Week 2

tr -cs 'A-Za-z' '[\n*]' < assign2.html | sort -u | comm -23 - words

- comm [OPTION] FILE1 FILE2
- When FILE1 or FILE2 (not both) is -, read standard input.
- With no options, produce three-column output.
- -1: suppress column 1 (lines unique to FILE1)
- -2: suppress column 2 (lines unique to FILE2)
- -3: suppress column 3 (lines that appear in both files)

```
cat assign2.html |
tr -cs 'A-Za-z' '[\n*]' |
tr '[:upper:]' '[:lower:]' |
sort -u |
comm -23 - words > misspelledEnglish.txt
```

*Buildwords.sh*

```
#! /bin/bash

# select <td> tags
grep -o '<td>.*</td>' |

# remove all the tags
sed 's/<[^>]*>//g' |

# squeeze new lines
tr -s '\n' |

# select even lines
sed -n '0~2p' |

# replace , with new line
sed 's/,/\n/g' |

# replace a space with new line
sed 's/ /\n/g' |

# replace ` with '
sed "s/\`/\'/g" |

# translate uppercase into lowercase
tr '[:upper:]' '[:lower:]' |
```

```
# remove non-Hawaiian words
sed /[^pk\'mnwlhaeiou]/d |

# squeeze new lines
tr -s '\n' |

# sort with unique option
sort -u
```

*Find-missing-head.sh*

```
#! /bin/bash

# prepend . or ./ to input if necessary
# this makes the names in current directory identical
# this also takes care of input with leading special character
add_dot_slash() {
   for input in $@
   do

   # check first character of input
   if [ ${input:0:1} = '.' ]; then
         echo "$input"
      elif [ ${input:0:1} = '/' ]; then
      echo ".$input"
   else
         echo "./$input"
      fi

   done
}

# return 0 (true) if arg1 is utf8 file
# return 1 (false) if arg1 is not utf8 file
is_utf() {

   # check if arg1 contains null byte
   num_lines=$( sed -n "/\x00/p" $1 | wc -l )

   # -s to check if arg1 file content is not empty
   if ([ $num_lines -eq 0 ] && [ -s $1 ] ); then

   # check if arg1 file contains non-asc character
```

```
    # check for match
    file=$( grep -Pl "[^\x00-\x7F]" $1 )
    if [ ! -z $file ]; then
        return 0
    fi
    fi

    # arg1 is not asc
    return 1
}

# takes files or directories as arguments
# return only utf files
# if an argument is a directory, it will recursively
# go through the files in subdirectories
find_utf() {
    for input in $@
    do

    # check if input is directory
    if [ -d $input ]; then

            files=$( find $input -type f )

        # recursively check if files in the directory are asc
            for file in $files
            do

        if is_utf $file; then
            echo "$file"
        fi

            done

    else

        if is_utf $input; then
        echo "$input"
            fi

    fi

    done
```

```
}

# take files as input
# return files without utf8 header
find_files_without_utf_8_header() {
    for file in $@
    do

    # check the first line of the file
    first_line=$( cat $file | head -1 )

    if [ "$first_line" != "-*- coding: utf-8 -*-" ]; then
            echo "$file"
    fi

    done

}

# split the argument based on new line instead of space
OIFS="$IFS"
IFS=$'\n'

# disable * and ?
set -f

inputs=$( add_dot_slash $@ )

files=$( find_utf $inputs )

files=$( find_files_without_utf_8_header $files )

files=$( echo "$files" | sort -u )

# restore settings
IFS=$OIFS
set +f

if [ ! -z "$files" ]; then
    echo "$files"
fi
```

