LAB 1

1. What shell command uses the man program to print all the commands that have a specific word in their man page (or at least the description part of the man page)? (hint: man man)
2. Where are the mv and sh programs located in the file system? List any shell commands you used to answer this question.
3. What executable programs have names that are exactly two characters long and end in r, and what do they do? List any shell commands you used to answer this question.
4. When you execute the command named by the symbolic link /usr/bin/emacs, which file actually is executed? List any shell commands you used to answer this question.
5. What is the version number of the /usr/bin/gcc program? of the plain gcc program? Why are they different programs?
6. The chmod program changes permissions on a file. What does the symbolic mode u+sx,o-w mean, in terms of permissions?
7. Use the find command to find all directories modified in the last four weeks that are located under (or are the same as) the directory /usr/local/cs. List any shell commands you used to answer this question.
8. Of the files in the same directory as find, how many of them are symbolic links? List any shell commands you used to answer this question.
9. What is the oldest regular file in the /usr/lib64 directory? Use the last-modified time to determine age. Specify the name of the file without the /usr/lib64/ prefix. Consider files whose names start with ".". List any shell commands you used to answer this question.
10. Where does the locale command get its data from? List any shell commands you used to answer this question.
11. In Emacs, what commands have downcase in their name? List any Emacs commands you used to answer this question.
12. Briefly, what do the Emacs keystrokes C-M-r through C-M-v do? Can you list their actions concisely? List any Emacs commands you used to answer this question.
13. In more detail, what does the Emacs keystroke C-g do? List any Emacs commands you used to answer this question.
14. What does the Emacs yank function do, and how can you easily invoke it using keystrokes? List any Emacs commands you used to answer this question.
15. When looking at the directory /usr/bin, what's the difference between the output of the ls -l command, and the directory listing of the Emacs dired command? List any shell or Emacs commands you used to answer this question.

1.
man -k [expression]
SHELL COMMANDS:
man man

2.
/usr/bin/mv
/usr/bin/sh

SHELL COMMANDS:
man which
which mv
which sh

3.
pr: convert text files for printing
tr: translate or delete characters

SHELL COMMANDS:
man find
find . -name '?r'
man tr
man pr

4.

/etc/alternatives/emacs

SHELL COMMANDS:
cd /usr/bin
ls -l emacs

5.

gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36)
gcc (GCC) 8.2.0

Different gcc programs are able to exist in different paths. The gcc program stored and used in the path /usr/bin has not been updated (4.8.5) whereas the plain gcc program is the one default to the operating system and is the most recent version (8.2.0).

SHELL COMMANDS:
/usr/bin/gcc --version
/usr/local/cs/bin/gcc --version

6.

u+sx,o-w sets the user ID on execution, allows the user to execute, and removes writing permissions from users not in the file group

SHELL COMMANDS:
man chmod

7.

SHELL COMMANDS:
cd /usr/local/cs
man find
find . -type d -mtime -28

8.

253 symbolic links

SHELL COMMANDS:
whereis find
man find
man wc
find /usr/bin -type l | wc -l

9.

librom1394.so.0.3.0

SHELL COMMANDS:
man ls
ls /usr/lib64 -lta

10.

The locale command displays information about the current locale, or all locales, on standard output.

SHELL COMMANDS:
man locale

11.

downcase-dwim
downcase-region
downcase-word

SHELL COMMANDS:
emacs
C-h a downcase

12.

C-M-r : reverse regular expression search
C-M-s : regular expression search
C-M-t : transpose expressions by moving one expression past the next one
C-M-u : move up parenthesis structure
C-M-v : scroll down the next window

SHELL COMMANDS:
C-h k C-M-r
C-h k C-M-s
C-h k C-M-t
C-h k C-M-u
C-h k C-M-v

13.

C-g : quits current command

SHELL COMMANDS:
C-h k C-g

14.

The yank function reinserts, or pastes, the last stretch of killed text.

Keystroke: C-y

SHELL COMMANDS:
emacs
C-h a yank
C-M-v

15.

ls -l : shows long listing including permission info
dired (C-x d) : "Edit" directory DIRNAME--delete, rename, print, etc. some files in it

The difference between the two commands is that ls -l ignores files beginning with '.' therefore more files are included when C-x d is run. Morever, the dired command allows the user to edit the director by actions like deleting, renaming, printing, etc. some files in it. The dired command also shows total space used and total space available in the directory whereas ls -l does not include these details in its output.

SHELL COMMANDS:
cd /usr/bin
ls -l
emacs
C-h a dired
C-x d

LAB 2

Create sorted list of English words, run following commands, tr …, Hawaiian language now, create hwords
Remove all instances of '?', '<u>' and '</u>'. For each remaining line of the form 'A<tdX>W</td>Z', where A and Z are zero or more spaces, X contains no '>' characters and W consists of entirely Hawaiian characters or spaces, assume that W contains zero or more nonempty Hawaiian words and extract those words; each word is a maximal sequence of one or more Hawaiian characters. Treat upper case letters as if they were lower case, and treat ` (ASCII grave accent) as if it were ' (ASCII apostrophe, which we use to represent 'okina). For example, the entry 'H<u>a</u>`ule lau' contains the two words 'ha'ule' and 'lau'. Sort the resulting list of words, removing any duplicates. Do not attempt to repair any remaining problems by hand; just use the systematic rules mentioned above.
Automate the systematic rules into a shell script buildwords

I first ran the shell command locale
which did NOT output LC_CTYPE="C".

I then ran the command export LC_ALL='C'
which sets the environment to the default locale.

I created a text file named words containing a sorted list of
the contents in the file /usr/share/dict/words by
running the following command:

```
$ sort /usr/share/dict/words >words
```

I created a text file containing the HTML of
the assignment's web page by running the following command:

```
$ wget https://web.cs.ucla.edu/classes/winter19/cs35L/assign/assign2.html
```

I ran the following command:

```
$ tr -c 'A-Za-z' '[\n*]' <assign2.html
```

This command takes the contents within the assign2.html file
and only outputs characters that are between A through
Z or a through z (alphabetic characters), printing new lines
for every character that isn't alphabetic. This is
done with -c which finds anything that is NOT A through Z or
a through z, replacing it with a newline.

I ran the following command:

```
$ tr -cs 'A-Za-z' '[\n*]' <assign2.html
```

This command does something similar to the previous where
only alphabetic characters are output, except reptitions
of newline between words are reduced to only one newline
between each word. This is done through -s which replaces
the input sequence of a repeated character with
a single occurence of the character.

I ran the following command:

```
$ tr -cs 'A-Za-z' '[\n*]' <assign2.html | sort
```

This command does what the last command does, except it
also alphabetically sorts the ouptut of characters that are
within range of A through Z or a through z found in the
assign2.html file. It does this by piping the resulting
output of the left side of the '|' character and sorting it
with the sort command on the right side.

I ran the following command:

```
$ tr -cs 'A-Za-z' '[\n*]' <assign2.html | sort -u
```

This command only outputs unique string values out of the
alphabetically sorted list of characters within range of
A-Z or a-z that was output by the previous command.
This is achieved by the -u specification which can also be
written as --unique which checks for strict ordering.

I ran the following command:

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

This command uses the comm command to compare the
alphabetically sorted output of the assign2.html file with the
alphabetically sorted list of strings in the words file,
producing a three-column output. The first column displays
strings unique to the sorted alphabetic characters of assign2.html,
the second column displays strings unique to
the words file, and the third column displays lines common
to both files (in this case the third column is empty
since the files lack any common lines).

I ran the following command:

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

This command suppresses the output of the second and
third columns that would nnormally display lines unique to the
words file and lines common to both assign2.html and words and
only outputs the first column which shows the lines
that are unique only to the alphabetically sorted ouptut
from assign2.html.

I ran the following command to download a copy of the
web page containing the simple Hawaiian dictionary:

```
$ wget http://mauimapp.com/moolelo/hwnwdseng.htm
```

I wrote a shell script buildwords.sh that
automated the systematic rules:

```
$ touch buildwords
```

```
#! /bin/bash

# buildwords
# automate systematic rules used to extract
# hawaiian words from table dictionary

# search for lines containing Hawaiian words
grep -E '<td>.+</td>' $@ |
sed '1~2d' |

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

# treat ASCII grave accent like ASCII apostrophe
sed "s/\`/\'/g" |

# treat upper case letters like lower case letters
tr [:upper:] [:lower:] |

# treat entries with spaces/commas as multiple words
tr ',' '\n' | tr ' ' '\n' |

# delete all leftover white spaces
sed '/^$/d' |

# check for non-Hawaiian letters
sed "/[^pk/'mnwlhaeiou]/d" |

# sort list of words & remove duplicates
sort -u
```

I ran the script on the web page to create the
hawaiian dictionary file (first giving it executable permission):

```
$ chmod +x buildwords
$ cat hwnwdseng.htm | ./buildwords | less > hwords
```

To check my work, I ran a Hawaiian spell checker on
the Hawaiian dictionary file itself (hwords) first:

```
$ tr [:upper:] [:lower:] <hwords | tr -cs "pk\'mnwlhaeiou" '[\n*]' |
  sort -u | comm -23 - hwords | wc -w

OUTPUT: 0
```

I then ran the Hawaiian spell checker on the assignment's web page:

```
$ tr [:upper:] [:lower:] <assign2.html | tr -cs "pk\'mnwlhaeiou" '[\n*]' |
  sort -u | comm -23 - hwords > hmispelled
$ wc -w <hmispelled

OUTPUT: 216
```

I also ran an English spell checker on the
assignment's web page as done earlier:

```
$ tr -cs 'A-Za-z' '[\n*]' <assign2.html | sort -u |
  comm -23 - words > emispelled
$ wc -w <emispelled

OUTPUT: 87
```

Examples of words that are "misspelled" as English, but not as Hawaiian:
zero
wiki
lau

Examples of words that are misspelled as Hawaiian but not as English:
ample
plea
open

1. A file name component must contain only ASCII letters, ‘.’, ‘-’, and ‘_’. A *file name component* is a nonempty part of a file name that is a maximal sequence of characters other than ‘/’; for example, the file name ‘/usr/lib64/libstdc++.so.6’ has the three components ‘usr’, ‘lib64’, and ‘libstdc++.so.6’, and the last component’s two ‘+’ characters violate this guideline.
2. A file name component cannot start with ‘-’.
3. Except for ‘.’ and ‘..’, a file name component cannot start with ‘.’.
4. A file name component must not contain more than 14 characters.
5. No two files in the same directory can have names that differ only in case. For example, if a directory contains a file ‘St._Andrews’ it cannot also contain a file name ‘st._anDrEWS’.

Basic poornames
In its basic form, when given one operand D, poornames should look for violations in D’s directory entries (i.e., in files immediately under D). If D is not the name of a directory, poornames should diagnose the problem on stderr and exit with a failure status. (If D is a symbolic link, poornames should not follow the symbolic link, and should treat it as an error just like any other non-directory.) When given no operands, poornames should act as if D is ‘.’, the current working directory. When given two or more operands, or a single operand whose first character is ‘-’, poornames should diagnose the problem on stderr and exit with a failure status. If poornames encounters an error when examining a directory (e.g., lack of permissions to read the directory) it should diagnose the problem on stderr, but it need not exit with a failure status.

