ ! -r "$dir/$file" ]
    then
        echo "$file is not readable"
    else
        FDarr[$size]="$dir/$file"
        let size=size+1
    fi
done

for (( i=0; i<$size; i++ ))
do
    for (( j=i+1; j<$size; j++ ))
    do
        cmp -s "${FDarr[$i]}" "${FDarr[$j]}"
        if [ $? -eq 0 ]
        then
            ln -fP "${FDarr[$i]}" "${FDarr[$j]}"
        FDarr[$j]=FDarr[$i]
        fi
    done
done
```

///RANDMAIN.MK

```
OPTIMIZE = -O2
CC = gcc
CFLAGS = $(OPTIMIZE) -g3 -Wall -Wextra -march=native -
mtune=native -\
mrdrnd
randlibsw.so:
        $(CC) $(CFLAGS) -fPIC -c randlibsw.c -o randlibsw.o
        $(CC) $(CFLAGS) -shared -o randlibsw.so randlibsw.o

randlibhw.so:
        $(CC) $(CFLAGS) -fPIC -c randlibhw.c -o randlibhw.o
        $(CC) $(CFLAGS) -shared -o randlibhw.so randlibhw.o

randmain:
        $(CC) $(CFLAGS) -c randcpuid.c -o randcpuid.o
        $(CC) $(CFLAGS) -c randmain.c -o randmain.o
        $(CC) $(CFLAGS) -ldl -Wl,-rpath=$(PWD) randcpuid.o
randmain.\
o -o randmain
```