Week 3

```
#!/usr/bin/python

import random, sys
from optparse import OptionParser

class Shuf:
    def __init__(self):
        version_msg = "%prog 2.0"
        usage_msg = """%prog [OPTION]... FILE
  or: %prog -i LO-HI [OPTION]...
Write a random permutation of the input lines to standard output.

With no FILE, or when FILE is -, read standard input."""

        self.help_msg = "\nTry '{0} --help' for more information.".format(sys.argv[0])

        self.parser = OptionParser(version=version_msg, usage=usage_msg)

        self.parser.add_option("-i", "--input-range",
                    action="store", dest="input_range",
                    help="treat each number LO through HI as an input line")

        self.parser.add_option("-n", "--head-count",
                    action="store", dest="head_count",
                    help="output at most COUNT lines")

        self.parser.add_option("-r", "--repeat",
                    action="store_true", dest="repeat",
                    help="output lines can be repeated")

    def parse_arguments(self, arguments):
        """
        If same options are provided,
        it will be overridden by later option.
        """
        options, args = self.parser.parse_args(arguments)

        return options, args

    def get_lines(self, options, args):
        if options.input_range is not None:
```

```
        # read from input_range
        if len(args) != 0:
            sys.exit("{0}: extra operand '{1}' {2}".format(sys.argv[0], args[0], self.help_msg))
        try:
            lo, hi = options.input_range.split("-")
            assert(int(lo) <= int(hi))
            nums = range(int(lo), int(hi) + 1)
            lines = []
            for num in nums:
                lines.append(str(num) + '\n')
        except:
            sys.exit("{0}: invalid input range '{1}'".format(sys.argv[0], options.input_range))
    else:
        # read from file or stdin
        if len(args) > 1:
            sys.exit("{0}: extra operand '{1}' {2}".format(sys.argv[0], args[1], self.help_msg))
        if len(args) == 0 or args[0] == "-":
            # read from stdin
            lines = sys.stdin.readlines()
        else:
            # read from file
            try:
                input_file = args[0]
                f = open(input_file, 'r')
                lines = f.readlines()
                f.close()
            except IOError as err:
                sys.exit("{0}: I/O error({1}) {2} {3}".format(sys.argv[0], err.errno, args[0], err.strerror))

    return lines

def write(self, lines, options):
    # shuffle the content
    random.shuffle(lines)

    # check head count
    if options.head_count is not None:
        try:
            head_count=int(options.head_count)
            assert(head_count >= 0)
        except:
            sys.exit("{0}: invalid head count '{1}'".format(sys.argv[0], options.head_count))
```

```
        if options.repeat is None:
            # no repeat
            num_iterations = len(lines)
            if options.head_count is not None and head_count < num_iterations:
                num_iterations = head_count
            for index in range(0, num_iterations):
                sys.stdout.write(str(lines[index]))
        else:
            # repeat
            if len(lines) == 0:
                sys.exit("{0}: no lines to repeat".format(sys.argv[0]))

            if options.head_count is not None:
                # repeat with head count
                for _ in range(0, head_count):
                    sys.stdout.write(str(random.choice(lines)))
            else:
                # repeat forever
                while (True):
                    sys.stdout.write(str(random.choice(lines)))


def main():
    shuf = Shuf()

