FILE SPACING:

```
# double space a file
sed G

# double space a file which already has blank lines in it. Output file
# should contain no more than one blank line between lines of text.
sed '/^$/d;G'

# triple space a file
sed 'G;G'

# undo double-spacing (assumes even-numbered lines are always blank)
sed 'n;d'

# insert a blank line above every line which matches "regex"
sed '/regex/{x;p;x;}'

# insert a blank line below every line which matches "regex"
sed '/regex/G'

# insert a blank line above and below every line which matches "regex"
sed '/regex/{x;p;x;G;}'
```

NUMBERING:

```
# number each line of a file (simple left alignment). Using a tab (see
# note on '\t' at end of file) instead of space will preserve margins.
sed = filename | sed 'N;s/\n/\t/'

# number each line of a file (number on left, right-aligned)
sed = filename | sed 'N; s/^/     /; s/ *\(.\{6,\}\)\n/\1  /'

# number each line of file, but only print numbers if line is not blank
sed '/./=' filename | sed '/./N; s/\n/ /'

# count lines (emulates "wc -l")
sed -n '$='
```

TEXT CONVERSION AND SUBSTITUTION:

```
# IN UNIX ENVIRONMENT: convert DOS newlines (CR/LF) to Unix format.
sed 's/.$//'               # assumes that all lines end with CR/LF
sed 's/^M$//'              # in bash/tcsh, press Ctrl-V then Ctrl-M
sed 's/\x0D$//'            # works on ssed, gsed 3.02.80 or higher

# IN UNIX ENVIRONMENT: convert Unix newlines (LF) to DOS format.
sed "s/$/`echo -e \\\r`/"            # command line under ksh
sed 's/$'"/`echo \\\r`/"             # command line under bash
sed "s/$/`echo \\\r`/"               # command line under zsh
sed 's/$/\r/'                        # gsed 3.02.80 or higher

# IN DOS ENVIRONMENT: convert Unix newlines (LF) to DOS format.
sed "s/$//"                          # method 1
sed -n p                             # method 2

# IN DOS ENVIRONMENT: convert DOS newlines (CR/LF) to Unix format.
# Can only be done with UnxUtils sed, version 4.0.7 or higher. The
# UnxUtils version can be identified by the custom "--text" switch
# which appears when you use the "--help" switch. Otherwise, changing
# DOS newlines to Unix newlines cannot be done with sed in a DOS
# environment. Use "tr" instead.
sed "s/\r//" infile >outfile         # UnxUtils sed v4.0.7 or higher
tr -d \r <infile >outfile            # GNU tr version 1.22 or higher

# delete leading whitespace (spaces, tabs) from front of each line
# aligns all text flush left
sed 's/^[ \t]*//'                    # see note on '\t' at end of file

# delete trailing whitespace (spaces, tabs) from end of each line
sed 's/[ \t]*$//'                    # see note on '\t' at end of file

# delete BOTH leading and trailing whitespace from each line
sed 's/^[ \t]*//;s/[ \t]*$//'

# insert 5 blank spaces at beginning of each line (make page offset)
sed 's/^/     /'

# align all text flush right on a 79-column width
sed -e :a -e 's/^.\{1,78\}$/ &/;ta'  # set at 78 plus 1 space

# center all text in the middle of 79-column width. In method 1,
# spaces at the beginning of the line are significant, and trailing
# spaces are appended at the end of the line. In method 2, spaces at
# the beginning of the line are discarded in centering the line, and
# no trailing spaces appear at the end of lines.
sed  -e :a -e 's/^.\{1,77\}$/ & /;ta'                  # method 1
sed  -e :a -e 's/^.\{1,77\}$/ &/;ta' -e 's/\( *\)\1/\1/'  # method 2

# substitute (find and replace) "foo" with "bar" on each line
sed 's/foo/bar/'             # replaces only 1st instance in a line
sed 's/foo/bar/4'            # replaces only 4th instance in a line
sed 's/foo/bar/g'            # replaces ALL instances in a line
sed 's/\(.*\)foo\(.*foo\)/\1bar\2/' # replace the next-to-last case
sed 's/\(.*\)foo/\1bar/'           # replace only the last case

# substitute "foo" with "bar" ONLY for lines which contain "baz"
sed '/baz/s/foo/bar/g'

# substitute "foo" with "bar" EXCEPT for lines which contain "baz"
sed '/baz/!s/foo/bar/g'

# change "scarlet" or "ruby" or "puce" to "red"
sed 's/scarlet/red/g;s/ruby/red/g;s/puce/red/g'   # most seds
gsed 's/scarlet\|ruby\|puce/red/g'                # GNU sed only

# reverse order of lines (emulates "tac")
# bug/feature in HHsed v1.5 causes blank lines to be deleted
sed '1!G;h;$!d'               # method 1
sed -n '1!G;h;$p'             # method 2

# reverse each character on the line (emulates "rev")
sed '/\n/!G;s/\(.\)\(.*\n\)/&\2\1/;//D;s/.//'

# join pairs of lines side-by-side (like "paste")
sed '$!N;s/\n/ /'

# if a line ends with a backslash, append the next line to it
sed -e :a -e '/\\$/N; s/\\\n//; ta'

# if a line begins with an equal sign, append it to the previous line
# and replace the "=" with a single space
sed -e :a -e '$!N;s/\n=/ /;ta' -e 'P;D'

# add commas to numeric strings, changing "1234567" to "1,234,567"
gsed ':a;s/\B[0-9]\{3\}\>/,&/;ta'                     # GNU sed
sed -e :a -e 's/\(.*[0-9]\)\([0-9]\{3\}\)/\1,\2/;ta'  # other seds

# add commas to numbers with decimal points and minus signs (GNU sed)
gsed -r ':a;s/(^|[^0-9.])([0-9]+)([0-9]{3})/\1\2,\3/g;ta'