The poornames script should output a line containing each full file name if and only if the file name’s last component does not follow the guidelines. The full file name should start with D, followed by ‘/’ if D does not already end in ‘/’, followed by the file name component that does not follow the guidelines, and finally followed by a trailing ‘/’ if the named file is a directory. The poornames script should not output duplicate lines. The order of output lines does not matter.

The poornames script should work with file names whose components contain special characters like spaces, ‘*’, and leading ‘-’ (except that D itself cannot begin with ‘-’). However, you need not worry about file names containing newlines.

Your script should be runnable as an ordinary user, and should be portable to any system that supports standard POSIX shell and utilities; please see its list of utilities for the commands that your script may use. (Hint: see the find and sort utilities.)

Recursive poornames
Once you’ve implemented the basic form, extend poornames so that it optionally runs recursively. When given an ‘-r’ option-argument, recursive poornames should look recursively at every directory entry in every directory under D, including D itself.

Any ‘-r’ option-argument must precede any D operand, and at most one option-argument (i.e., argument beginning with ‘-’) can be given; otherwise poornames should report the problem on standard error and exit with a failure status.

Recursive poornames should not follow symbolic links when recursively looking for poorly-named files. However, it should check the symbolic links’ names, just as it checks the names of all other files.

If a subdirectory name violates the guidelines, poornames should report the name, but this should not affect whether names of files under the subdirectory are also reported.

Although basic ‘poornames -r’ should report an error and exit with a failure status, recursive ‘poornames -r’ should behave like ‘poornames -r .’ and recurse through the working directory. The grader should be able to use use the behavior of ‘poornames -r’ to test whether you have attempted to do recursive poornames, or just basic poornames.

```
#!/bin/bash

function poor () {

    declare -a arr
    declare -a dupes
    let count=0
    let dupeCount=0

    if [ "$1" == "-r" ]
    then
            local dir=$2
    else
            local dir=$1
    fi

    if [ ! -d $dir ];
    then
            echo "Not a Directory" 1>&2
            exit 1
    fi

    if [ $# -ge 2 ] && [ "$1" != "-r" ];
    then
            echo "Too many arguments" 1>&2
            exit 1
    fi

    if [ $# -eq 0 ]
    then
            dir='.'
    fi

    if [ $# -eq 1 ] && [ "$1" == "-r" ]
    then
            dir='.'
    fi

    [[ $dir = -* ]] && echo "Directory cannot begin with '-'" && exit 1

    List2=`ls -a $dir | tr [:upper:] [:lower:]`
    List3=`ls -a $dir`
    List=`ls -a $dir |
              grep -E "[^-[:alpha:]._]"\|"^\.[^.]+"\|"^\.\..+"\|"^-"\|".{15,100}"`


    for file in $List
    do
            if [ -d "$dir/$file" ];
            then
               echo "$dir/$file/"
            else
               echo "$dir/$file"
            fi
    done


    for file1 in $List2
    do
            arr[$count]="$file1"
            let count=count+1
    done


    for (( i=0; i<$count; i++ ))
    do
            for (( j=i+1; j<$count; j++ ))
            do
               if [ ${arr[$i]} == ${arr[$j]} ]
               then
```

```
                dupes[$dupeCount]="$dir/${arr[$i]}"
                let dupeCount=dupeCount+1
            fi
        done
    done

    if [ $dupeCount -gt 0 ]
    then
        echo ${dupes[@]} | tr ' ' '\n' | sort -u
    fi

    if [ "$1" == "-r" ]
    then
        for file3 in $List3
        do
            if [ -d "$dir/$file3" ] && [ "$file3" != "." ] && [ "$file3" != ".." ]
            then
                poor -r "$dir/$file3"
            fi
        done
    fi

    return
}

poor $1 $2
:
```

LAB 3

1. Grab the Coreutils 8.29 source code compressed tarball and verify its signature with the GNU keyring by running the shell command gpg --verify --keyring ./gnu-keyring.gpg coreutils-8.29.tar.xz.sig in your directory. Note any problems with this verification, and briefly explain why they happen.
2. Compile and install your copy of Coreutils into a temporary directory of your own. Note any problems you run into.
3. Reproduce the bug on your machine with the unmodified version of coreutils. The test case that comes with the patch should give you ideas for how to reproduce the bug.
4. Apply the patch of Bug#30963 Message #10. Note any problems that occur (they should not prevent the modified coreutils from working).
5. Type the command make at the top level of your source tree, so that you build the fixed version, without installing it into your temporary directory. For each command that gets executed, explain why it needed to be executed (or say that it wasn't neeeded).
6. Make sure your change fixes the bug, by testing that the modified ls works on the same tests where the unmodified ls didn't work.

(STEP 1)

I first downloaded the Coreutils 8.29 source code and
its signature into my home directory:

```
$ wget ftp://ftp.gnu.org/gnu/coreutils/coreutils-8.29.tar.xz
$ wget ftp://ftp.gnu.org/gnu/coreutils/coreutils-8.29.tar.xz.sig
```

I downloaded the GNU keyring so that I could then verify
the signature. The gpg command I use is an encryption and signing tool
that uses the OpenPGP standard:

```
$ wget https://ftp.gnu.org/gnu/gnu-keyring.gpg
$ gpg --verify --keyring ./gnu-keyring.gpg coreutils-8.29.tar.xz.sig
```

A warning message printed that states:

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
```

This warning should be normal since it clearly states that
there is no proof that the signature belongs to the owner.

(STEP 2)

Once I verified the signature, I untarred and unzipped the tarball:

```
$ tar -xJvf coreutils-8.29.tar.xz
```

I created a directory named coreutilsInstall in my home
directory in which I will install coreutils:

```
$ mkdir ~/coreutilsInstall
```

I navigated into the coreutils-8.29 directory and read the INSTALL file
on how to configure "make" and ran the configure script using the
prefix flag so that coreutils would be installed in the directory
I detailed above (~/coreutilsInstall):

```
$ cd coreutils-8.29
$ emacs INSTALL
$ ./configure --prefix=/u/ee/ugrad/kathy/coreutilsInstall
```

Next, I compiled and installed coreutils. The make command
compiles the program and creates exectuable files while the
make install command installs coreutils into ~/coreutilsInstall:

```
$ make
$ make install
```

I didn't run into any problems during this step.

(STEP 3)

I then reproduced the bug on my machine using the unmodified
version of coreutils. Justing running 'ls' won't produce the
results I want because the shell looks for /bin/ls, so I used
./bin/ls instead to manually run the exectuable in the appropiate
directory:

```
$ cd ~/coreutilsInstall
$ ./bin/ls -a -A
```

The result was the expected buggy output which produced four lines
instead of two lines:

```
. .. bin libexec share
```

(STEP 4)

To apply the patch of Bug #30963 Message #10, I created a file named
bug_patch and copied over the patch code:

```
$ cd ../coreutils-8.29
$ emacs patch_file
(Copied and pasted the patch)
$ man patch
$ patch -p1 < patch_file
```

I used the -p1 flag to modify the file paths in the patch code
since I did not have directories 'a' and 'b' although they were
referred to in the patch file.

The only error I recieved was as follows:

```
1 out of 1 hunk FAILED -- saving rejects to file NEWS.rej
```

However, the message also said that src/ls.c was successfully patched so I checked that the patch was actually applied:

```
$ emacs src/ls.c
  X-s case 'A'
```

Under case 'A', it said 'ignore_mode = IGNORE_DOT_AND_DOTDOT;'
So the patch was applied!

(STEP 5)

Now I can rebuild ls.c using the make command.
This compiles the program code and creates new exectuables
in the current directory:

```
$ make
```

I did not run make install because it would copy the executables
into the temporary directory I created earlier ~/coreutilsInstall
which the assignment mentioned to avoid.

(STEP 6)

To make sure that my changes actually fixed the bug, I tested
it out in the src file containing ls.c:

```
$ cd src
$ ./ls -a -A
```

It looks like . and .. were not printed, so the patch was successful!

PYTHON 3
Your program should support the following shuf options, with the same behavior as GNU shuf: --input-range (-i), --head-count (-n), --repeat (-r), and --help. As with GNU shuf, if --repeat (-r) is used without --head-count (-n), your program should run forever. Your program should also support zero non-option arguments or a single non-option argument "-" (either of which means read from standard input), or a single non-option argument other than "-" (which specifies the input file name). Your program need not support the other options of GNU shuf. As with GNU shuf, your program should report an error if given invalid arguments.

Q1. Does the patch improve the performance of ls or make it worse? Briefly explain.
Q2. If your company adopts this patched version of Coreutils instead of the default one, what else should you watch out for? Might this new version of Coreutils introduce other problems with your application?
Q3. What happens when this script is invoked on an empty file like /dev/null, and why?
Q4. What happens when this script is invoked with Python 3 rather than Python 2, and why? (You can run Python 3 on the SEASnet hosts by using the command python3 instead of python.)
Q5. What happens when your shuf.py script is invoked with Python 2 rather than Python 3, and why?

Q1: Does the patch improve the performance of ls or make it worse?

The patch removes an if statement from the ls.c file as seen
from the following section of the diff file:

```
        case 'A':
-         if (ignore_mode == IGNORE_DEFAULT)
-           ignore_mode = IGNORE_DOT_AND_DOTDOT;
+          ignore_mode = IGNORE_DOT_AND_DOTDOT;
          break;
```

This speeds up execution time because there are now less instructions for the computer to execute. The patch also ensures that the -a option doesn't always override the -A option anymore, improving performance regarding that specific behavior.

Q2: If your company adopts thispatched version of Coreutils,
what else should you watch out for? Might this new version
introduce other problems with your application?

We must be careful to distinguish between the order of flags. In other words, paying special attention to the different between -a -A and -A -a. After applying the patch, they do different things unlike previously when it didn't matter what order they were typed.

Q3: What happens when this script is invoked on an empty file
like /dev/null and why?

I recieved the following error when running the script on /dev/null:

```
IndexError: list index out of range
```

This is because the script attempts the following line of code:

```
return seq[int(self.random() * len(seq))]
```

It raises an error because seq[0] is called on an empty file so it falls out of bounds since there are no lines in the file to select. However, if I were to invoke the script with an argument of 0, no error would occur since no lines are attempted to be read.

Q4: What happens when this script is invoked with Python 3 rather
than Python 2, and why?

When I invoke the script using python3 instead of python, I recieved the following error message:

```
File "randline.py", line 65
  except IOError as (errno, strerror):
                    ^
SyntaxError: invalid syntax
```

The exception provides a 2-tuple detailing a numerical error code, errno, along with a string explaining the error itself, strerror. Python 2 exceptions are iterable, meaning you can iterate over parameters passed to the exception. Python 3 does not support tuples in error statements which leads to the syntax error shown.

Q5. What happens when your shuf.py script is invoked with Python 2 rather than Python 3, and why?

It gives a syntax error because the exceptions are different

SHUF.PY

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

class shuf:

    def __init__(self, input_list, count, repeat):
        self.input_list = input_list
        self.repeat = repeat
        self.count = count
```

```
        # shuffle lines of input
        random.shuffle(self.input_list)

    def shuffleline(self):
        # do nothing if length of input is 0
        if len(self.input_list) == 0:
            return

        # if repeat option is used, pick randomly
        if self.repeat:
            while self.count > 0:
                sys.stdout.write(random.choice(self.input_list))
                self.count -= 1

        # If repeat option not used, pick in order
        if not self.repeat:
            for i in range(0, self.count):
                sys.stdout.write(self.input_list[i])


def main():
    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."""

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

    # -n, --head-count
    # output at most COUNT lines
    parser.add_option("-n", "--head-count",
                      action="store", dest="count", default=sys.maxsize,
                      help="output at most COUNT lines")

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

    # -r, --repeat
    # output lines can be repeated
    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:])

    # -n, --head-count
    try:
        count = int(options.count)
    except:
        parser.error("invalid COUNT: {0}".
                     format(options.count))
    if count < 0:
        parser.error("negative count {0}".
                     format(count))

    # -r, --repeat
    repeat = options.repeat

    # -i, --input-range=LO-HI
    inputRange = options.inputRange

    # if option i
    if len(inputRange) > 0:

        # -i option cannot take in arguments
        if len(args) != 0:
            parser.error("extra operand '{0}'".
                         format(args[0]))

        # input range cannot accept a single number
        try:
            dash = inputRange.index('-')
        except ValueError as e:
            parser.error("invalid input range: '{0}'".
                         format(options.inputRange))
        if dash == 0:
            parser.error("invalid input range: '{0}'".
                         format(options.inputRange))

        # check for integer values before/after '-'
        first, last = inputRange.split("-")
        try:
            firstNum = int(first)
        except ValueError as e:
            parser.error("invalid input range: '{0}'".
                         format(options.inputRange))
        try:
            lastNum = int(last)
        except ValueError as e:
            parser.error("invalid input range: '{0}'".
                         format(options.inputRange))

        # check that first value < last value of range
        if first > last:
            parser.error("invalid input range: '{0}'".
                         format(options.inputRange))

        # store number range into input
        input_list = list(range(firstNum,lastNum+1))

        # convert integer list into string list with newlines
        for i in range(len(input_list)):
            input_list[i] = str(input_list[i]) + "\n"

    else:
        # accept input from stdin if there are
        # either zero non-option arguments or "-"
        if len(args) == 0 or (len(args) == 1 and args[0] == "-"):
            input_list = sys.stdin.readlines()

        # accept input from a file
        elif len(args) == 1:
            # check that input file is valid
            try:
                f = open(args[0], 'r')
                input_list = f.readlines()
                f.close()
            except IOError as e:
                errno, strerror = e.args
                parser.error("I/O error({0}): {1}".
                             format(errno,strerror))

        # cannot accept more than one argument
        else:
            parser.error("extra operand '{0}'".
                         format(args[1]))

    # check if count is greater than input
    # (and repeat is not used)
    if count > len(input_list) and not repeat:
        count = len(input_list)

    generator = shuf(input_list, count, repeat)
    generator.shuffleline()

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

LAB 4

The bug is that ls -t mishandles files whose time stamps are very far in the past. It seems to act as if they are in the future. For example:

```
$ tmp=$(mktemp -d)
$ cd $tmp
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice-cs35L
$ touch now
$ sleep 1
$ touch now1
$ TZ=UTC0 ls -lt --full-time wwi-armistice-cs35L now now1
-rw-r--r-- 1 eggert csfac 0 1918-11-11 11:00:00.000000000 +0000 wwi-
armistice-cs35L
-rw-r--r-- 1 eggert csfac 0 2018-10-29 16:43:16.805404419 +0000 now1
-rw-r--r-- 1 eggert csfac 0 2018-10-29 16:43:15.801376773 +0000 now
$ cd
$ rm -fr $tmp
```

Build this old version of coreutils as-is, and then again with this renaming patch. What problems did you have when building it as-is, and why did the renaming patch fix them?

Reproduce the problem. Use a debugger to figure out what went wrong and to fix the corresponding source file.

Construct a new patch file lab4.diff containing your coreutils fixes, in the form of a ChangeLog entry followed by a diff -u patch.

Also, try to reproduce the problem in your home directory on the SEASnet Linux servers, instead of using the $tmp directory. When running the above test case, use the already-installed touch and ls utilities instead of the old version of coreutils. How well does SEASnet do?

I started by downloading the old version of coreutils with the buggy ls program and unzipping it:

```
$ wget https://web.cs.ucla.edu/classes/winter19/cs35L/
        assign/coreutils-with-bug.tar.gz
$ tar -xzvf coreutils-with-bug.tar.gz
```

Then I created a directory named coreutilsBuggy in my home directory in which I will install the buggy version of coreutils:

```
$ mkdir ~/coreutilsBuggy
```

I navigated into the coreutils-with-bug direcrory and ran the configure script using the prefix flag so it would be installed in the ~/coreutilsBuggy directory:

```
$ cd coreutils-with-bug
$ emacs INSTALL
$ ./configure --prefix=/u/ee/ugrad/kathy/coreutilsBuggy
```

When I ran 'make' and 'make install', I got the following error:

```
error: conflicting types for 'futimens'
int futimens (int, char const *, struct timespec const [2]);
    ^
```

As we can see, there were two different function declarations with the same name, which led to a compilation error.

I next downloaded and applied the renaming patch:

```
$ wget https://web.cs.ucla.edu/classes/winter19/cs35L/
      assign/coreutils.diff
$ man patch
$ patch -p0 < coreutils.diff
```

I used p0 to so that the file path would be unmodified in the patch code. I then ran the 'make' command again and it successfully executed:

```
$ make
```

The patch was able to fix the errors because it changed every instance of the futimens function to coreutils_futimens and every instance of the tee function to coreutils_tee. Doing this prevented the conflicting types error to occur:

```
-  return futimens (-1, file, timespec);
+  return coreutils_futimens (-1, file, timespec);

-static bool tee (int nfiles, const char **files);
+static bool coreutils_tee (int nfiles, const char **files);
```

To reproduce the problem, I navigated to my coreutilsBuggy directory and ran the following commands detailed in the spec:

```
$ tmp=$(mktemp -d)
$ cd $tmp
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ TZ=UTC0 ~/coreutils-with-bug/src/ls -lt --full-time wwi-armistice now
now1
```

The following output was produced:

```
-rw-r--r-- 1 kathy eeugrad 0 1918-11-11 11:00:00.000000000 +0000 wwi-
armistice
-rw-r--r-- 1 kathy eeugrad 0 2019-02-07 19:38:09.059162428 +0000 now1
-rw-r--r-- 1 kathy eeugrad 0 2019-02-07 19:38:03.997075516 +0000 now
```

As we can see, the bug was reproduced because the oldest file, wwi-armistice, was placed at the top, when it should be at the bottom.

To figure out what went wrong, I used gdb on the buggy ls program.

```
$ gdb ~/coreutils-with-bug/src/ls
```

Using 'info functions' to see a list of all defined functions used in ls.c, I identifed compare_mtime' as potential candidate and set a breakpoint there:

```
(gdb) info functions
(gdb) break compare_mtime
(gdb) r ~/$temp -lt
```

Once I hit the breakpoint, I saw that compare_mtime calls another function, cmp_mtime. I decided to step into the function:

```
(gdb) stepi
```

I saw that cmp_mtime also calls another function, timespec_cmp and decided to step into that too:

```
(gdb) stepi
(gdb) list 49

46  timespec_cmp (struct timespec a, struct timespec b)
47  {
48    int diff = a.tv_sec - b.tv_sec;
49    return diff ? diff : a.tv_nsec - b.tv_nsec;
50  }
```

As we can see from the contents of the function, it subtracts two time values and returns either a postiive or negative value based on what the value of 'diff' is. I continued the program and examined 'diff' for each of the 3 files (now, now1, wwi-armistice):

```
(gdb) info locals
diff = 28483830
(gdb) stepi
(gdb) stepi
(gdb) info locals
diff = -18480010
```

There seems to be a problem with diff at this point in the program. It is a very large negative number, but the difference between a file created today and a file created ~100 years ago should be positive. This is likely caused by overflow that occured during the subtraction. This issue makes sense since the amount of seconds in 100 years is too large a value to store into an integer, which can only hold up to a maximum value of 0x7FFFFFFF.

To fix this problem, we can avoid subtraction entirely. To do this, we can simply just compare the two values directly and return more straightfoward values such as -1 or 1 based on the result.

To fix the source file, I want to create a new patch file lab4.diff containing my coreutils fixes. I started by making a copy of the file that contained the definition of the timespec_cmp function. I found this while stepping through the debugger, in the following line:

```
timespec_cmp (b=..., a=...) at ../lib/timespec.h:49
```

As you can see, timestamp.h contains the definition. I proceeded with the following commands:

```
$ cd ~/coreutils-with-bug
$ cp lib/timespec.h timespec.h
$ emacs timespec.h
```

Once inside the header file, I modified the timespec_cmp function to have the functionality I described earlier, using comparisons to avoid overflow:

```
timespec_cmp (struct timespec a, struct timespec b)
{
  int diff = a.tv_nsec - b.tv_nsec;
  return (a.tv_sec > b.tv_sec) ? 1 : (a.tv_sec < b.tv_sec) ? -1 : diff;
}
```

As you can see, I modified the code such that it would return 1 or -1 depending on whether one timestamp was greater than the other. If they are evaluated as equal, I then return the difference in nanoseconds to be more precise since doing so would likely not encounter the same issue of overflow.

Now I can create the actual diff file using the diff command to compare changes between the original header file and the copy I modified. I added the -u flag so that the output text would be in a unified context.

```
$ touch lab4.diff
$ diff -u lib/timespec.h timespec.h > lab4.diff
```

I also made sure to change the file path in the second line from timespec.h to lib/timespec.h so that the correct file would be patched:

```
--- lib/timespec.h      2005-09-21 23:05:39.000000000 -0700
+++ lib/timespec.h      2019-02-07 15:45:17.201657000 -0800
```

I created a ChangeLog entry by navigating into the lab4.diff file and using 'C-x 4 a' and added the following description:

```
2019-02-09  Kathy Douangpanya Daniels  <kathy@lnxsrv09.seas.ucla.edu>

        * timespec.h (timespec_cmp): Replaced subtraction between
        integers with direct comparison to avoid overflow causing
        unwanted behavior in the program. Return 1 or -1 depending
        on whether one timestamp is larger than the other and
        return the difference in nanoseconds if the seconds are
        evaluated to be equal.
```

Now that I've created the patch, I can test it as I did before:

```
$ patch -p0 < lab4.diff
$ make
```

I re-ran the commands that were on the spec to see what how the output would turn out:

```
$ tmp=$(mktemp -d)
$ cd $tmp
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ TZ=UTC0 ~/coreutils-with-bug/src/ls -lt --full-time wwi-armistice now
now1
```

The output looks correct, so the patch worked!