    options, args = shuf.parse_arguments(sys.argv[1:])

    lines = shuf.get_lines(options, args)

    shuf.write(lines, options)


if __name__ == "__main__":
    main()
```

```c
#include <stdio.h>
#include <stdlib.h>

int frobcmp(const void *c1, const void *c2) {
  const char *word1 = *(const char**)c1;
  const char *word2 = *(const char**)c2;
  int i = 0;

  while (word1[i] != ' ' && word2[i] != ' ') {
    if (word1[i] == word2[i]) {
      i++;
    }
    else {
      return ((word1[i] ^ 42) > (word2[i] ^ 42)) ? 1 : -1;
    }
  }

  if (word1[i] == ' ' && word2[i] == ' ') {
    return 0;
  }

  if (word1[i] == ' ' && word2[i] != ' ') {
    return -1;
  }

  return 1;
}

void freeData(char **data, int row) {
  // free memory allocation
  int i;
  for (i = 0; i < row; i++) {
    free(data[i]);
  }
  free(data);
}

int isIOError() {
  if (ferror(stdin) || ferror(stdout)) {
    fprintf(stderr, "IO Error\n");
    return 1;
  }
```

```
  return 0;
}

int isNULLPtr(void *ptr) {
  if (ptr == NULL) {
    fprintf(stderr, "Memory Allocation Error\n");
    return 1;
  }

  return 0;
}

void printData(char **data, int row) {
  int i;
  for (i = 0; i < row; i++) {
    int j = 0;
    while (data[i][j] != ' ') {
      printf("%c", data[i][j++]);
      if (isIOError()) {
        freeData(data, row);
        exit(1);
      }
    }

    printf(" ");
    if (isIOError()) {
      freeData(data, row);
      exit(1);
    }
  }

}

int main() {
  char **data;
  char c;

  data = (char**)malloc(sizeof(char*));
  if (isNULLPtr(data)) {
    exit(1);
  }
```

```
data[0] = (char*)malloc(sizeof(char));
if (isNULLPtr(data[0])) {
  free(data);
  exit(1);
}

int row = 0; // index
int col = 0; // index

c = getchar();
if (isIOError()) {
  freeData(data, row + 1);
  exit(1);
}

while (!feof(stdin)) {

  if (c == ' ') {
    data[row++][col] = c;
    col = 0;
    char **tmp = data;
    data = (char**)realloc(data, (row+1)*sizeof(char*));
    if (isNULLPtr(data)) {
      freeData(tmp, row + 1);
      exit(1);
    }

    data[row] = (char*)malloc(sizeof(char));
    if (isNULLPtr(data[row])) {
      freeData(data, row + 1);
      exit(1);
    }
  }
  else {
    data[row][col++] = c;
    char *tmp = data[row];
    data[row] = (char*)realloc(data[row], (col+1)*sizeof(char));
    if (isNULLPtr(data[row])) {
      free(tmp);
      freeData(data, row + 1);
      exit(1);
    }
  }
```

```
    c = getchar();
    if (isIOError()) {
      freeData(data, row + 1);
      exit(1);
    }

  }

  data[row][col] = ' ';


  // sort
  qsort(data, row + 1, sizeof(char*), frobcmp);

  // print
  printData(data, row + 1);

  // free data
  freeData(data, row + 1);

  return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char **argv) {
  int i, j;
  char c;
  char *from;
  char *to;

  if (argc < 3) {
    fprintf(stderr, "%s\n", "missing operands");
    exit(1);
  }

  if (argc > 3) {
    fprintf(stderr, "%s\n", "extra operands");
    exit(1);
  }

  from = argv[1];
  to = argv[2];

  if (strlen(from) != strlen(to)) {
    fprintf(stderr, "%s\n", "two operands must have same length");
    exit(1);
  }

  for (i = 0; i < strlen(from); i++) {
    for (j = i + 1; j < strlen(from); j++) {
      if (from[i] == from[j]) {
        fprintf(stderr, "%s\n", "first operand may not contain any duplicate byte");
        exit(1);
      }
    }
  }

  c = getchar();
  if (ferror(stdin)) {
    fprintf(stderr, "%s\n", "stdin error");
    exit(1);
  }
```

```
  while (!feof(stdin)) {

    for (i = 0; i < strlen(from); i++) {
      if (c == from[i]) {
        c = to[i];
        break;
      }
    }

    putchar(c);
    if (ferror(stdout)) {
      fprintf(stderr, "%s\n", "stdout error");
      exit(1);
    }

    c = getchar();
    if (ferror(stdin)) {
      fprintf(stderr, "%s\n", "stdin error");