# add a blank line every 5 lines (after lines 5, 10, 15, 20, etc.)
gsed '0~5G'                  # GNU sed only
sed 'n;n;n;n;G;'             # other seds
```

SELECTIVE PRINTING OF CERTAIN LINES:

```
# print first 10 lines of file (emulates behavior of "head")
sed 10q

# print first line of file (emulates "head -1")
sed q

# print the last 10 lines of a file (emulates "tail")
sed -e :a -e '$q;N;11,$D;ba'

# print the last 2 lines of a file (emulates "tail -2")
sed '$!N;$!D'

# print the last line of a file (emulates "tail -1")
sed '$!d'                    # method 1
sed -n '$p'                  # method 2

# print the next-to-the-last line of a file
sed -e '$!{h;d;}' -e x              # for 1-line files, print blank line
sed -e '1{$q;}' -e '$!{h;d;}' -e x  # for 1-line files, print the line
sed -e '1{$d;}' -e '$!{h;d;}' -e x  # for 1-line files, print nothing

# print only lines which match regular expression (emulates "grep")
sed -n '/regexp/p'           # method 1
sed '/regexp/!d'             # method 2

# print only lines which do NOT match regexp (emulates "grep -v")
sed -n '/regexp/!p'          # method 1, corresponds to above
sed '/regexp/d'              # method 2, simpler syntax

# print the line immediately before a regexp, but not the line
# containing the regexp
sed -n '/regexp/{g;1!p;};h'

# print the line immediately after a regexp, but not the line
# containing the regexp
sed -n '/regexp/{n;p;}'

# print 1 line of context before and after regexp, with line number
# indicating where the regexp occurred (similar to "grep -A1 -B1")
sed -n -e '/regexp/{=;x;1!p;g;$!N;p;D;}' -e h

# grep for AAA and BBB and CCC (in any order)
sed '/AAA/!d; /BBB/!d; /CCC/!d'

# grep for AAA and BBB and CCC (in that order)
sed '/AAA.*BBB.*CCC/!d'

# grep for AAA or BBB or CCC (emulates "egrep")
sed -e '/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d    # most seds
gsed '/AAA\|BBB\|CCC/!d'                        # GNU sed only

# print paragraph if it contains AAA (blank lines separate paragraphs)
# HHsed v1.5 must insert a 'G;' after 'x;' in the next 3 scripts below
sed -e '/./{H;$!d;}' -e 'x;/AAA/!d;'

# print paragraph if it contains AAA and BBB and CCC (in any order)
sed -e '/./{H;$!d;}' -e 'x;/AAA/!d;/BBB/!d;/CCC/!d'

# print paragraph if it contains AAA or BBB or CCC
sed -e '/./{H;$!d;}' -e 'x;/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d
gsed '/./{H;$!d;};x;/AAA\|BBB\|CCC/b;d'         # GNU sed only

# print only lines of 65 characters or longer
sed -n '/^.\{65\}/p'

# print only lines of less than 65 characters
sed -n '/^.\{65\}/!p'        # method 1, corresponds to above
sed '/^.\{65\}/d'            # method 2, simpler syntax

# print section of file from regular expression to end of file
sed -n '/regexp/,$p'

# print section of file based on line numbers (lines 8-12, inclusive)
sed -n '8,12p'               # method 1
sed '8,12!d'                 # method 2

# print line number 52
sed -n '52p'                 # method 1
sed '52!d'                   # method 2
sed '52q;d'                  # method 3, efficient on large files

# beginning at line 3, print every 7th line
gsed -n '3~7p'               # GNU sed only
sed -n '3,${p;n;n;n;n;n;n;}' # other seds

# print section of file between two regular expressions (inclusive)
sed -n '/Iowa/,/Montana/p'             # case sensitive
```

SELECTIVE DELETION OF CERTAIN LINES:

```
# print all of file EXCEPT section between 2 regular expressions
sed '/Iowa/,/Montana/d'

# delete duplicate, consecutive lines from a file (emulates "uniq").
# First line in a set of duplicate lines is kept, rest are deleted.
sed '$!N; /^\(.*\)\n\1$/!P; D'

# delete duplicate, nonconsecutive lines from a file. Beware not to
# overflow the buffer size of the hold space, or else use GNU sed.
sed -n 'G; s/\n/&&/; /^\([ -~]*\n\).*\n\1/d; s/\n//; h; P'

# delete all lines except duplicate lines (emulates "uniq -d").
sed '$!N; s/^\(.*\)\n\1$/\1/; t; D'

# delete the first 10 lines of a file
sed '1,10d'

# delete the last line of a file
sed '$d'

# delete the last 2 lines of a file
sed 'N;$!P;$!D;$d'

# delete the last 10 lines of a file
sed -e :a -e '$d;N;2,10ba' -e 'P;D'   # method 1
sed -n -e :a -e '1,10!{P;N;D;};N;ba'  # method 2

# delete every 8th line
gsed '0~8d'                           # GNU sed only
sed 'n;n;n;n;n;n;n;d;'                # other seds

# delete lines matching pattern
sed '/pattern/d'
```

```
 # delete ALL blank lines from a file (same as "grep '.' ")
 sed '/^$/d'                    # method 1
 sed '/./!d'                    # method 2

 # delete all CONSECUTIVE blank lines from file except the first; also
 # deletes all blank lines from top and end of file (emulates "cat -s")
 sed '/./,/^$/!d'           # method 1, allows 0 blanks at top, 1 at EOF
 sed '/^$/N;/\n$/D'         # method 2, allows 1 blank at top, 0 at EOF

 # delete all CONSECUTIVE blank lines from file except the first 2:
 sed '/^$/N;/\n$/N;//D'

 # delete all leading blank lines at top of file
 sed '/./,$!d'

 # delete all trailing blank lines at end of file
 sed -e :a -e '/^\n*$/{$d;N;ba' -e '}'  # works on all seds
 sed -e :a -e '/^\n*$/N;/\n$/ba'        # ditto, except for gsed 3.02.*

 # delete the last line of each paragraph
 sed -n '/^$/{p;h;};/./{x;/./p;}'

SPECIAL APPLICATIONS:

 # remove nroff overstrikes (char, backspace) from man pages. The 'echo'
 # command may need an -e switch if you use Unix System V or bash shell.
 sed "s/.`echo \\\b`//g"    # double quotes required for Unix environment
 sed 's/.^H//g'             # in bash/tcsh, press Ctrl-V and then Ctrl-H
 sed 's/.\x08//g'           # hex expression for sed 1.5, GNU sed, ssed

 # get Usenet/e-mail message header
 sed '/^$/q'                # deletes everything after first blank line

 # get Usenet/e-mail message body
 sed '1,/^$/d'              # deletes everything up to first blank line

 # get Subject header, but remove initial "Subject: " portion
 sed '/^Subject: */!d; s///;q'

 # get return address header
 sed '/^Reply-To:/q; /^From:/h; /./d;g;q'

 # parse out the address proper. Pulls out the e-mail address by itself
 # from the 1-line return address header (see preceding script)
 sed 's/ *(.*)//; s/>.*//; s/.*[:<] *//'

 # add a leading angle bracket and space to each line (quote a message)
 sed 's/^/> /'

 # delete leading angle bracket & space from each line (unquote a message)
 sed 's/^> //'

 # remove most HTML tags (accommodates multiple-line tags)
 sed -e :a -e 's/<[^>]*>//g;/</N;//ba'

 # extract multi-part uuencoded binaries, removing extraneous header
 # info, so that only the uuencoded portion remains. Files passed to
 # sed must be passed in the proper order. Version 1 can be entered
 # from the command line; version 2 can be made into an executable
 # Unix shell script. (Modified from a script by Rahul Dhesi.)
 sed '/^end/,/^begin/d' file1 file2 ... fileX | uudecode   # vers. 1
 sed '/^end/,/^begin/d' "$@" | uudecode                    # vers. 2

 # sort paragraphs of file alphabetically. Paragraphs are separated by blank
 # lines. GNU sed uses \v for vertical tab, or any unique char will do.
 sed '/./{H;d;};x;s/\n/={NL}=/g' file | sort | sed '1s/={NL}=//;s/={NL}=/\n/g'
 gsed '/./{H;d};x;y/\n/\v/' file | sort | sed '1s/\v//;y/\v/\n/'