```
-rw-r--r-- 1 kathy eeugrad 0 2019-02-08 01:06:10.019670444 +0000 now1
-rw-r--r-- 1 kathy eeugrad 0 2019-02-08 01:06:04.101569066 +0000 now
-rw-r--r-- 1 kathy eeugrad 0 1918-11-11 11:00:00.000000000 +0000 wwi-
armistice
```

Attempting to reproduce the problem in my home directory (on the SEASnet severs), I ran the following commands:

```
$ cd
$ mkdir tmp
$ cd tmp
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ TZ=UTC0 ls -lt --full-time wwi-armistice now now1
```

The following output was the result:

```
-rw-r--r-- 1 kathy eeugrad 0 2054-12-17 17:28:16.000000000 +0000 wwi-
armistice
-rw-r--r-- 1 kathy eeugrad 0 2019-02-08 01:13:31.408442000 +0000 now1
-rw-r--r-- 1 kathy eeugrad 0 2019-02-08 01:13:25.451889000 +0000 now
```

As we can see, the SEASnet servers have a bug! The timestamp for wwi-armistice displays as made in the year 2054 even though I set it to be 1918. The reason why this occurs is due to the Unix epoch, or Unix time; it keeps time as the number of seconds that have elapsed since January 1st, 1970. As a result of this, times earlier than the epoch are negative, the epoch is zero, and times later are positive.

Unix time is usually represented using 32-bit unsigned integers, so since it is unable to represent negative timestamps, or times earlier than the epoch (in this case 1918), it wraps around and leads to a positive, future time of 2054.

Use your C function to write a program sfrob that reads frobnicated text lines from standard input, and writes a sorted version to standard output in frobnicated form. Frobnicated text lines consist of a series of non-space bytes followed by a single space; the spaces represent newlines in the original text. Your program should do all the sorting work itself, by calling frobcmp. If standard input ends in a partial record that does not have a trailing space, your program should behave as if a space were appended to the input.

```
// sfrob.c - reads frobnicated text lines from standard input,
// and writes a sorted version to standard output in frobnicated form

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

// returns an int result that is negative, zero, or positive
// depending on whether a is less than, equal to, or greater than b
int frobcmp(char const * a, char const * b) {

  // check that each argument points to
  // non-space bytes
  while (*a != ' ' && *b != ' ') {

    // unfrobnicate byte by byte (XOR with 42)
    char a1 = *a ^ 42;
    char b1 = *b ^ 42;

    // if a1 is smaller than b1
    // OR
    // a is shorter than b
    // return -1
    if (a1 < b1 || *a == ' ') { return -1; }

    // if a1 is larger than b1
    // OR
    // b is shorter than a
    // return 1
    if (a1 > b1 || *b == ' ') { return 1; }

    // increment index in byte array
    a++;
    b++;
  }

  // if a and b are equal
  // return 0
  return 0;

}

// comparator function to pass into qsort in main
int cmp(const void * a, const void * b) {

  return frobcmp(*(char**) a, *(char**) b);

}

int main() {

  char newChar;

  // allocate memory for array of strings (char arrays)
  // check for memory error
  char** strArray = (char**) malloc(sizeof(char*));
  if (strArray == NULL) {
    fprintf(stderr, "Error allocating memory.");
    exit(1);
  }

  // index of string in strArray
  // set as -1 since s is incremented on first iteration
  int s = -1;

  // allocate memory for new string (or char array)
  // check for memory error
  char* newString = (char*) malloc(sizeof(char));
  if (newString == NULL) {
    fprintf(stderr, "Error allocating memory.");
    exit(1);
  }

  // index of char in newString
  int c = 0;

  // set temporary string to first index in string array
  strArray[0] = newString;

  // set to TRUE if program must create a new string
  // set to FALSE if program is adding to existing string
  bool addNewString = true;

  // continue collecting input until end of file is reached
  while (true) {

    // read text from standard input
    // print message to stderr if program has error reading input
    newChar = getchar();
    if (ferror(stdin)) {
      fprintf(stderr, "Error reading standard input.");
      exit(1);
    }

    // break from reading input if reaches end of file
    if (feof(stdin)) { break; }

    // if program is still adding characters to existing string
    if (!addNewString) {

      // reallocates memory for new character in growing string
      newString  = (char*) realloc(newString, (c+1) * sizeof(char));

      // checks for memory error
      if (newString == NULL) {
          fprintf(stderr, "Error allocating memory.");
          exit(1);
      }

      // updates growing string to reallocated string
      newString[c] = newChar;
      strArray[s] = newString;
      c++;

      // space byte signals the end of a text line
      // program should start on a new string
      if (newChar == ' ') {
          // ignore multiple spaces
          addNewString = true;
      }

    }

    // if program must create a new string
    else {

      // reset new string length
      c = 0;

      // account for multiple spaces in input
      if (newChar == ' ' && c == 0) { continue; };

      // increment index in string array
```

```
    s++;

    // reallocates memory for growing string array
    strArray = (char**) realloc(strArray, (s+1) * sizeof(char*));

    // checks for memory error
    if (strArray == NULL) {
            fprintf(stderr, "Error allocating memory.");
            exit(1);
    }

    // allocate memory for new string in array
    newString = (char*) malloc(sizeof(char));

    if (newString == NULL) {
            fprintf(stderr, "Error allocating memory.");
            exit(1);
    }

    // updates growing string to reallocated string
    newString[c] = newChar;
    strArray[s] = newString;
    c++;

    // program can start adding characters to this new string
    // since the appropiate memory has been allocated
    addNewString = false;

  }

}

// if standard input does not have a trailing space
// append space to the very end
if (s != -1 && strArray[s][c-1] != ' ') {

  newString = (char*) realloc(newString, (c+1) * sizeof(char));
  newString[c] = ' ';
  strArray[s] = newString;

}

// use frobcomp to sort array of strings
qsort(strArray, s+1, sizeof(char*), cmp);

// print every character in every string to stdout
int i;
for (i = 0; i < s+1; i++) {

  int j = 0;
  while(true) {

    // print character by character and catch any errors
    if (putchar(strArray[i][j]) == EOF) {
            fprintf(stderr, "Error printing characters.");
            exit(1);
    }

    // skip to the next string if space byte is detected
    if (strArray[i][j] == ' ') { break; }

    j++;

  }

  // free memory holding each string
  free(strArray[i]);
}

// free memory holding array of strings
free(strArray);

  exit(0);
}
```

LAB 5

1. Write a C transliteration program tr2b.c that uses getchar and putchar to transliterate bytes as described above.
2. Write a C program tr2u.c that uses read and write to transliterate bytes, instead of using getchar and putchar. The *nbyte* arguments to read and write should be 1, so that the program reads and writes single bytes at a time.
3. Use the strace command to compare the system calls issued by your tr2b and tr2u commands (a) when copying one file to another, and (b) when copying a file to your terminal. Use a file that contains at least 5,000,000 bytes.
4. Use the time command to measure how much faster one program is, compared to the other, when copying the same amount of data.

==========

lab.txt

==========

1.

I first wrote a C transliteration program tr2b.c that uses getchar and putchar to transliterate bytes. I checked for the correct number of arguments (3), matching string lengths, and for duplicates. The method I implemented used a character array (or a map) of size 256, mapping every character byte in 'from' to every corresponding character byte in 'to'.

When collecting input, I checked if the current character byte had a mapped value. If not, the program should output the original input. If yes, the program should output the mapped character found in the large character array.

2.

I wrote another C transliteration program tr2u.c that uses read and write instead of getchar and putchar to transliterate bytes. I still performed the same checks as previously mentioned, and also implemented it using the same character array. However, I read in and wrote out characters directly through system calls.

3.

I used the strace command to compare system calls issued by tr2b and tr2u commands. I created a file that was at least 5,000,000 bytes to test this:

```
$ head --bytes=5000000 /dev/urandom > testfile.txt
```

First, I compared the system calls when copying one file to another. To do so, I used the 'c' option as follows:

```
$ strace -c ./tr2b 'A' 'X' < testfile.txt > output1.txt
```

% time	seconds	usecs/call	calls	errors	syscall
27.27	0.000051	6	9		mmap
16.58	0.000031	8	4		mprotect
10.16	0.000019	5	4		fstat
9.09	0.000017	17	1		brk

```
  8.02    0.000015           8         2           open
  7.49    0.000014          14         1           write
  6.95    0.000013           7         2           read
  5.88    0.000011          11         1           munmap
  3.74    0.000007           7         1         1 access
  3.21    0.000006           3         2           close
  1.60    0.000003           3         1           arch_prctl
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    0.000187                    29         1 total
```

As you can see, there were 29 system calls for tr2b.

```
  $ strace -c ./tr2u 'A' 'X' < testfile.txt > output2.txt

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 55.11   21.234223           4   5000000           write
 44.89   17.294951           3   5000002           read
  0.00    0.000000           0         2           open
  0.00    0.000000           0         2           close
  0.00    0.000000           0         2           fstat
  0.00    0.000000           0         7           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         1           brk
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00   38.529174             10000024         1 total
```

We see here that there were 10000024 system calls for tr2u. tr2u clearly has a much larger amount of calls than tr2b since it requires many individual read and write calls for each of the 5,000,000 bytes.

I next compared the system calls when copying a file to terminal. I used the same commands but just didn't pipe it into a separate output file.

```
  $ strace -c ./tr2b 'A' 'X' < testfile.txt
  $ strace -c ./tr2u 'A' 'X' < testfile.txt
```

Again, I recieved 29 system calls for tr2b and 10000024 system calls for tr2u. We expect this since tr2b uses buffered I/O, so less system calls are needed to gather the same desired input. This also leads tr2b to be a much faster program, confirmed below.

4.

To compare the speed of each program, I used the time command. I first timed them in the context of copying one file to another:

```
  $ time ./tr2b 'A' 'X' < testfile.txt > output1.txt

  real      0m0.004s
  user      0m0.001s
  sys       0m0.000s

  $ time ./tr2u 'A' 'X' < testfile.txt > output2.txt

  real      0m10.164s
  user      0m1.266s
  sys       0m8.854s
```

I then timed them again in the context of copying a file to the terminal:

```
  $ time ./tr2b 'A' 'X' < testfile.txt

  real      0m0.002s
  user      0m0.001s
  sys       0m0.000s

  $ time ./tr2u 'A' 'X' < testfile.txt

  real      0m9.116s
  user      0m1.173s
  sys       0m7.932s
```

As we can see, tr2u is a lot slower than tr2b in copying the same files. tr2b barely took any time to finish while tr2u took up to 10.162 seconds. Again, this is what we would expect since tr2b uses buffered I/O so it makes less calls and reduces overhead while the unbuferred I/O implementation causes tr2u to spend a lot of extra time reading and writing every byte.