      exit(1);
    }
  }

  return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

int main(int argc, char **argv) {
  int i, j;
  char buf[1];
  char *from;
  char *to;

  if (argc < 3) {
```

```
  fprintf(stderr, "%s\n", "missing operands");
  exit(1);
}

if (argc > 3) {
  fprintf(stderr, "%s\n", "extra operands");
  exit(1);
}

from = argv[1];
to = argv[2];

if (strlen(from) != strlen(to)) {
  fprintf(stderr, "%s\n", "two operands must have same length");
  exit(1);
}

for (i = 0; i < strlen(from); i++) {
  for (j = i + 1; j < strlen(from); j++) {
    if (from[i] == from[j]) {
      fprintf(stderr, "%s\n", "first operand may not contain any duplicate byte");
      exit(1);
    }
  }
}

ssize_t flag;

flag = read(0, buf, 1);
if (flag == -1) {
  fprintf(stderr, "%s\n", errno);
  exit(1);
}

while (flag != -1 && flag != 0) {

  for (i = 0; i < strlen(from); i++) {
    if (buf[0] == from[i]) {
      buf[0] = to[i];
      break;
    }
  }
```

```
    flag = write(1, buf, 1);
    if (flag == -1) {
      fprintf(stderr, "%s\n", errno);
      exit(1);
    }

    flag = read(0, buf, 1);
    if (flag == -1) {
      fprintf(stderr, "%s\n", errno);
      exit(1);
    }
  }

  return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <ctype.h>

int frobcmp(const void *c1, const void *c2) {
  const char *word1 = *(const char**)c1;
  const char *word2 = *(const char**)c2;
  int i = 0;

  while (word1[i] != ' ' && word2[i] != ' ') {
    if (word1[i] == word2[i]) {
      i++;
    }
    else {
      return ((word1[i] ^ 42) > (word2[i] ^ 42)) ? 1 : -1;
    }
  }

  if (word1[i] == ' ' && word2[i] == ' ') {
    return 0;
```

```
  }

  if (word1[i] == ' ' && word2[i] != ' ') {
    return -1;
  }

  return 1;
}

int frobcmpF(const void *c1, const void *c2) {
  const char *word1 = *(const char**)c1;
  const char *word2 = *(const char**)c2;
  int i = 0;

  while (word1[i] != ' ' && word2[i] != ' ') {
    if (word1[i] == word2[i]) {
      i++;
    }
    else {
      return (toupper((unsigned char)(word1[i] ^ 42)) > toupper((unsigned char)(word2[i] ^ 42))) ?
1 : -1;
    }
  }

  if (word1[i] == ' ' && word2[i] == ' ') {
    return 0;
  }

  if (word1[i] == ' ' && word2[i] != ' ') {
    return -1;
  }

  return 1;
}


void freeData(char **data, int row) {
  if (data == NULL) return;

  // free memory allocation
  int i;
  for (i = 0; i < row; i++) {
    free(data[i]);
```

```
  }
  free(data);
}

void freeInput(char **data, char *input) {
  if (input != NULL) {
    free(input);
  }
  if (data != NULL) {
    free(data);
  }
}

int isNULLPtr(void *ptr) {
  if (ptr == NULL) {
    fprintf(stderr, "Memory Allocation Error\n");
    return 1;
  }

  return 0;
}

int printData(char **data, int row) {
  int i;
  for (i = 0; i < row; i++) {
    int j = 0;
    while (data[i][j] != ' ') {
      int flag = write(1, &data[i][j++], 1);
      if (flag == -1) {
        fprintf(stderr, "%s\n", "write error");
        return -1;
      }
    }

    int flag = write(1, " ", 1);
    if (flag == -1) {
      fprintf(stderr, "%s\n", "write error");
      return -1;
    }
  }

  return 0;
```

```
}

int main(int argc, char **argv) {
  char **data = NULL;
  char *input = NULL;
  int i;
  int row;
  int ptr;
  int isFound;
  int f_option = 0;

  // statbuf contains information about input file
  struct stat statbuf;

  // ============
  // check option
  // ============
  int opt;
  while ((opt = getopt(argc, argv, "f")) != -1) {
    switch (opt) {
      case 'f':
        f_option = 1;
        break;
      case '?':
        fprintf(stderr, "%s -%c\n", "Unknown option", optopt);
        exit(1);
        break;
      default:
        break;
    }
  }

  if (fstat(0, &statbuf) == -1) {
    fprintf(stderr, "%s\n", "fstat error");