 # zip up each .TXT file individually, deleting the source file and
 # setting the name of each .ZIP file to the basename of the .TXT file
 # (under DOS: the "dir /b" switch returns bare filenames in all caps).
 echo @echo off >zipup.bat
 dir /b *.txt | sed "s/^\(.*\)\.TXT/pkzip -mo \1 \1.TXT/" >>zipup.bat
```

TYPICAL USE: Sed takes one or more editing commands and applies all of them, in sequence, to each line of input. After all the commands have been applied to the first input line, that line is output and a second input line is taken for processing, and the cycle repeats. The preceding examples assume that input comes from the standard input device (i.e, the console, normally this will be piped input). One or more filenames can be appended to the command line if the input does not come from stdin. Output is sent to stdout (the screen). Thus:

```
 cat filename | sed '10q'       # uses piped input
 sed '10q' filename             # same effect, avoids a useless "cat"
 sed '10q' filename > newfile   # redirects output to disk
```

For additional syntax instructions, including the way to apply editing commands from a disk file instead of the command line, consult "sed & awk, 2nd Edition," by Dale Dougherty and Arnold Robbins (O'Reilly, 1997; http://www.ora.com), "UNIX Text Processing," by Dale Dougherty and Tim O'Reilly (Hayden Books, 1987) or the tutorials by Mike Arst distributed in U-SEDIT2.ZIP (many sites). To fully exploit the power of sed, one must understand "regular expressions." For this, see "Mastering Regular Expressions" by Jeffrey Friedl (O'Reilly, 1997). The manual ("man") pages on Unix systems may be helpful (try "man sed", "man regexp", or the subsection on regular expressions in "man ed"), but man pages are notoriously difficult. They are not written to teach sed use or regexps to first-time users, but as a reference text for those already acquainted with these tools.

QUOTING SYNTAX: The preceding examples use single quotes ('...') instead of double quotes ("...") to enclose editing commands, since sed is typically used on a Unix platform. Single quotes prevent the Unix shell from intrepreting the dollar sign ($) and backquotes (`...`), which are expanded by the shell if they are enclosed in double quotes. Users of the "csh" shell and derivatives will also need to quote the exclamation mark (!) with the backslash (i.e., \!) to properly run the examples listed above, even within single quotes. Versions of sed written for DOS invariably require double quotes ("...") instead of single quotes to enclose editing commands.

USE OF '\t' IN SED SCRIPTS: For clarity in documentation, we have used the expression '\t' to indicate a tab character (0x09) in the scripts. However, most versions of sed do not recognize the '\t' abbreviation, so when typing these scripts from the command line, you should press the TAB key instead. '\t' is supported as a regular expression metacharacter in awk, perl, and HHsed, sedmod, and GNU sed v3.02.80.

VERSIONS OF SED: Versions of sed do differ, and some slight syntax variation is to be expected. In particular, most do not support the use of labels (:name) or branch instructions (b,t) within editing commands, except at the end of those commands. We have used the syntax which will be portable to most users of sed, even though the popular GNU versions of sed allow a more succinct syntax. When the reader sees a fairly long command such as this:

```
   sed -e '/AAA/b' -e '/BBB/b' -e '/CCC/b' -e d
```

it is heartening to know that GNU sed will let you reduce it to:

```
   sed '/AAA/b;/BBB/b;/CCC/b;d'      # or even
   sed '/AAA\|BBB\|CCC/b;d'
```

In addition, remember that while many versions of sed accept a command like "/one/ s/RE1/RE2/", some do NOT allow "/one/! s/RE1/RE2/", which contains space before the 's'. Omit the space when typing the command.

OPTIMIZING FOR SPEED: If execution speed needs to be increased (due to large input files or slow processors or hard disks), substitution will be executed more quickly if the "find" expression is specified before giving the "s/.../.../" instruction. Thus:

```
   sed 's/foo/bar/g' filename         # standard replace command
   sed '/foo/ s/foo/bar/g' filename   # executes more quickly
   sed '/foo/ s//bar/g' filename      # shorthand sed syntax
```

On line selection or deletion in which you only need to output lines from the first part of the file, a "quit" command (q) in the script will drastically reduce processing time for large files. Thus:

```
   sed -n '45,50p' filename           # print line nos. 45-50 of a file
   sed -n '51q;45,50p' filename       # same, but executes much faster
```

```
-------------------------------------------------------------------------
# Single line comments start with a number symbol.

""" Multiline strings can be written
    using three "s, and are often used
    as comments
"""

####################################################
# 1. Primitive Datatypes and Operators
####################################################

# You have numbers
3  # => 3

# Math is what you would expect
1 + 1  # => 2
8 - 1  # => 7
10 * 2  # => 20
35 / 5  # => 7

# Division is a bit tricky. It is integer division and floors the results
# automatically.
5 / 2  # => 2

# To fix division we need to learn about floats.
2.0  # This is a float
11.0 / 4.0  # => 2.75 ahhh...much better

# Result of integer division truncated down both for positive and negative.
5 // 3  # => 1
5.0 // 3.0  # => 1.0 # works on floats too
-5 // 3  # => -2
-5.0 // 3.0  # => -2.0

# Note that we can also import division module(Section 6 Modules)
# to carry out normal division with just one '/'.
from __future__ import division

11 / 4  # => 2.75  ...normal division
11 // 4  # => 2 ...floored division

# Modulo operation
7 % 3  # => 1

# Exponentiation (x to the yth power)
2 ** 4  # => 16

# Enforce precedence with parentheses
(1 + 3) * 2  # => 8

# Boolean Operators
# Note "and" and "or" are case-sensitive
True and False  # => False
False or True  # => True

# Note using Bool operators with ints
0 and 2  # => 0
-5 or 0  # => -5
0 == False  # => True
2 == True  # => False
1 == True  # => True

# negate with not
not True  # => False
not False  # => True

# Equality is ==
1 == 1  # => True
2 == 1  # => False

# Inequality is !=
1 != 1  # => False
2 != 1  # => True

# More comparisons
1 < 10  # => True
1 > 10  # => False
2 <= 2  # => True
2 >= 2  # => True

# Comparisons can be chained!
1 < 2 < 3  # => True
2 < 3 < 2  # => False

# Strings are created with " or '
"This is a string."
'This is also a string.'

# Strings can be added too!
"Hello " + "world!"  # => "Hello world!"
# Strings can be added without using '+'
"Hello " "world!"  # => "Hello world!"