```
==========================================================
==========
                                    report.txt
==========================================================
==========
```

To estimate the number of comparisons as a function of input lines, I ran sfrobu on inputs of varying numbers of input lines including 10, 100, 1000 and 10000. I added a global variable to keep track of every instance frobcmp was called in the program and printed it at the end:

```
# of Lines  ||  # of Comparisons
================================
10          ||  175
100             ||  2880
1000        ||  40352
10000       ||  521345
```

If we represent the number of input lines as variable N, the number of comparison calls can be roughly estimated by:

```
  comparisons = 4*Nlog(N)

  40log(10) = ~133
  400log(100) = ~2660
  4000log(1000) = ~39960
  40000log(10000) = ~531500
```

The big O complexity of quicksort is also Nlog(N), so it makes sense here that our function closely matches Nlog(N) since our implementation utilizes quicksort.

I then used the time command to measure the difference in performance between sfrob, sfrobu, sfrobu -f, and sfrobs -f. I again tested the programs using the large file containing 5,000,000 bytes from earlier:

```
  $ time ./sfrob < testfile.txt > output.txt
  $ time ./sfrobu < testfile.txt > output.txt
  $ time ./sfrobu -f < testfile.txt > output.txt
  $ time ./sfrobs < testfile.txt > output.txt
  $ time ./sfrobs -f < testfile.txt > output.txt

----- sfrob -----

real  0m0.371s
user  0m0.306s
sys   0m0.015s

---- sfrobu -----

real 0m7.393s
```

```
user 0m0.714s
sys  0m6.636s

--- sfrobu -f ---

real  0m6.466s
user  0m0.669s
sys   0m5.752s

---- sfrobs -----

real  0m0.080s
user  0m0.018s
sys   0m0.032s

--- sfrobs -f ---

real  0m0.087s
user  0m0.026s
sys   0m0.026s
```

As we can see from the various time outputs, sfrob and sfrobs run a lot faster than sfrobu. sfrobu uses the unbuffered I/O implementation which slows down performance through the countless system calls to read and write every single byte. This increases overhead a lot more for extremely large files like the one I tested with 5,000,000 bytes. sfrobs is also clearly the fastest program out of all of them because using bash is much more efficient than using c to create the program. sfrobs only contains about 20 lines of code with some mapped values while sfrobu and sfrob contain over 200 lines of code.

Tr2b.c

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

int main(int argc, const char* argv[]) {

  const char* from = argv[1];
  const char* to = argv[2];
  char map[256] = {'\0'};

  // Check whether number of input arguments is valid
  if (argc != 3) {
    fprintf(stderr, "Error: Invalid number of arguments.\n");
    exit(1);
  }

  // Check whether byte strings are the same length
  if (strlen(from) != strlen(to)) {
    fprintf(stderr, "Error: Strings are not the same length.\n");
    exit(1);
  }

  // Map each character in 'from' to each in 'to'
  // Check for duplicates in 'from'
  int i;
  for (i = 0; i < strlen(from); i++) {
    if (map[from[i]] != '\0') {
      fprintf(stderr, "Error: Duplicates detected.\n");
      exit(1);
    }
    map[from[i]] = to[i];
  }

  while (!feof(stdin)) {
    char c = getchar();

    if (c == EOF) { break; }

    // Check for error in reading from stdin
    if (ferror(stdin)) {
      fprintf(stderr,"Error reading from standard input.\n");
      exit(1);
    }

    if (map[c] != '\0') {
      char output = map[c];
      putchar(output);
      if (ferror(stdout)) {
            fprintf(stderr,"Error outputting characters.\n");
            exit(1);
      }
    }
    else {
      putchar(c);
      if (ferror(stdout)) {
            fprintf(stderr,"Error outputting characters.\n");
            exit(1);
      }
    }

  }

}
```

Tr2u.c

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

int main(int argc, const char* argv[]) {

  const char* from = argv[1];
  const char* to = argv[2];
  char map[256] = {'\0'};

  // Buffers for read and write
  char c[1];
  char output[1];

  // Check whether number of input arguments is valid
  if (argc != 3) {
    write(STDERR_FILENO, "Error: Invalid number of arguments.\n", 36);
    exit(1);
  }

  // Check whether byte strings are the same length
  if (strlen(from) != strlen(to)) {
    write(STDERR_FILENO, "Error: Strings are not the same length.\n", 40);
    exit(1);
  }

  // Map each character in 'from' to each in 'to'
  // Check for duplicates in 'from'
  int i;
  for (i = 0; i < strlen(from); i++) {
    if (map[from[i]] != '\0') {
      write(STDERR_FILENO, "Error: Duplicates detected.\n", 28);
      exit(1);
    }
    map[from[i]] = to[i];
  }

  while(read(0, c, 1) > 0) {
    if (map[c[0]] != '\0') {
      output[0] = map[c[0]];
      if (write(1, output, 1) < 0) {
```

```
            write(2, "Error outputting characters.\n", 29);
            exit(1);
    }
   }
   else {
    if (write(1, c, 1) < 0) {
            write(2, "Error outputting characters.\n", 29);
            exit(1);
    }
   }

 }

}
```

Rewrite the sfrob program you wrote previously so that it uses system calls rather than <stdio.h> to read standard input and write standard output. If standard input is a regular file, your program should initially allocate enough memory to hold all the data in that file all at once, rather than the usual algorithm of reallocating memory as you go. However, if the regular file grows while you are reading it, your program should still work, by allocating more memory after the initial file size has been read. Your program should do one thing in addition to sfrob. If given the -f option, your program should ignore case while sorting, by using the standard toupper function to upper-case each byte after decrypting and before comparing the byte. You can assume that each input byte represents a separate character; that is, you need not worry about multi-byte encodings. As noted in its specification, toupper's argument should be either EOF or a nonnegative value that is at most UCHAR_MAX (as defined in <limits.h>); hence one cannot simply pass a char value to toupper, as char is in the range CHAR_MIN..CHAR_MAX.

```
// Assignment 5
// sfrobu.c - similar to sfrob.c functionality, but
// uses system calls instead of <stdio.h>

#include <unistd.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <ctype.h>

bool f_option = false;

// returns an int result that is negative, zero, or positive
// depending on whether a is less than, equal to, or greater than b
int frobcmp(char const * a, char const * b) {

  // check that each argument points to
  // non-space bytes
  while (*a != ' ' && *b != ' ') {

    char a1 = *a ^ 42;
    char b1 = *b ^ 42;

    if (f_option) {
      a1 = toupper((unsigned char) (*a^42));
      b1 = toupper((unsigned char) (*b^42));
    }

    // if a1 is smaller than b1
    // OR
    // a is shorter than b
    // return -1
    if (a1 < b1 || *a == ' ') { return -1; }

    // if a1 is larger than b1
    // OR
    // b is shorter than a
    // return 1
    if (a1 > b1 || *b == ' ') { return 1; }

    // increment index in byte array
    a++;
    b++;
  }

  // if a and b are equal
  // return 0
  return 0;

}

// comparator function to pass into qsort in main
int cmp(const void * a, const void * b) {

  return frobcmp(*(char**) a, *(char**) b);

}

bool checkArgs(int argc, const char*argv[]) {

  // check number of arguments
  if (argc > 2) {
    write(2, "Error: Too many arguments.\n", 27);
    exit(1);
  }
  // verify -f option as only possible second argument
  else if (argc == 2) {
    if (argv[1][0] != '-' && argv[1][1] != 'f') {
      write(2, "Error: Invalid arguments.\n", 26);
      exit(1);
    }
    else {
      return true;
    }
  }

  return false;

}

int main(int argc, const char*argv[]) {

  // checks for -f option
  f_option = checkArgs(argc, argv);

  struct stat buf;
  fstat(0, &buf);
  size_t fileSize;
  if (fstat(0, &buf) < 0) {
    write(2, "Error with system call.\n", 24);
    exit(1);
  }

  char* regFile;
  char** strArray = NULL;
  int s = -1;
  bool addNewString = true;

  if (S_ISREG(buf.st_mode)) {
    fileSize = buf.st_size;

    // allocate memory for file
    regFile = (char*) malloc(sizeof(char) * (fileSize+1));
    if (read(0, regFile, fileSize) < 0) {
      write(2, "Error with system call.\n", 24);
      exit(1);
    }

    // count up number of words in file
    int wordCount = 0;
```

```
  int i;
  for (i = 0; i < fileSize; i++) {
    // make sure first character isn't a space
    if (i == 0 && regFile[i] != ' ') {
            wordCount++;
    }

    if (regFile[i] == ' ') {
            // account for multiple spaces
            while (regFile[i] == ' ' && i < fileSize) {
              i++;
            }
            // continue counting words
            if (i < fileSize) {
              wordCount++;
            }
            }
  }

  // append space at the end of the file
  regFile[fileSize] = ' ';

  // allocate memory for array based on file's word count
  strArray = (char**) malloc(sizeof(char*) * wordCount);
  if (strArray == NULL) {
    write(2, "Error allocating memory.\n", 25);
    exit(1);
  }

  // add words in file to array
  for (i = 0; i < fileSize; i++) {
    if (addNewString && regFile[i] != ' ') {
            s++;
            addNewString = false;
            strArray[s] = &regFile[i];
    }
    if (!addNewString && regFile[i] == ' ') {
            addNewString = true;
    }
  }

}
else {
  // allocate new memory for array
  strArray = (char**) malloc(sizeof(char*));
  if (strArray == NULL) {
    write(2, "Error allocating memory.\n", 25);
    exit(1);
  }
}

char* newString;
char in[1];
char newChar;
int c = 0;
while (true) {

  int r = read(0, in, 1);
  if (r == 0) { break; }
  else if (r < 0) {
    write(2, "Error reading from standard input.\n", 35);
    exit(1);
  }

  // read in character
  newChar = in[0];

  // if program is still adding characters to existing string
  if (!addNewString) {

    // reallocates memory for new character
    newString  = (char*) realloc(newString, (c+1) * sizeof(char));
    if (newString == NULL) {
            write(2, "Error allocating memory.\n", 25);
            exit(1);
    }

    // updates growing string
    newString[c] = newChar;
    strArray[s] = newString;
    c++;

    // ignore multiple spaces
    if (newChar == ' ') {
            addNewString = true;
    }

  }

  // if program must create a new string
  else {

    // reset string length
    c = 0;

    // account for multiple spaces
    if (newChar == ' ' && c == 0) { continue; };

    s++;

    // reallocates memory for growing string
    strArray = (char**) realloc(strArray, (s+1) * sizeof(char*));
    if (strArray == NULL) {
            write(2, "Error allocating memory.\n", 25);
            exit(1);
    }

    // allocate memory for new string
    newString = (char*) malloc(sizeof(char));
    if (newString == NULL) {
            write(2, "Error allocating memory.\n", 25);
            exit(1);
    }

    // updates growing string
    newString[c] = newChar;
    strArray[s] = newString;
    c++;

    addNewString = false;

  }

}

// append trailing space to the very end
if (s != -1 && strArray[s][c-1] != ' ') {

  newString = (char*) realloc(newString, (c+1) * sizeof(char));
  newString[c] = ' ';
  strArray[s] = newString;

}

// use frobcomp to sort array of strings
qsort(strArray, s+1, sizeof(char*), cmp);

// print characters to stdout
int i;
for (i = 0; i < s+1; i++) {

  int j = 0;
```

```
  while(true) {

    in[0] = strArray[i][j];

    // print character by character and catch any errors
    if (write(1, in, 1) < 0) {
            write(2, "Error outputting characters.\n", 29);
            exit(1);
    }

    // skip to the next string if space byte is detected
    if (strArray[i][j] == ' ') { break; }
    j++;