    exit(1);
  }

  // ============================
  // if input is a file
  // allocate enough memory
  // ============================
  if (S_ISREG(statbuf.st_mode)) {
```

```
// empty file
if (statbuf.st_size == 0) return 0;

input = (char*)malloc(statbuf.st_size*sizeof(char));
if (isNULLPtr(input)) {
  exit(1);
}

if (read(0, input, statbuf.st_size) == -1) {
  fprintf(stderr, "%s\n", "read error");
  free(input);
  exit(1);
}

// check number of spaces
row = 0;
for (i = 0; i < statbuf.st_size; i++) {
  if (input[i] == ' ') {
    row++;
  }
}
// EOF will be replaced by a space
row++;

data = (char**)malloc(row*sizeof(char*));
if (isNULLPtr(data)) {
  free(input);
  exit(1);
}

i = 0;
ptr = 0;
isFound = 0;

// assign addressed of words to data

while (ptr != row) {
  if (!isFound) {
    data[ptr++] = &input[i];
    isFound = 1;
    if (input[i] != ' ') i++;
  }
  else {
```

```
      if (input[i] == ' ') isFound = 0;
      i++;
    }
  }

  // replace EOF with space
  input[statbuf.st_size - 1] = ' ';

}
// ==============================
// if input is a not file
// allocate memory byte by byte
// ==============================
else {

  char buf[1];
  int flag = read(0, buf, 1);
  if (flag == -1) {
    fprintf(stderr, "%s\n", "read error");
    exit(1);
  }

  // empty stdin
  if (flag == 0) return 0;

  data = (char**)malloc(sizeof(char*));
  if (isNULLPtr(data)) {
    exit(1);
  }

  data[0] = (char*)malloc(sizeof(char));
  if (isNULLPtr(data[0])) {
    free(data);
    exit(1);
  }

  row = 1;
  int col = 1;

  while (flag > 0) {

    if (buf[0] == ' ') {
      data[row - 1][col - 1] = ' ';
```

```
    char **temp = data;
    data = (char**)realloc(data, (row + 1)*sizeof(char*));
    if (isNULLPtr(data)) {
      freeData(temp, row);
      exit(1);
    }

    row++;

    data[row - 1] = (char*)malloc(sizeof(char));
    if (isNULLPtr(data[row - 1])) {
      freeData(data, row);
      exit(1);
    }

    col = 1;

  }
  else {

    data[row - 1][col - 1] = buf[0];
    char *temp = data[row - 1];
    data[row - 1] = (char*)realloc(data[row - 1], (col + 1)*sizeof(char));
    if (isNULLPtr(data[row - 1])) {
      free(temp);
      freeData(data, row - 1);
      exit(1);
    }

    col++;

  }

  flag = read(0, buf, 1);
  if (flag == -1) {
    fprintf(stderr, "%s\n", "read error");
    freeData(data, row);
    exit(1);
  }

}

data[row - 1][col - 1] = ' ';
```

```
  }

  // sort
  (f_option) ? qsort(data, row, sizeof(char*), frobcmpF) : qsort(data, row, sizeof(char*), frobcmp);

  // print
  int flag = printData(data, row);
  if (flag == -1) {
    if (S_ISREG(statbuf.st_mode)) {
      freeInput(data, input);
    } else {
      freeData(data, row);
    }
    exit(1);
  }


  // free data
  if (S_ISREG(statbuf.st_mode)) {
    freeInput(data, input);
  } else {
    freeData(data, row);
  }

  return 0;
}
```

```
#! /bin/bash

# make sure the order of sort is standard
export LC_ALL='C'

# oct form of ascii
decoded="\0\1\2\3\4\5\6\7\10\11\12\13\14\15\
\16\17\20\21\22\23\24\25\26\27\30\31\32\
\33\34\35\36\37\40\41\42\43\44\45\46\47\50\
```

```
\51\52\53\54\55\56\57\60\61\62\63\64\
\65\66\67\70\71\72\73\74\75\76\77\100\101\
\102\103\104\105\106\107\110\111\112\113\114\115\116\
\117\120\121\122\123\124\125\126\127\130\131\132\133\
\134\135\136\137\140\141\142\143\144\145\146\147\150\
\151\152\153\154\155\156\157\160\161\162\163\164\165\
\166\167\170\171\172\173\174\175\176\177\200\201\202\
\203\204\205\206\207\210\211\212\213\214\215\216\217\
\220\221\222\223\224\225\226\227\230\231\232\233\234\
\235\236\237\240\241\242\243\244\245\246\247\250\251\
\252\253\254\255\256\257\260\261\262\263\264\265\266\
\267\270\271\272\273\274\275\276\277\300\301\302\303\
\304\305\306\307\310\311\312\313\314\315\316\317\320\