# ... or multiplied
"Hello" * 3  # => "HelloHelloHello"

# A string can be treated like a list of characters
"This is a string"[0]  # => 'T'

# You can find the length of a string
len("This is a string")  # => 16

# String formatting with %
# Even though the % string operator will be deprecated on Python 3.1 and removed
```

```
# later at some time, it may still be good to know how it works.
x = 'apple'
y = 'lemon'
z = "The items in the basket are %s and %s" % (x, y)

# A newer way to format strings is the format method.
# This method is the preferred way
"{} is a {}".format("This", "placeholder")
"{0} can be {1}".format("strings", "formatted")
# You can use keywords if you don't want to count.
"{name} wants to eat {food}".format(name="Bob", food="lasagna")

# None is an object
None  # => None

# Don't use the equality "==" symbol to compare objects to None
# Use "is" instead
"etc" is None  # => False
None is None  # => True

# The 'is' operator tests for object identity. This isn't
# very useful when dealing with primitive values, but is
# very useful when dealing with objects.

# Any object can be used in a Boolean context.
# The following values are considered falsey:
#    - None
#    - zero of any numeric type (e.g., 0, 0L, 0.0, 0j)
#    - empty sequences (e.g., '', (), [])
#    - empty containers (e.g., {}, set())
#    - instances of user-defined classes meeting certain conditions
#      see: https://docs.python.org/2/reference/datamodel.html#object.__nonzero__
#
# All other values are truthy (using the bool() function on them returns True).
bool(0)  # => False
bool("")  # => False


####################################################
# 2. Variables and Collections
####################################################

# Python has a print statement
print "I'm Python. Nice to meet you!"  # => I'm Python. Nice to meet you!

# Simple way to get input data from console
input_string_var = raw_input(
    "Enter some data: ")  # Returns the data as a string
input_var = input("Enter some data: ")  # Evaluates the data as python code
# Warning: Caution is recommended for input() method usage
# Note: In python 3, input() is deprecated and raw_input() is renamed to input()

# No need to declare variables before assigning to them.
some_var = 5  # Convention is to use lower_case_with_underscores
some_var  # => 5

# Accessing a previously unassigned variable is an exception.
# See Control Flow to learn more about exception handling.
some_other_var  # Raises a name error

# if can be used as an expression
# Equivalent of C's '?:' ternary operator
"yahoo!" if 3 > 2 else 2  # => "yahoo!"

# Lists store sequences
li = []
# You can start with a prefilled list
other_li = [4, 5, 6]

# Add stuff to the end of a list with append
li.append(1)  # li is now [1]
li.append(2)  # li is now [1, 2]
li.append(4)  # li is now [1, 2, 4]
li.append(3)  # li is now [1, 2, 4, 3]
# Remove from the end with pop
li.pop()  # => 3 and li is now [1, 2, 4]
# Let's put it back
li.append(3)  # li is now [1, 2, 4, 3] again.

# Access a list like you would any array
li[0]  # => 1
# Assign new values to indexes that have already been initialized with =
li[0] = 42
li[0]  # => 42
li[0] = 1  # Note: setting it back to the original value
# Look at the last element
li[-1]  # => 3

# Looking out of bounds is an IndexError
li[4]  # Raises an IndexError

# You can look at ranges with slice syntax.
# (It's a closed/open range for you mathy types.)
li[1:3]  # => [2, 4]
# Omit the beginning
li[2:]  # => [4, 3]
# Omit the end
li[:3]  # => [1, 2, 4]
# Select every second entry
li[::2]  # =>[1, 4]
# Reverse a copy of the list
li[::-1]  # => [3, 4, 2, 1]
# Use any combination of these to make advanced slices
# li[start:end:step]

# Remove arbitrary elements from a list with "del"
del li[2]  # li is now [1, 2, 3]

# You can add lists
li + other_li  # => [1, 2, 3, 4, 5, 6]
# Note: values for li and for other_li are not modified.

# Concatenate lists with "extend()"
li.extend(other_li)  # Now li is [1, 2, 3, 4, 5, 6]

# Remove first occurrence of a value
li.remove(2)  # li is now [1, 3, 4, 5, 6]
li.remove(2)  # Raises a ValueError as 2 is not in the list

# Insert an element at a specific index
li.insert(1, 2)  # li is now [1, 2, 3, 4, 5, 6] again

# Get the index of the first item found
li.index(2)  # => 1
li.index(7)  # Raises a ValueError as 7 is not in the list

# Check for existence in a list with "in"
1 in li  # => True

# Examine the length with "len()"
len(li)  # => 6

# Tuples are like lists but are immutable.
tup = (1, 2, 3)
tup[0]  # => 1
tup[0] = 3  # Raises a TypeError

# You can do all those list thingies on tuples too
len(tup)  # => 3
tup + (4, 5, 6)  # => (1, 2, 3, 4, 5, 6)
tup[:2]  # => (1, 2)
2 in tup  # => True

# You can unpack tuples (or lists) into variables
a, b, c = (1, 2, 3)  # a is now 1, b is now 2 and c is now 3
d, e, f = 4, 5, 6  # you can leave out the parentheses
# Tuples are created by default if you leave out the parentheses
g = 4, 5, 6  # => (4, 5, 6)
# Now look how easy it is to swap two values
e, d = d, e  # d is now 5 and e is now 4

# Dictionaries store mappings
empty_dict = {}
# Here is a prefilled dictionary
filled_dict = {"one": 1, "two": 2, "three": 3}

# Look up values with []
filled_dict["one"]  # => 1

# Get all keys as a list with "keys()"
filled_dict.keys()  # => ["three", "two", "one"]
# Note - Dictionary key ordering is not guaranteed.
# Your results might not match this exactly.

# Get all values as a list with "values()"
filled_dict.values()  # => [3, 2, 1]
# Note - Same as above regarding key ordering.

# Get all key-value pairs as a list of tuples with "items()"
filled_dict.items()  # => [("one", 1), ("two", 2), ("three", 3)]

# Check for existence of keys in a dictionary with "in"
"one" in filled_dict  # => True
1 in filled_dict  # => False

# Looking up a non-existing key is a KeyError
filled_dict["four"]  # KeyError

# Use "get()" method to avoid the KeyError
filled_dict.get("one")  # => 1
filled_dict.get("four")  # => None
# The get method supports a default argument when the value is missing
filled_dict.get("one", 4)  # => 1
filled_dict.get("four", 4)  # => 4
# note that filled_dict.get("four") is still => None
# (get doesn't set the value in the dictionary)

# set the value of a key with a syntax similar to lists
filled_dict["four"] = 4  # now, filled_dict["four"] => 4

# "setdefault()" inserts into a dictionary only if the given key isn't present
filled_dict.setdefault("five", 5)  # filled_dict["five"] is set to 5
filled_dict.setdefault("five", 6)  # filled_dict["five"] is still 5

# Sets store ... well sets (which are like lists but can contain no duplicates)
empty_set = set()
# Initialize a "set()" with a bunch of values
some_set = set([1, 2, 2, 3, 4])  # some_set is now set([1, 2, 3, 4])

# order is not guaranteed, even though it may sometimes look sorted
another_set = set([4, 3, 2, 2, 1])  # another_set is now set([1, 2, 3, 4])

# Since Python 2.7, {} can be used to declare a set
filled_set = {1, 2, 2, 3, 4}  # => {1, 2, 3, 4}

# Add more items to a set
filled_set.add(5)  # filled_set is now {1, 2, 3, 4, 5}

# Do set intersection with &
other_set = {3, 4, 5, 6}
filled_set & other_set  # => {3, 4, 5}

# Do set union with |
filled_set | other_set  # => {1, 2, 3, 4, 5, 6}

# Do set difference with -
{1, 2, 3, 4} - {2, 3, 5}  # => {1, 4}

# Do set symmetric difference with ^
{1, 2, 3, 4} ^ {2, 3, 5}  # => {1, 4, 5}

# Check if set on the left is a superset of set on the right
{1, 2} >= {1, 2, 3}  # => False

# Check if set on the left is a subset of set on the right
{1, 2} <= {1, 2, 3}  # => True

# Check for existence in a set with in
2 in filled_set  # => True
10 in filled_set  # => False
10 not in filled_set # => True

# Check data type of variable
type(li)   # => list
type(filled_dict)   # => dict
type(5)   # => int


####################################################
#  3. Control Flow
####################################################

# Let's just make a variable
some_var = 5

# Here is an if statement. Indentation is significant in python!
# prints "some_var is smaller than 10"
if some_var > 10:
    print "some_var is totally bigger than 10."
elif some_var < 10:  # This elif clause is optional.
    print "some_var is smaller than 10."
else:  # This is optional too.
    print "some_var is indeed 10."