  }

  if (!S_ISREG(buf.st_mode)) {
    free(strArray[i]);
  }
 }

 if (S_ISREG(buf.st_mode)) {
   free(regFile);
 }

 free(strArray);

 exit(0);
}
```

LAB 6

1. Compile, build and run the program simpgmp.c in C on the SEASnet GNU/Linux servers. Since it uses GMP you will need to use the linker flag -lgmp to build this program.
2. Use the program to compute 2**24 (i.e., 2^{24}, or 2 to the 24th power) and 2**(2**24); verify that the latter number has 5,050,446 decimal digits, starts with "1818" and ends with "7536".
3. Use the ldd command to see which dynamic libraries your simple program uses.
4. Use the strace command to see which system calls your simple program makes. Which of these calls are related to dynamic linking and what is the relationship?
5. Suppose your student ID is the 9-digit number *nnnnnnnnn*. On a SEASnet GNU/Linux server, run the shell command “ls /usr/bin | awk '(NR-*nnnnnnnnn*)%251 == 0'” to get a list of nine or so commands to investigate.
6. Invoke ldd on each command in your list. If there are error messages, investigate why they're occurring.
7. Get a sorted list of every dynamic library that is used by any of the commands on your list (omitting duplicates from your list).

1.
$ mkdir Lab6
$ cd Lab6
$ wget http://web.cs.ucla.edu/classes/fall19/cs35L/assign/simpgmp.c
$ cat simpgmp.c

I downloaded the corresponding file and checked to see it worked

$ gcc -o simpgmp simpgmp.c -lgmp

I used gcc to compile and make an executatble, using the -lgmp flag as instructed

2.
$ ./simpgmp 24
output: 16777216

$ ./simpgmp 16777216
output: very very large number starting with 1818 and ending with 7536

The program works correctly with the inputs provided

3.
$ ldd ./simpgmp

Dynamic libraries the program uses:

linux-vdso.so.1 => (0x00007ffe1c5bf000)
libgmp.so.10 => /lib64/libgmp.so.10 (0x00007f17b28e8000)
libc.so.6 => /lib64/libc.so.6 (0x00007f17b251a000)
/lib64/ld-linux-x86-64.so.2 (0x00007f17b2b60000)

4.
$ strace -c ./simpgmp 16777216

% time	seconds	usecs/call	calls	errors	syscall
97.13	0.035509	17755	2		write
2.18	0.000797	53	15		munmap
0.24	0.000086	3	25		mmap
0.14	0.000051	9	6		mprotect
0.12	0.000043	0	100		brk
0.08	0.000028	9	3		open
0.03	0.000012	12	1		access
0.03	0.000011	4	3		close
0.03	0.000011	3	4		fstat
0.02	0.000009	5	2		read
0.01	0.000003	3	1		arch_prctl
0.00	0.000000	0	1		execve
100.00	0.036560		163	1	total

The program uses 25 mmap and 15 munmaps meaning that it is allocating and freeing memory constantly. It also uses 100 brk calls which change the locating of the program's data segment. This basically allocates more memory when it increases the program break, and it deallocates memory when it decreases the program break. In regards to dynamic linking,
open system call, opens the library and mmap saves it to memory. Then it uses close and munmap when it is finished.

5.
$ ls /usr/bin | awk '(NR-105114897)%251 == 0'

commands:
atrm
db_tuner
gemtopbm
hugeedit
krb5-config
mpstat
pamrecolor
pktopbm
python2.7-config
slptool
usb-devices

6.

$ ldd /usr/bin/atrm
        linux-vdso.so.1 => (0x00007ffd1b7f3000)
        librt.so.1 => /lib64/librt.so.1 (0x00007feee2045000)
        libselinux.so.1 => /lib64/libselinux.so.1 (0x00007feee1e1e000)
        libpam.so.0 => /lib64/libpam.so.0 (0x00007feee1c0f000)
        libpam_misc.so.0 => /lib64/libpam_misc.so.0 (0x00007feee1a0b000)
        libc.so.6 => /lib64/libc.so.6 (0x00007feee163d000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007feee1421000)

/usr/bin/krb5-config
/usr/bin/usb-devices
/usr/bin/python2.7-config

These three files had no dynamic libraries. I invoked the file command on each of the commands and these 3 were all not executables, which makes sense
that they do not have dynamic libraries. The remaining files are ELF 64-bit LSB executables or shared objects

krb5-config is a shell script
python2.7-config is a python script
usb-devices is another shell script

7.
```
$ ldd /usr/bin/usb-devices /usr/bin/slptool /usr/bin/python2.7-config
  /usr/bin/pktopbm /usr/bin/pamrecolor /usr/bin/mpstat /usr/bin/krb5-config
  /usr/bin/hugeedit  /usr/bin/gemtopbm /usr/bin/db_tuner  /usr/bin/atrm
  | sed 's/([^()]*)//g'
  | sort -u
```

Sorted Dynamic libraries with no duplicates from 9 commands:

```
        /lib64/ld-linux-x86-64.so.2
    libaudit.so.1 => /lib64/libaudit.so.1
    libcap-ng.so.0 => /lib64/libcap-ng.so.0
    libcrypto.so.10 => /usr/lib64/libcrypto.so.10
    libc.so.6 => /lib64/libc.so.6
```

Next, write a makefile include file randmain.mk that builds the program randmain using three types of linking. First, it should use static linking to combine randmain.o and randcpuid.o into a single program executable randmain. Second, it should use dynamic linking as usual to link the C library and any other necessary system-supplied files before its main function is called. Third, after main is called, it should use dynamic linking via dlsym as described above. randmain.mk should link randmain with the options “-ldl -Wl,-rpath=$PWD”. It should compile randlibhw.c and randlibsw.c with the -fPIC options as well as the other GCC options already used. And it should build shared object files randlibhw.so and randlibsw.so by linking the corresponding object modules with the -shared option, e.g., “gcc ... -shared randlibsw.o -o randlibsw.so”.

Randmain.mk
```
OPTIMIZE = -O2

CC = gcc
CFLAGS = $(OPTIMIZE) -g3 -Wall -Wextra -march=native -mtune=native -mrdrnd

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

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

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

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

randcpuid.o: randcpuid.c
	$(CC) -c randcpuid.c -o randcpuid.o

randmain.o: randmain.c
	$(CC) -c randmain.c

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

LAB 7

1. Get a copy of the Diffutils repository, in Git format, from the file ~eggert/src/gnu/diffutils on the SEASnet GNU/Linux servers, or from its main Savannah repository.
2. Get a log of changes to Diffutils’ master branch using the “git log” command, and put it into the file git-log.txt.
3. Generate a list of tags used for Diffutils using the “git tag” command, and put it into the file git-tags.txt
4. Find the commit entitled “maint: quote 'like this' or "like this", not `like this'”, and generate a patch for that commit, putting it into the file quote-patch.txt.
5. Check out version 3.0 of Diffutils from your repository.
6. Use the patch command to apply quote-patch.txt to version 3.0. In some cases it will not be able to figure out what file to patch; skip past those by typing RETURN. Record any problems you had in applying the patch.
7. Use the git status command to get an overview of what happened.
8. Learn how to use the Emacs functions vc-diff (C-x v =) and vc-revert (C-x v u). When you're in the *vc-diff* buffer generated by vc-diff, use describe-mode (C-h m) to find out the Emacs functions that you can use there, and in particular learn how to use the diff-apply-hunk (C-c C-a) and diff-goto-source (C-c C-c) functions.
9. Use Emacs to revert all the changes to files other than .c files, since you want only changes to .c files. Also, **and don't forget this part**, undo all the changes to .c files other than changes to character string constants, as the character-string changes are the only changes that you want; this may require editing some files by hand.
10. Use Emacs to examine the files src/*.c.rej carefully, and copy rejected patches into the corresponding .c files as needed.
11. Remove all untracked files that git status warns you about, since you don’t plan on adding any files in your patch.
12. When you’re done with the above, git status should report a half-dozen modified files, and git diff should output a patch that is three or four hundred lines long. Put that patch into a file quote-3.0-patch.txt.
13. Build the resulting modified version of Diffutils, using the commands described in the file README-hacking, skipping the part about CVS; CVS is obsolescent. (If you are building on lnxsrv07 or lnxsrv09 or any other host that is using version 2.16 or later of the GNU C Library, you will need to apply an additional patch after running ./bootstrap and before running ./configure, because glibc 2.16 removed the obsolete and dangerous gets function declared by a Diffutils header.) Verify that Diffutils does the right thing with the “diff . -” scenario, as well as with “diff --help”.
14. Do a sanity test using the modified version of Diffutils that you just built, by using the just-built diff to compare the source code of Diffutils 3.0 to the source code of your modified version. Put the former source code into a directory diffutils-3.0 and the latter source code into a directory diffutils-3.0-patch, and run your implementation of diff with the command “*D*/diff -pru diffutils-3.0 diffutils-3.0-patch >quote-3.0-test.txt”, where the *D* is the directory containing your diff implementation.
15. Use diff to compare the contents of quote-3.0-test.txt and quote-3.0-patch.txt. Are the files identical? If not, are the differences innocuous?

Assignment 9
Laboratory
---

1)

To get a copy of the Diffutils repository in Git format (from its main Savannah repository), I ran:

```
$ git clone https://git.savannah.gnu.org/git/diffutils.git
```

2)

To get a log of changes to Diffutils' master branch and put it into the file git-log.txt, I ran:

```
$ cd diffutils/
$ git log > git-log.txt
```

3)

To generate a list of tags used for Diffutils and put it into the file git-tags.txt, I ran:

```
$ git tag > git-tags.txt
```

4)

To find a commit entitled "maint: quote 'like this' or "like this", not `like this'", generate a patch for the commit, and put it into the file quote-patch.txt, I ran:

```
$ git log --all --grep='maint: quote'
commit 62ca21c8c1a5aa3488589dcb191a4ef04ae9ed4f
Author: Paul Eggert <eggert@cs.ucla.edu>
Date:   Wed Jan 25 23:46:16 2012 -0800

$ git format-patch -1 62ca21c8c1a5aa3488589dcb191a4ef04ae9ed4f
  --stdout > quote-patch.txt
```

5)

To check out version 3.0 of Diffiutils from my repository, I first inspected the git-tags.txt file to see each version, then ran:

```
$ git checkout tags/v3.0
```

6)

I used the patch command to apply quote-patch.txt to version 3.0. Everytime the command could not figure out what file to patch, I skipped it by hitting RETURN:

```
$ patch -p1 < quote-patch.txt
```

The errors that appeared were the following:

```
Hunk #10 FAILED at 876.
Hunk #11 FAILED at 906.
Hunk #12 FAILED at 942.
3 out of 14 hunks FAILED -- saving rejects to file src/diff.c.rej

Hunk #6 FAILED at 433.
1 out of 9 hunks FAILED -- saving rejects to file src/diff3.c.rej
```

7)