\321\322\323\324\325\326\327\330\331\332\333\334\335\
\336\337\340\341\342\343\344\345\346\347\350\351\352\
\353\354\355\356\357\360\361\362\363\364\365\366\367\
\370\371\372\373\374\375\376\377"

# oct form of ascii after XOR 42
encoded="\52\53\50\51\56\57\54\55\42\43\40\41\46\47\44\
\45\72\73\70\71\76\77\74\75\62\63\60\
\61\66\67\64\65\12\13\10\11\16\17\14\
\15\2\3\0\1\6\7\4\5\32\33\30\31\36\
\37\34\35\22\23\20\21\26\27\24\25\152\153\
\150\151\156\157\154\155\142\143\140\141\146\147\144\
\145\172\173\170\171\176\177\174\175\162\163\160\161\
\166\167\164\165\112\113\110\111\116\117\114\115\102\
\103\100\101\106\107\104\105\132\133\130\131\136\137\
\134\135\122\123\120\121\126\127\124\125\252\253\250\
\251\256\257\254\255\242\243\240\241\246\247\244\245\
\272\273\270\271\276\277\274\275\262\263\260\261\266\
\267\264\265\212\213\210\211\216\217\214\215\202\203\
\200\201\206\207\204\205\232\233\230\231\236\237\234\
\235\222\223\220\221\226\227\224\225\352\353\350\351\
\356\357\354\355\342\343\340\341\346\347\344\345\372\
\373\370\371\376\377\374\375\362\363\360\361\366\367\
\364\365\312\313\310\311\316\317\314\315\302\303\300\
\301\306\307\304\305\332\333\330\331\336\337\334\335\
\322\323\320\321\326\327\324\325"


if [ "$#" -gt 1 ]; then
        echo "Extra operands"
```

```
elif [ "$#" -eq 1 ]; then
        # one option
        if [ "$1" == "-f" ]; then
                # -f option
                tr "$encoded" "$decoded" | sort -f | tr "$decoded" "$encoded"
        else
                # invalud option
                echo "Invalid option"
        fi
else
        # no option
        tr "$encoded" "$decoded" | sort | tr "$decoded" "$encoded"
fi
```

```
void* computePixels(void* info_ptr) {
   struct thread_info info = *(struct thread_info*)info_ptr;
   int num_threads = info.num_threads;
   int thread_index = info.thread_index;

...

   pthread_exit(0);
}

int
main( int argc, char **argv )
{
   int nthreads = argc == 2 ? atoi( argv[1] ) : 0;

   if( nthreads < 1 )
   {
     fprintf( stderr, "%s: usage: %s NTHREADS\n", argv[0], argv[0] );
     return 1;
   }

   scene = create_sphereflake_scene( sphereflake_recursion );

   /* Write the image format header */
   /* P3 is an ASCII-formatted, color, PPM file */
   printf( "P3\n%d %d\n%d\n", width, height, max_color );
   printf( "# Rendering scene with %d spheres and %d lights\n",
```

```
        scene.sphere_count,
        scene.light_count );

pthread_t thread_ids[nthreads];
struct thread_info info[nthreads];
for (int i = 0; i < nthreads; i++) {

    info[i].num_threads = nthreads;
    info[i].thread_index = i;

    int status = pthread_create(&thread_ids[i], NULL, computePixels, &info[i]);
    if (status) {
        fprintf(stderr, "%s\n", "error creating a thread");
        exit(-1);
    }
}

for (int i = 0; i < nthreads; i++) {
    int status = pthread_join(thread_ids[i], NULL);
    if (status) {
        fprintf(stderr, "%s\n", "error joining a thread");
        exit(-1);
    }
}

for (int w = 0; w < width; w++) {
    for (int h = 0; h < height; h++) {
        printf( "%.0f %.0f %.0f\n",
        pixels[w][h][0], pixels[w][h][1], pixels[w][h][2] );
    }
    printf("\n");
}


free_scene( &scene );

if( ferror( stdout ) || fclose( stdout ) != 0 )
{
    fprintf( stderr, "Output error\n" );
        return 1;
}

return 0;
```

```
}
```

```
randmain: randmain.c randcpuid.c randlibhw.so randlibsw.so
	$(CC) $(CFLAGS) -ldl -Wl,-rpath=$PWD randmain.c randcpuid.c -o $@

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

randlibsw.so: randlibsw.o
	$(CC) $(CFLAGS) -shared -o randlibsw.so randlibsw.o

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

randlibsw.o: randlibsw.c
	$(CC) $(CFLAGS) -fPIC -c randlibsw.c -o $@
```