"""
For loops iterate over lists
prints:
    dog is a mammal
    cat is a mammal
    mouse is a mammal
"""
for animal in ["dog", "cat", "mouse"]:
    # You can use {0} to interpolate formatted strings. (See above.)
    print "{0} is a mammal".format(animal)
```

```
"""
"range(number)" returns a list of numbers
from zero to the given number
prints:
    0
    1
    2
    3
"""
for i in range(4):
    print i

"""
"range(lower, upper)" returns a list of numbers
from the lower number to the upper number
prints:
    4
    5
    6
    7
"""
for i in range(4, 8):
    print i

"""
While loops go until a condition is no longer met.
prints:
    0
    1
    2
    3
"""
x = 0
while x < 4:
    print x
    x += 1  # Shorthand for x = x + 1

# Handle exceptions with a try/except block

# Works on Python 2.6 and up:
try:
    # Use "raise" to raise an error
    raise IndexError("This is an index error")
except IndexError as e:
    pass  # Pass is just a no-op. Usually you would do recovery here.
except (TypeError, NameError):
    pass  # Multiple exceptions can be handled together, if required.
else:  # Optional clause to the try/except block. Must follow all except blocks
    print "All good!"  # Runs only if the code in try raises no exceptions
finally:  # Execute under all circumstances
    print "We can clean up resources here"

# Instead of try/finally to cleanup resources you can use a with statement
with open("myfile.txt") as f:
    for line in f:
        print line


####################################################
# 4. Functions
####################################################

# Use "def" to create new functions
def add(x, y):
    print "x is {0} and y is {1}".format(x, y)
    return x + y  # Return values with a return statement


# Calling functions with parameters
add(5, 6)  # => prints out "x is 5 and y is 6" and returns 11

# Another way to call functions is with keyword arguments
add(y=6, x=5)  # Keyword arguments can arrive in any order.


# You can define functions that take a variable number of
# positional args, which will be interpreted as a tuple by using *
def varargs(*args):
    return args


varargs(1, 2, 3)  # => (1, 2, 3)


# You can define functions that take a variable number of
# keyword args, as well, which will be interpreted as a dict by using **
def keyword_args(**kwargs):
    return kwargs


# Let's call it to see what happens
keyword_args(big="foot", loch="ness")  # => {"big": "foot", "loch": "ness"}


# You can do both at once, if you like
def all_the_args(*args, **kwargs):
    print args
    print kwargs


"""
all_the_args(1, 2, a=3, b=4) prints:
    (1, 2)
    {"a": 3, "b": 4}
"""

# When calling functions, you can do the opposite of args/kwargs!
# Use * to expand positional args and use ** to expand keyword args.
args = (1, 2, 3, 4)
kwargs = {"a": 3, "b": 4}
all_the_args(*args)  # equivalent to all_the_args(1, 2, 3, 4)
all_the_args(**kwargs)  # equivalent to all_the_args(a=3, b=4)
all_the_args(*args, **kwargs)  # equivalent to all_the_args(1, 2, 3, 4, a=3, b=4)


# you can pass args and kwargs along to other functions that take args/kwargs
# by expanding them with * and ** respectively
def pass_all_the_args(*args, **kwargs):
    all_the_args(*args, **kwargs)
    print varargs(*args)
    print keyword_args(**kwargs)


# Function Scope
x = 5


def set_x(num):
    # Local var x not the same as global variable x
    x = num  # => 43
    print x  # => 43

def set_global_x(num):
    global x
    print x  # => 5
    x = num  # global var x is now set to 6
    print x  # => 6


set_x(43)
set_global_x(6)


# Python has first class functions
def create_adder(x):
    def adder(y):
        return x + y

    return adder


add_10 = create_adder(10)
add_10(3)  # => 13

# There are also anonymous functions
(lambda x: x > 2)(3)  # => True
(lambda x, y: x ** 2 + y ** 2)(2, 1)  # => 5

# There are built-in higher order functions
map(add_10, [1, 2, 3])  # => [11, 12, 13]
map(max, [1, 2, 3], [4, 2, 1])  # => [4, 2, 3]

filter(lambda x: x > 5, [3, 4, 5, 6, 7])  # => [6, 7]

# We can use list comprehensions for nice maps and filters
[add_10(i) for i in [1, 2, 3]]  # => [11, 12, 13]
[x for x in [3, 4, 5, 6, 7] if x > 5]  # => [6, 7]

# You can construct set and dict comprehensions as well.
{x for x in 'abcddeef' if x in 'abc'}  # => {'a', 'b', 'c'}
{x: x ** 2 for x in range(5)}  # => {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}


####################################################
# 5. Classes
####################################################

# We subclass from object to get a class.
class Human(object):
    # A class attribute. It is shared by all instances of this class
    species = "H. sapiens"

    # Basic initializer, this is called when this class is instantiated.
    # Note that the double leading and trailing underscores denote objects
    # or attributes that are used by python but that live in user-controlled
    # namespaces. You should not invent such names on your own.
    def __init__(self, name):
        # Assign the argument to the instance's name attribute
        self.name = name

        # Initialize property
        self.age = 0

    # An instance method. All methods take "self" as the first argument
    def say(self, msg):
        return "{0}: {1}".format(self.name, msg)

    # A class method is shared among all instances
    # They are called with the calling class as the first argument
    @classmethod
    def get_species(cls):
        return cls.species

    # A static method is called without a class or instance reference
    @staticmethod
    def grunt():
        return "*grunt*"

    # A property is just like a getter.
    # It turns the method age() into an read-only attribute
    # of the same name.
    @property
    def age(self):
        return self._age

    # This allows the property to be set
    @age.setter
    def age(self, age):
        self._age = age

    # This allows the property to be deleted
    @age.deleter
    def age(self):
        del self._age


# Instantiate a class
i = Human(name="Ian")
print i.say("hi")  # prints out "Ian: hi"

j = Human("Joel")
print j.say("hello")  # prints out "Joel: hello"

# Call our class method
i.get_species()  # => "H. sapiens"

# Change the shared attribute
Human.species = "H. neanderthalensis"
i.get_species()  # => "H. neanderthalensis"
j.get_species()  # => "H. neanderthalensis"

# Call the static method
Human.grunt()  # => "*grunt*"

# Update the property
i.age = 42

# Get the property
i.age  # => 42

# Delete the property
del i.age
i.age  # => raises an AttributeError

####################################################
# 6. Modules
####################################################

# You can import modules
import math

print math.sqrt(16)  # => 4
```

```python
# You can get specific functions from a module
from math import ceil, floor

print ceil(3.7)  # => 4.0
print floor(3.7)  # => 3.0