To get an overview of what happened, I ran Git's status command:

```
$ git status

# HEAD detached at v3.0
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   NEWS
#       modified:   README
#       modified:   TODO
#       modified:   doc/diagmeet.note
#       modified:   ms/config.bat
#       modified:   ms/config.site
#       modified:   po/en.po
#       modified:   src/analyze.c
#       modified:   src/cmp.c
#       modified:   src/context.c
#       modified:   src/diff.c
#       modified:   src/diff.h
#       modified:   src/diff3.c
#       modified:   src/dir.c
#       modified:   src/ifdef.c
#       modified:   src/io.c
#       modified:   src/sdiff.c
#       modified:   src/side.c
#       modified:   src/system.h
#       modified:   src/util.c
#       modified:   tests/help-version
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       NEWS.orig
#       README-hacking.orig
#       README-hacking.rej
#       README.orig
#       README.rej
#       cfg.mk.orig
#       cfg.mk.rej
#       git-log.txt
#       git-tags.txt
#       ms/config.site.orig
#       quote-patch.txt
#       src/cmp.c.orig
#       src/context.c.orig
#       src/diff.c.orig
#       src/diff.c.rej
#       src/diff.h.orig
#       src/diff3.c.orig
#       src/diff3.c.rej
#       src/dir.c.orig
#       src/sdiff.c.orig
#       src/system.h.orig
#       src/util.c.orig
#       tests/help-version.orig
no changes added to commit (use "git add" and/or "git commit -a")
```

8)

I learned how to use the Emacs functions vc-diff (C-x v =) and vc-revert (C-x v u). I used describe-mode (C-h m) in the *vc-diff* buffer generated by vc-diff to learn how to use the diff-apply-hunk (C-c C-a) and diff-goto-source (C-c C-c) functions.

vc-diff : Compare the work files in the current VC fileset with the versions you started from

vc-revert : Revert the work file(s) in the current VC fileset to the last revision

diff-apply-hunk : Apply this hunk to its target file

diff-goto-source : Go to the source file and line corresponding to this hunk

9)

To revert all the changes to files other than .c files in Emacs, I opened each of the appropiate files, ran vc-revert, and typed yes when prompted for confirmation to discard changes:

```
$ emacs <FILENAME>
C-x v u
```

(Entered 'yes' when prompted)

The files I reverted were NEWS, README, TODO, doc/diagmeet.note, ms/config.bat, ms/config.site, po/en.po, src/diff.h, src/system.h, and tests/help-version.

To undo all the changes to .c files other than changes to character string constants, I opened each .c file, used vc-diff to see each change, and used diff-apply-hunk to revert the appropiate changes:

```
$ emacs <FILENAME>
C-x v =
C-u C-c C-a
```

The files I changed were src/analyze.c, src/cmp.c, src/context.c, src/diff.c, src/diff3.c, src/dir.c, src/ifdef.c, src/io.c, src/sdiff.c, src/side.c, and src/util.c.

The only files that contained changes to charaacter string constants (which I kept) were src/cmp.c, src/diff.c, src/diff3.c, src/dir.c, src/sdiff.c, and src/util.c.

10)

To find files in the src directory that ended in .c.rej, I ran:

```
$ cd src
$ find *.c.rej
diff3.c.rej
diff.c.rej
```

There were only 2 files that came up, so I opened each of these and manually replaced every instance of ` with ' in the corresponding files wherever the patches specified.

```
$ emacs diff3.c.rej
$ emacs diff.c.rej

(In a different window, using C-s to search for the correct statements)
$ emacs diff3.c
$ emacs diff.c
```

11)

To remove all untracked files that git status warns me about, I first moved the files I made earlier outside of the directory, then used the clean command to get rid of everything else:

```
$ mv git-log.txt ..
$ mv git-tags.txt ..
$ mv quote-patch.txt ..
$ git clean -f
```

12)

Now running git status, I see the following 6 modified files:

```
$ git status
modified:   src/cmp.c
modified:   src/diff.c
modified:   src/diff3.c
modified:   src/dir.c
modified:   src/sdiff.c
modified:   src/util.c
```

I then produced a patch with 3 or 4 hundred lines, putting it into a file called quote-3.0-patch.txt:

```
$ git diff > quote-3.0-patch.txt
$ mv quote-3.0-patch.txt
$ cat quote-3.0-patch.txt | wc -l
```


13)

I examined the contents of README-hacking in order to build the resulting modified version of Diffutils. Since I am using lnxsrv03, I didn't have to apply an additional patch:

```
$ ./bootstrap
$ ./configure
$ make
$ make check
```

I recieved no errors with a message telling me that "All 88 tests passed". To verify that Diffutils does the right thing, I ran the following two commands:

```
$ src/diff . -
diff: cannot compare '-' to a directory

$ src/diff --help
```

The output for this second command was a bit long, but examining the few lines I know I edited earlier, we can see that it outputs with ' rather than `. Here are a few examples:

```
-l  --paginate  Pass the output through 'pr' to paginate it.

FILES are 'FILE1 FILE2' or 'DIR1 DIR2' or 'DIR FILE...' or 'FILE... DIR'.

If a FILE is '-', read standard input.

LTYPE is 'old', 'new', or 'unchanged'.  GTYPE is LTYPE or 'changed'.
```

14)

I now perform a sanity test by using the just-built diff to compare the source code of Diffutils 3.0 to the source code of my modified version.

I started by copying the source code of the modified version into a new directory:

```
$ cd ..
$ mkdir diffutils-3.0-patch
$ cp -r diffutils/src/*.c ./diffutils-3.0-patch
```

I then cloned a new unmodified version of diffutils into another folder and changed the version to v3.0 as I did previously:

```
$ git clone https://git.savannah.gnu.org/git/diffutils.git
  diffutils-original
$ cd diffutils-original
$ git checkout tags/v3.0
$ cd ..
```

Next I copied the source code of the unmodified 3.0 version into a new directory:

```
$ mkdir diffutils-3.0
$ cp -r diffutils-original/src/*.c ./diffutils-3.0
```

I then ran my implementation of diff with the following command:

```
$ diffutils/src/diff -pru diffutils-3.0 diffutils-3.0-patch
  > quote-3.0-test.txt
```

15)

Lastly, I used diff to compare the contents of quote-3.0-test.txt and quote-3.0-patch.txt to see the differences or similarities:

```
$ diffutils/src/diff quote-3.0-patch.txt quote-3.0-test.txt
```

The output was very lengthy, but after a closer look, I can see that
the files are (correctly) not identical, with only minor differences
in corrections from ` to '.

1. Maintain a file hw9.txt that logs the actions you do in solving the homework. This is like your lab notebook lab9.txt, except it's for the homework instead of the lab.
2. Check out version 3.0 of Diffutils from your repository, into a new branch named "quote".
3. Install your change into this new branch, by running the patch command with your patch quote-3.0-patch.txt as input.
4. Learn how to use the Emacs function add-change-log-entry-other-window (C-x 4 a).
5. Use this Emacs function to compose an appropriate ChangeLog entry for your patch, by adapting the change log from the original patch.
6. Commit your changes to the new branch, using the ChangeLog entry as the commit message.
7. Use the command "git format-patch" to generate a file formatted-patch.txt. This patch should work without having to fix things by hand afterwards.
8. Your teaching assistant will assign you a partner, who will also generate a patch. Verify that your partner's patch works, by checking out version 3.0 again into a new temporary branch partner, applying the patch with the command "git am", and building the resulting system, checking that it works with "make check".
9. Verify that your ChangeLog entry works, by running the command "make distdir" and inspecting the resulting diffutils*/ChangeLog file.
10. There is a copy of the GNU Emacs git repository's master branch on SEASnet in the directory ~eggert/src/gnu/emacs. Run the command gitk on it, and find the newest merge that is not newer than 2015-01-25. Briefly describe (in hw9.txt) your view of the mergepoint, along with the roles of subwindows that you see.

Assignment 9
Homework
---

1)

I created this file hw9.txt to log each step of the homework:

```
$ emacs hw9.txt
```

2)

To check out version 3.0 of Diffutils from my repository into
a new branch named "quote", I ran:

```
$ cd diffutils
$ git checkout v3.0 -b quote
M src/cmp.c
M src/diff.c
M src/diff3.c
M src/dir.c
M src/sdiff.c
M src/util.c
Switched to a new branch 'quote'
```

3)

I installed my change into this new branch by running:

```
$ patch -p1 < quote-3.0-patch.txt
```

The patch seemed to be applied without any errors.

4)

I learned how to use the Emacs function add-change-log-entry-other-window.

C-x 4 a : adds a new entry to the change log file for the file
you are editing

5)

I composed an appropiate ChangeLog entry for my patch by adapting
the change log from the quote-patch.txt using the
add-change-log-entry-other-window Emacs function:

```
$ emacs quote-3.0-patch.txt
C-x 4 a

2019-03-12  Kathy Douangpanya Daniels  <kathy@lnxsrv09.seas.ucla.edu>

    * quote-3.0-patch.txt: This patch was created in response
    to a recent change in the GNU coding standards, which now
    suggests quoting 'like this' or "like this", instead of
    `like this' or ``like this''. Note that these changes are
    only applied to character string constants in .c files.

C-x C-s
C-x C-c
```

6)

I then committed my changes to the new branch using the ChangeLog
entry as the commit message:

```
$ mv quote-3.0-patch.txt ..
$ git add .
$ git commit -F ChangeLog
```

7)

I then generated a new patch file that should work without having
to fix things by hand afterwards:

```
$ git format-patch -1 --stdout > formatted-patch.txt
```

8)

Next, my partner and I exchanged patches through email for testing.
I moved this file onto the SEASnet server and into diffutils.
To verify that my partner's patch works, I checked out version 3.0
again into a new temporary branch and applied her patch:

```
$ git checkout v3.0 -b quote-2
$ git am < nikki-formatted-patch.txt
$ ./bootstrap
$ ./configure
$ make
$ make check
```

As before, I recieved no error, along with a message telling me that
all 88 tests successfully passed!

I tested the following commands again:

```
$ src/diff . -
src/diff: cannot compare '-' to a directory

$ src/diff --help
```

Examining the lengthy output for lines that are supposed to contain
' instead of `, I could see that the patch was successful! Here
are a few examples:

```
LTYPE is 'old', 'new', or 'unchanged'.  GTYPE is LTYPE or 'changed'.
```

```
  -l  --paginate  Pass the output through 'pr' to paginate it.

  FILES are 'FILE1 FILE2' or 'DIR1 DIR2' or 'DIR FILE...' or 'FILE... DIR'.

  If a FILE is '-', read standard input.
```

9)

To verify that my ChangeLog entry works, I ran:

```
  $ make distdir
```

A new directory called diffutils-3.0.1-ac41 was created! I opened the directory and examined the ChangeLog file inside:

```
  $ cd diffutils-3.0.1-ac41
  $ emacs ChangeLog
```

I could see that my partner's ChangeLog entry was at the top of the file, so we know that the ChangeLog entries worked.

10)

I logged out of the SEASnet server and logged back in using the -X flag to enable X forwarding:

```
  $ ssh -X kathy@lnxsrv09.seas.ucla.edu
```

I then went into the directory containing the copy of the GNU Emacs git repository's master branch and ran gitk on it:

```
  $ cd ~eggert/src/gnu/emacs
  $ gitk
```

A window popped up soon after in XQuartz containing the repository. I scrolled down the long list of commits to find the newest merge that is no newer than 2015-01-25.

LAB 8

Assignment 8
Laboratory

======
SET UP
======

I started by connecting my BeagleBone to my computer using the USB cable.