Server Side
---
Command
---
ssh-keygen

---
Note
---
Create public and private keys on server side.
The client needs to store server's public key.
Later on, client will use it for server authentication.

---
Command
---
sudo useradd -d /home/jun -m jun
sudo passwd jun

---
Note
---
register an user "jun"

---
Command
---
cd /home/jun
sudo mkdir .ssh

---
Note
---
store anything related to ssh command

---
Command
---
sudo chown -R jun .ssh

---
Note
---

change owner of .ssh to jun

---
Command
---
sudo chmod 700 .ssh

---
Note
---
allow read and write for owner but no read nor write for group and others

---
Command
---
ifconfig

---
Note
---
Check server's IP address (it's under wlan0).
Use lo if the device is offline.

Client Side
---
Command
---
ssh-keygen

---
Note
---
Generate public and private keys, this will ask for passphrase.
A passphrase will be used to encrypt and decrypt the client's private key.

---
Command
---
ssh-copy-id -i jun@10.97.85.83

---
Note

---
Copy client's public key to server.

---
Command
---
ssh -X jun@10.97.85.83

---
Note
---
Login as jun, this will ask for the passphrase
because it needs to use private key to respond to
server's user authentication.

---
Command
---
eval `ssh-agent`

---
Note
---
`ssh-agent` returns a set of commands for remembering a passphrase.
This will start the process and return the process id.

---
Command
--------
ssh-add

------
Note
------
type the passphrase.
Let ssh-agent remember the passphrase
so that we don't need to enter passphrase every time.

---
Command
---
ssh -X jun@10.97.85.83

---
Note
---
login as jun again. This will no longer ask for the passphrase

---
Command
---
cd diffutils/
git log > ~/git-log.txt

---
Note
---
git log - Shows the commit logs.

---
Command
---
git tag > ~/git-tags.txt

---
Note
---
git-tag - Create, list, delete or verify a tag object signed with GPG

---
Command
---
git show 62ca21c8c1a5aa3488589dcb191a4ef04ae9ed4f > ~/quote-patch.txt

---
Note
---
git-show - Show various types of objects
For commits it shows the log message and textual diff

---
Command
---
patch -p1 < ~/quote-patch.txt

---
Note
---

C-x v = : displays a diff which compares each work file in the current VC fileset to the version(s) from which you started editing

C-x v u : Revert the work file(s) in the current VC fileset to the last revision

C-c C-a : Apply this hunk to its target file

C-c C-c : Go to the source file and line corresponding to this hunk

---
Command
---
emacs NEWS
C-x v u
yes
C-x C-c

---
Note
---
revert NEWS

---
Command
---
emacs src/analyze.c
C-x v =

C-u C-c C-a

C-x C-c
y

---
Note
---
C-u C-c C-a : undone specific hunk

undone the hunk of changes in comments

add-change-log-entry-other-window (C-x 4 a)

---
Command
---
git add .
git commit -F ChangeLog

---
Command
---
git format-patch -1 --stdout > ~/formatted-patch.txt

---
Command
---
git checkout v3.0 -b partner

---
Command
---
git am ~/renee-formatted-patch.txt