# You can import all functions from a module.
# Warning: this is not recommended
from math import *

# You can shorten module names
import math as m

math.sqrt(16) == m.sqrt(16)  # => True
# you can also test that the functions are equivalent
from math import sqrt

math.sqrt == m.sqrt == sqrt  # => True

# Python modules are just ordinary python files. You
# can write your own, and import them. The name of the
# module is the same as the name of the file.

# You can find out which functions and attributes
# defines a module.
import math

dir(math)


# If you have a Python script named math.py in the same
# folder as your current script, the file math.py will
# be loaded instead of the built-in Python module.
# This happens because the local folder has priority
# over Python's built-in libraries.


####################################################
# 7. Advanced
####################################################

# Generators
# A generator "generates" values as they are requested instead of storing
# everything up front

# The following method (*NOT* a generator) will double all values and store it
# in `double_arr`. For large size of iterables, that might get huge!
def double_numbers(iterable):
    double_arr = []
    for i in iterable:
        double_arr.append(i + i)
    return double_arr


# Running the following would mean we'll double all values first and return all
# of them back to be checked by our condition
for value in double_numbers(range(1000000)):  # `test_non_generator`
    print value
    if value > 5:
        break


# We could instead use a generator to "generate" the doubled value as the item
# is being requested
def double_numbers_generator(iterable):
    for i in iterable:
        yield i + i


# Running the same code as before, but with a generator, now allows us to iterate
# over the values and doubling them one by one as they are being consumed by
# our logic. Hence as soon as we see a value > 5, we break out of the
# loop and don't need to double most of the values sent in (MUCH FASTER!)
for value in double_numbers_generator(xrange(1000000)):  # `test_generator`
    print value
    if value > 5:
        break

# BTW: did you notice the use of `range` in `test_non_generator` and `xrange` in
`test_generator`?
# Just as `double_numbers_generator` is the generator version of `double_numbers`
# We have `xrange` as the generator version of `range`
# `range` would return back and array with 1000000 values for us to use
# `xrange` would generate 1000000 values for us as we request / iterate over those items

# Just as you can create a list comprehension, you can create generator
# comprehensions as well.
values = (-x for x in [1, 2, 3, 4, 5])
for x in values:
    print(x)  # prints -1 -2 -3 -4 -5 to console/terminal

# You can also cast a generator comprehension directly to a list.
values = (-x for x in [1, 2, 3, 4, 5])
gen_to_list = list(values)
print(gen_to_list)  # => [-1, -2, -3, -4, -5]

# Decorators
# A decorator is a higher order function, which accepts and returns a function.
# Simple usage example – add_apples decorator will add 'Apple' element into
# fruits list returned by get_fruits target function.
def add_apples(func):
    def get_fruits():
        fruits = func()
        fruits.append('Apple')
        return fruits
    return get_fruits

@add_apples
def get_fruits():
    return ['Banana', 'Mango', 'Orange']

# Prints out the list of fruits with 'Apple' element in it:
# Banana, Mango, Orange, Apple
print ', '.join(get_fruits())

# in this example beg wraps say
# Beg will call say. If say_please is True then it will change the returned
# message
from functools import wraps


def beg(target_function):
    @wraps(target_function)
    def wrapper(*args, **kwargs):
        msg, say_please = target_function(*args, **kwargs)
        if say_please:
            return "{} {}".format(msg, "Please! I am poor :(")
        return msg

    return wrapper


@beg
def say(say_please=False):
    msg = "Can you buy me a beer?"
    return msg, say_please


print say()  # Can you buy me a beer?
print say(say_please=True)  # Can you buy me a beer? Please! I am poor :(
```

```bash
#!/bin/bash
# First line of the script is shebang which tells the system how to execute
# the script: http://en.wikipedia.org/wiki/Shebang_(Unix)
# As you already figured, comments start with #. Shebang is also a comment.

# Simple hello world example:
echo Hello world! # => Hello world!

# Each command starts on a new line, or after semicolon:
echo 'This is the first line'; echo 'This is the second line'
# => This is the first line
# => This is the second line

# Declaring a variable looks like this:
Variable="Some string"

# But not like this:
Variable = "Some string" # => returns error "Variable: command not found"
# Bash will decide that Variable is a command it must execute and give an error
# because it can't be found.

# Or like this:
Variable= 'Some string' # => returns error: "Some string: command not found"
# Bash will decide that 'Some string' is a command it must execute and give an
# error because it can't be found. (In this case the 'Variable=' part is seen
# as a variable assignment valid only for the scope of the 'Some string'
# command.)

# Using the variable:
echo $Variable # => Some string
echo "$Variable" # => Some string
echo '$Variable' # => $Variable
# When you use the variable itself — assign it, export it, or else — you write
# its name without $. If you want to use the variable's value, you should use $.
# Note that ' (single quote) won't expand the variables!

# Parameter expansion ${ }:
echo ${Variable} # => Some string
# This is a simple usage of parameter expansion
# Parameter Expansion gets a value from a variable.  It "expands" or prints the value
# During the expansion time the value or parameter are able to be modified
# Below are other modifications that add onto this expansion

# String substitution in variables
echo ${Variable/Some/A} # => A string
# This will substitute the first occurrence of "Some" with "A"

# Substring from a variable
Length=7
echo ${Variable:0:Length} # => Some st
# This will return only the first 7 characters of the value

# Default value for variable
echo ${Foo:-"DefaultValueIfFooIsMissingOrEmpty"}
# => DefaultValueIfFooIsMissingOrEmpty
# This works for null (Foo=) and empty string (Foo=""); zero (Foo=0) returns 0.
# Note that it only returns default value and doesn't change variable value.

# Declare an array with 6 elements
array0=(one two three four five six)
# Print first element
echo $array0 # => "one"
# Print first element
echo ${array0[0]} # => "one"
# Print all elements
echo ${array0[@]} # => "one two three four five six"
# Print number of elements
echo ${#array0[@]} # => "6"
# Print number of characters in third element
echo ${#array0[2]} # => "5"
# Print 2 elements starting from forth
echo ${array0[@]:3:2} # => "four five"
# Print all elements. Each of them on new line.
for i in "${array0[@]}"; do
    echo "$i"
done

# Brace Expansion { }
# Used to generate arbitrary strings
echo {1..10} # => 1 2 3 4 5 6 7 8 9 10
echo {a..z} # => a b c d e f g h i j k l m n o p q r s t u v w x y z
# This will output the range from the start value to the end value

# Builtin variables:
# There are some useful builtin variables, like
echo "Last program's return value: $?"
echo "Script's PID: $$"
echo "Number of arguments passed to script: $#"
echo "All arguments passed to script: $@"
echo "Script's arguments separated into different variables: $1 $2..."