Once I found BEAGLEBONE as an external storage device, I opened it and clicked the "START.htm" file. This led me to a webpage located on the BeagleBone that led me through instructions on what drivers to download.

After I restarted my computer, I used the following command to ssh into my BeagleBone:

```
  $ sudo ssh root@192.168.7.2
  Password:
```

Once successfully SSHing into my BeagleBone, I connected it to wifi by executing the following commands:

```
  # sudo connmanctl
  # enable wifi
  # scan wifi
  # services
  # agent on
```

I then connected to the lab's wifi, entered the given password, and ran quit to exit.

==========
IP ADDRESS
==========

To find the IP address of my BeagleBone, I ran the following:

```
  # ifconfig

SoftAp0   Link encap:Ethernet  HWaddr 40:bd:32:e4:7c:10
          inet addr:192.168.8.1  Bcast:192.168.8.255  Mask:255.255.255.0
          inet6 addr: fe80::42bd:32ff:fee4:7c10/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:95 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:25038 (24.4 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1045 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1045 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:87205 (85.1 KiB)  TX bytes:87205 (85.1 KiB)

usb0      Link encap:Ethernet  HWaddr 40:bd:32:e4:7c:12
          inet addr:192.168.7.2  Bcast:192.168.7.255  Mask:255.255.255.0
          inet6 addr: fe80::42bd:32ff:fee4:7c12/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3474 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2171 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:298671 (291.6 KiB)  TX bytes:530439 (518.0 KiB)

wlan0     Link encap:Ethernet  HWaddr 2c:f7:f1:06:79:0b
          inet addr:10.97.85.209  Bcast:10.97.85.255  Mask:255.255.255.0
          inet6 addr: fe80::2ef7:f1ff:fe06:790b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST DYNAMIC  MTU:1500
Metric:1
          RX packets:4293 errors:0 dropped:10 overruns:0 frame:0
          TX packets:1497 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:756954 (739.2 KiB)  TX bytes:241593 (235.9 KiB)
```

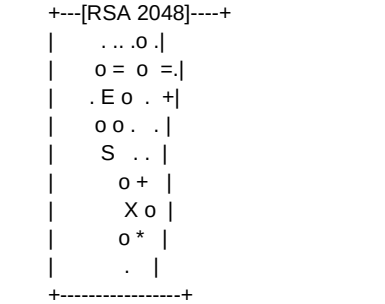
Looking under the network interface "wlan0", we can see that my IP address is 10.97.85.209. I gave this to my partner so she could later ssh into my BeagleBone.

============
SERVER STEPS
============

I started by generating public and private keys:

```
  $ ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
9c:14:da:fd:ee:61:73:c2:8d:53:c5:0d:27:38:e0:92 root@beaglebone
```

The key's randomart image is:

```
+---[RSA 2048]----+
|       . .. .o .|
|      o =  o  =.|
|     . E o  .  +|
|      o o .   . |
|       S   . .  |
|         o +    |
|          X o   |
|         o *    |
|          .     |
+----------------+
```

I created an account for my partner (the client) on the server:

```
$ sudo useradd -d /home/nik -m nik
$ sudo passwd nik
```

The password I set was: password
I then created a .ssh directory for my partner:

```
$ cd /home/nik
$ sudo mkdir .ssh
```

I changed ownership and permission on the .ssh directory so that it could be read, written and executed by my partner:

```
$ sudo chown -R nik .ssh
$ sudo chmod 700 .ssh
```

Once the server was set up and my partner had my IP address, she performed all the following client steps and we switched roles.

==========
CLIENT STEPS
==========

After my partner finished completing all the same steps that I did above, she gave me her IP address so I could SSH into her BeagleBone.

I started by generating public and private keys once again:

```
$ ssh-keygen
```

I then copied my public key to the server for key-based authentication (/root/.ssh/authorized_keys):

```
$ ssh-copy-id -i kathy@10.97.85.205
```

Then I added the private key to the authentication agent:

```
$ eval $(ssh-agent)
$ ssh-add
```

Once everything was finally set up, I SSHed into my partner's BeagleBone:

```
$ ssh kathy@10.97.85.205
```

Once connected to the server, I created a test file to see if my partner would be able to view or access the file on her BeagleBone:

```
$ nano test.txt
```

After inputting some random characters and saving the file, my partner confirmed that she was able to see the file, showing us that the connection between server and client was successfully working.

==========
X11 FORWARDING
==========

Next, I wanted to try trusted port forwarding. Logging out of my partner's BeagleBone and then my own, I re-SSHed into my onto my BeagleBone with X11 forwarding and SSHed into my partner's BeagleBone with X11 forwarding as well:

```
$ sudo ssh -X root@192.168.7.2
$ ssh -X kathy@10.97.85.205
```

I then tried running firefox:

```
$ firefox
```

The firefox browser came up successfully on my laptop!

Lastly, I wanted to try trusted X11 forwarding using the -Y flag instead. I had to again log out of my BeagleBone and my partner's and re-SSH onto them with the following commands:

```
$ sudo ssh -Y root@192.168.7.2
$ ssh -Y kathy@10.97.85.205
```

I ran firefox again on the server and the browser came up successfully once again! Although there were no obvious differences between the two procedures (besides changing the flags) and results, using trusted forwarding (-Y) could open security issues since it causes the remote machine to be treated as a trusted client. Other clients could steal data from the remote machine or even alter data. In contrast, using unstrusted forwarding would cause an error to pop up if any commands violate a security setting.

HW
1. Suppose the other teams really had been observing all the bytes going across the network in your lab exercise. Is your resulting network still secure? If so, explain why, and explain whether your answer would change if (1) you assumed the other teams had also tapped your keyboards after you completed all client-server setup and had thereby obtained your team's keystrokes during later operation, or (2) you are booting off USB and you assume the other teams temporarily had physical control of the USB. If not, explain any weaknesses of your team's setups, focusing on possible attacks by such outside observers.

2. Explain why the gpg2 --verify command in the following instructions doesn't really verify that you personally created the file in question. How would you go about fixing this problem?

Assignment 8
Homework

1.

If other teams really had been observing all the bytes going across the network, the resulting network would still be secure. This is because secure shell (SSH) is encrypted and operates securely, meaning that data going across the network is protected and may only be decrypted and accessed by the server or client.

(1)

The network would still remain secure even if other teams had also tapped our keyboards after completing the client-server setup since we successfully accomplished password-free authentication, preventing other teams from obtaining the keystrokes to our passwords.

However, had we not accomplished password-free access using our public and private keys, we would have had to type our passwords each time, thereby giving other teams access to our password keystrokes!

(2)

The network would no longer be secure if other teams had temporary

physical control of the USB since they could obtain access to the .ssh file we created, therefore revealing our private key. If they were to access this information, they would then be able to log onto the server (assuming password-free authentication is still enabled).

2.

The gpg2 --verify command doesn't really verify that I personally created the file in question because it only compares the file with a detached signature to see if they match. This becomes a serious problem if an attacker manages to create a separate detached signature after tampering with private data. In this case, the verify command would then still verify that the file and signature match, but not WHO signed the files.

To fix this issue, a third party could be utilized. Certificate Authority (CA) issues digital certificates that cerify the ownership of a public key by the specific named subject of the certificate. Therefore, if I were to supply them with my public key, they would be able to verify signatures concerning the private key that corresponds to the certified public key.

LAB 9

1. How much disk space is used to represent the working files in this directory? How much is used to represent the Git repository? What file or files consume most of the repository space and why?
2. How many branches are local to the repository? How many are remote?
3. How many repositories are remote to this repository, and where are they?
4. What are the ten local branches most recently committed to? List their names in order of commit date, most recent first.
5. How many commits are in the master branch?
6. What percentage of the commits that are in any branch, are also in the master branch?
7. Which ten people authored the most master-branch commits that were committed in the year 2013 or later, and how many commits did each of them author during that period?
8. Use the gitk command to visualize the commit graph in this repository. If you are SSHing into SEASnet, you'll need to log in via ssh -X or (less securely) ssh -Y. Draw a diagram relating the following commits to each other, and explain what likely happened to cause their commit-graph neighborhood. You need not list every single intervening commit individually; you can simply use ellipses

CS 35L Lab 9

1.
Find disk space for working files in directory
$ cd ~eggert/src/gnu/emacs-CS-35L
$ du -sh .

du is disk usage, -s is summarize total, ad -h is human readable.
Output: 509M

the directory has 509MB of disk space

Disk space for git repository

$ git count-objects -vH

count: 0
size: 0 bytes
in-pack: 871160
packs: 1
size-pack: 333.34 MiB
prune-packable: 0
garbage: 0
size-garbage: 0 bytes

Total size is 333.34MB for the git repo

Git blobs take up the most space because they are binary large objects that are used to store the contents of each file in a repository.

2.
Number of Local and Remote branches

$ git branch | wc -l

176 local branches

$ git branch -r | wc -l

177 remote branches

3.
Remote repositories

$ git remote -v
origin https://git.savannah.gnu.org/git/emacs.git (fetch)

There is one remote repository (origin) and it is stored at the above URL.

4.
Most recent local branches

$ git branch --sort=-committerdate | head -10

* master
  scratch/joaot/make-completion-at-point-function
  feature/windows-with-utils
  scratch/completion-api
  scratch/a-modest-completion-redesign-proposal
  scratch/fido-mode
  feature/gnus-select2
  feature/extend_face_id
  scratch/jit-lock-antiblink-cleaned-up
  emacs-26

5.
Count how many commits on master branch

$ git rev-list --count master

139583 Commits

6.
Percentage of all commits that are also in master branch

$ git rev-list --count -all
143910 total commits

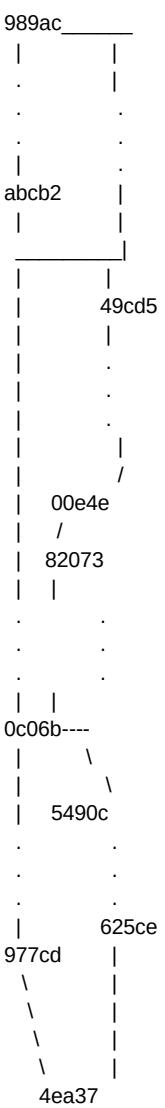
139583/143910 = 97.0%

7.
Most commits by author since 2013, top 10

-ns is organize by number and summarize

```
$ git shortlog -ns --since=2013 | head -10
  3230  Eli Zaretskii
  3175  Paul Eggert
  2602  Glenn Morris
  1806  Lars Ingebrigtsen
  1389  Michael Albinus
  1367  Stefan Monnier
   576  Noam Postavsky
   517  Dmitry Gutov
   438  Alan Mackenzie
   413  Juri Linkov
```

8.
Gitk Graph



Starting at 4ea37, it branched into two paths, one for 977cd and 625ce.
625ce continued to 5490c and then merged back with 977cd at 0c06b.
However,
that second branch kept going in 82073. It created another sub branch at
00e4
and that continued until 49cd5. This branch merged with the first at abcb2.
49cd5 continued its branch until finally merging at 989ac. abcb2 also
continued
and to 989ac.