# Now that we know how to echo and use variables,
# let's learn some of the other basics of bash!

# Our current directory is available through the command `pwd`.
# `pwd` stands for "print working directory".
# We can also use the builtin variable `$PWD`.
# Observe that the following are equivalent:
echo "I'm in $(pwd)" # execs `pwd` and interpolates output
echo "I'm in $PWD" # interpolates the variable

# If you get too much output in your terminal, or from a script, the command
# `clear` clears your screen
clear
# Ctrl-L also works for clearing output

# Reading a value from input:
echo "What's your name?"
read Name # Note that we didn't need to declare a new variable
echo Hello, $Name!

# We have the usual if structure:
# use 'man test' for more info about conditionals
if [ $Name != $USER ]
then
    echo "Your name isn't your username"
else
```

```
    echo "Your name is your username"
fi
# True if the value of $Name is not equal to the current user's login username

# NOTE: if $Name is empty, bash sees the above condition as:
if [ != $USER ]
# which is invalid syntax
# so the "safe" way to use potentially empty variables in bash is:
if [ "$Name" != $USER ] ...
# which, when $Name is empty, is seen by bash as:
if [ "" != $USER ] ...
# which works as expected

# There is also conditional execution
echo "Always executed" || echo "Only executed if first command fails"
# => Always executed
echo "Always executed" && echo "Only executed if first command does NOT fail"
# => Always executed
# => Only executed if first command does NOT fail


# To use && and || with if statements, you need multiple pairs of square brackets:
if [ "$Name" == "Steve" ] && [ "$Age" -eq 15 ]
then
    echo "This will run if $Name is Steve AND $Age is 15."
fi

if [ "$Name" == "Daniya" ] || [ "$Name" == "Zach" ]
then
    echo "This will run if $Name is Daniya OR Zach."
fi

# Redefine command 'ping' as alias to send only 5 packets
alias ping='ping -c 5'
# Escape alias and use command with this name instead
\ping 192.168.1.1
# Print all aliases
alias -p

# Expressions are denoted with the following format:
echo $(( 10 + 5 )) # => 15

# Unlike other programming languages, bash is a shell so it works in the context
# of a current directory. You can list files and directories in the current
# directory with the ls command:
ls # Lists the files and subdirectories contained in the current directory

# These commands have options that control their execution:
ls -l # Lists every file and directory on a separate line
ls -t # Sorts the directory contents by last-modified date (descending)
ls -R # Recursively `ls` this directory and all of its subdirectories

# Results of the previous command can be passed to the next command as input.
# grep command filters the input with provided patterns. That's how we can list
# .txt files in the current directory:
ls -l | grep "\.txt"

# Use `cat` to print files to stdout:
cat file.txt

# We can also read the file using `cat`:
Contents=$(cat file.txt)
echo "START OF FILE\n$Contents\nEND OF FILE" # "\n" prints a new line character
# => START OF FILE
# => [contents of file.txt]
# => END OF FILE

# Use `cp` to copy files or directories from one place to another.
# `cp` creates NEW versions of the sources,
# so editing the copy won't affect the original (and vice versa).
# Note that it will overwrite the destination if it already exists.
cp srcFile.txt clone.txt
cp -r srcDirectory/ dst/ # recursively copy

# Look into `scp` or `sftp` if you plan on exchanging files between computers.
# `scp` behaves very similarly to `cp`.
# `sftp` is more interactive.

# Use `mv` to move files or directories from one place to another.
# `mv` is similar to `cp`, but it deletes the source.
# `mv` is also useful for renaming files!
mv s0urc3.txt dst.txt # sorry, l33t hackers...

# Since bash works in the context of a current directory, you might want to
# run your command in some other directory. We have cd for changing location:
cd ~    # change to home directory
cd ..   # go up one directory
        # (^^say, from /home/username/Downloads to /home/username)
cd /home/username/Documents   # change to specified directory
cd ~/Documents/..    # still in home directory..isn't it??

# Use subshells to work across directories
(echo "First, I'm here: $PWD") && (cd someDir; echo "Then, I'm here: $PWD")
pwd # still in first directory

# Use `mkdir` to create new directories.
mkdir myNewDir
# The `-p` flag causes new intermediate directories to be created as necessary.
mkdir -p myNewDir/with/intermediate/directories
# if the intermediate directories didn't already exist, running the above
# command without the `-p` flag would return an error

# You can redirect command input and output (stdin, stdout, and stderr).
# Read from stdin until ^EOF$ and overwrite hello.py with the lines
# between "EOF":
cat > hello.py << EOF
#!/usr/bin/env python
from __future__ import print_function
import sys
print("#stdout", file=sys.stdout)
print("#stderr", file=sys.stderr)
for line in sys.stdin:
    print(line, file=sys.stdout)
EOF

# Run the hello.py Python script with various stdin, stdout, and
# stderr redirections:
python hello.py < "input.in" # pass input.in as input to the script
python hello.py > "output.out" # redirect output from the script to output.out
python hello.py 2> "error.err" # redirect error output to error.err
python hello.py > "output-and-error.log" 2>&1 # redirect both output and errors to
output-and-error.log
python hello.py > /dev/null 2>&1 # redirect all output and errors to the black hole,
/dev/null, i.e., no output
# The output error will overwrite the file if it exists,
# if you want to append instead, use ">>":
python hello.py >> "output.out" 2>> "error.err"

# Overwrite output.out, append to error.err, and count lines:
info bash 'Basic Shell Features' 'Redirections' > output.out 2>> error.err
wc -l output.out error.err

# Run a command and print its file descriptor (e.g. /dev/fd/123)
# see: man fd
echo <(echo "#helloworld")

# Overwrite output.out with "#helloworld":
cat > output.out <(echo "#helloworld")
echo "#helloworld" > output.out
echo "#helloworld" | cat > output.out
echo "#helloworld" | tee output.out >/dev/null

# Cleanup temporary files verbosely (add '-i' for interactive)
# WARNING: `rm` commands cannot be undone
rm -v output.out error.err output-and-error.log
rm -r tempDir/ # recursively delete

# Commands can be substituted within other commands using $( ):
# The following command displays the number of files and directories in the
# current directory.
echo "There are $(ls | wc -l) items here."

# The same can be done using backticks `` but they can't be nested - the preferred way
# is to use $( ).
echo "There are `ls | wc -l` items here."

# Bash uses a case statement that works similarly to switch in Java and C++:
case "$Variable" in
    #List patterns for the conditions you want to meet
    0) echo "There is a zero.";;
    1) echo "There is a one.";;
    *) echo "It is not null.";;
esac

# for loops iterate for as many arguments given:
# The contents of $Variable is printed three times.
for Variable in {1..3}
do
    echo "$Variable"
done
# => 1
# => 2
# => 3


# Or write it the "traditional for loop" way:
for ((a=1; a <= 3; a++))
do
    echo $a
done
# => 1
# => 2
# => 3

# They can also be used to act on files..
# This will run the command 'cat' on file1 and file2
for Variable in file1 file2
do
    cat "$Variable"
done

# ..or the output from a command
# This will cat the output from ls.
for Output in $(ls)
do
    cat "$Output"
done

# while loop:
while [ true ]
do
    echo "loop body here..."
    break
done
# => loop body here...

# You can also define functions
# Definition:
function foo ()
{
    echo "Arguments work just like script arguments: $@"
    echo "And: $1 $2..."
    echo "This is a function"
    return 0
}
# Call the function `foo` with two arguments, arg1 and arg2:
foo arg1 arg2
# => Arguments work just like script arguments: arg1 arg2
# => And: arg1 arg2...
# => This is a function

# or simply
bar ()
{
    echo "Another way to declare functions!"
    return 0
}
# Call the function `bar` with no arguments:
bar # => Another way to declare functions!

# Calling your function
foo "My name is" $Name

# There are a lot of useful commands you should learn:
# prints last 10 lines of file.txt
tail -n 10 file.txt

# prints first 10 lines of file.txt
head -n 10 file.txt

# sort file.txt's lines
sort file.txt

# report or omit repeated lines, with -d it reports them
uniq -d file.txt

# prints only the first column before the ',' character
cut -d ',' -f 1 file.txt

# replaces every occurrence of 'okay' with 'great' in file.txt
# (regex compatible)
sed -i 's/okay/great/g' file.txt

# print to stdout all lines of file.txt which match some regex
# The example prints lines which begin with "foo" and end in "bar"
grep "^foo.*bar$" file.txt

# pass the option "-c" to instead print the number of lines matching the regex
grep -c "^foo.*bar$" file.txt

# Other useful options are:
grep -r "^foo.*bar$" someDir/ # recursively `grep`
grep -n "^foo.*bar$" file.txt # give line numbers
grep -rI "^foo.*bar$" someDir/ # recursively `grep`, but ignore binary files
```

```
# perform the same initial search, but filter out the lines containing "baz"
grep "^foo.*bar$" file.txt | grep -v "baz"

# if you literally want to search for the string,
# and not the regex, use fgrep (or grep -F)
fgrep "foobar" file.txt

# The trap command allows you to execute a command whenever your script
# receives a signal. Here, trap will execute `rm` if it receives any of the
# three listed signals.
trap "rm $TEMP_FILE; exit" SIGHUP SIGINT SIGTERM

# `sudo` is used to perform commands as the superuser
NAME1=$(whoami)
NAME2=$(sudo whoami)
echo "Was $NAME1, then became more powerful $NAME2"

# Read Bash shell builtins documentation with the bash 'help' builtin:
help
help help
help for
help return
help source
help .

# Read Bash manpage documentation with man
apropos bash
man 1 bash
man bash

# Read info documentation with info (? for help)
apropos info | grep '^info.*('
man info
info info
info 5 info

# Read bash info documentation:
info bash
info bash 'Bash Features'
info bash 6
info --apropos bash


================================================================================
CS35L-FinalReview

Code: C, Python, Bash
Man page: sed, tr, grep, find, sort, diff, od, strace, time

Assignment 1
-Basic commands
            pwd:  print working directory
            rmdir: remove a directory
  mv: move/rename a file(no undo)
            ln: create a link
  ls -d: list only directories
            touch: update access & modification time to current time/create a file
            find:
    type, perm(permission), name, prune(don't descend)
    $ find -perm -g=w -> match all files with g=w
    $ find -perm g=w -> match files with only g=w
    $ find -perm /g=w, u=w -> if no perm bits are set, -000
            which: shows the full path of shell commands
            ps: report a snapshot of current process
            kill: terminate a process
    $ kill [-s signal|-p] [--] pid...
    $ kill -l [signal]
    $ kill -s SIGHUP 24837

Shell
-How do i find where files are on the system?
  $ whereis
    locate the binary, source, and manual page files for a command
  $ which
-How do i find out what options are available for an utility?
  $ man
-When is a file a file and when is it a process?
-What types of links are there?
  Hard links: points to physical data
  Symbolic links: points to a file
What are the differences between absolute and relative paths?
  Absolute path = pwd + relative path
Why do we have permissions?
  filetype user group other
           rwx   rwx   rwx

Assignment 2
What else does the locale affect? sort
Why do we have environment variables? What functions do they serve?
  Environment variables help programs know what directory to install files in, where to
store temporary files, and where to find user profile settings. They help shape the
environment in which programs run on your computer.

When do we use a compiled language/interpreted language?
Touted benefits of interpreters:
  No compilation means the time from editing code to testing the app can be diminished
  No need to generate binaries for multiple architectures because the interpreter will
manage the architecture abstraction (though you may need to still worry about the
scripts handling integer sizes correctly, just not the binary distribution)
Touted benefits of compilers:
  Compiled native code does not have the overhead of an interpreter and is therefore
usually more efficient on time and space
  Interoperability is usually better, the only way for in-proc interoperation with
scripts is via an interpreter rather than a standard FFI
  Ability to support architectures the interpreter hasn't been compiled for (such as
embedded systems)

Why to redirect I/O? What are some examples of use cases for I/O redirection? How do we
implement I/O redirection in C?

Why do regular expressions exist no just program  our own text searching? Are the
expressions the same across languages, platforms?
Available across so many languages that most developers will learn them sooner or later.

How do I write a regular expression to accomplish x?

Quantification
            How many times of previous expressions?
Grouping
            Which subset of previous expressions?
Alternation
            Which choices?
Anchors
            Where?

How to combine the above to accomplish tasks?

Practice!
Use egrep
$ egrep [^a-z]{7}   -> seven characters not in lower case
What are the differences between tr, sed, and grep?
tr command Translate, squeeze, delete characters from standard input, writing to
standard output. on the other hand sed is a stream editor or it is used to perform basic
text transformations on an input stream. tr perform character based transformation but
sed perform string based transformation. grep can only shows the lines matching pattern.
You can not transform the output. sed command is super set of grep, tr and cut command.
Additionally it can transform output. sed is mainly used for extraction and
substitution.

Assignment 3
Why do we have compilation process?
Work at low level
preprocessing->compilation->assembly->linking
What are the different components of the process?
Why can't I execute individual object code files?
The object code generated in the assembly stage is composed of machine instructions that
the processor understands but some pieces of the program are out of order or missing. To
produce an executable program, the existing pieces have to be rearranged and the missing
ones filled in. This process is called linking.
What are the differences between open source and closed source software? When would I
want to use one or the other?
With closed source software, the source code is closely guarded, often because it's
considered a trade secret that creates scarcity and keeps the organization competitive.
Such programs come with restrictions against modifying the software or using it in ways
untended by the original creators. In principle, open source software means the source
code is made available on a universal level. There are some variations on this — some
producers do restrict who can access or modify the code — generally, however, the idea
is to open up the software to the public, creating a mass collaboration that results in
the software being constantly updated, fixed, improved, and expanded on.
Why do we have 'make'? Why don't we just run gcc from terminal?
The purpose of a makefile is to be easily build an executable that might take many
commands to create (which would be a pain to compile over and over again manually).
Why not just change the original source code to fix it? Why do we have patches?
Patches make development easier, because instead of supplying a replacement file,
possibly consisting of thousands of lines of code, the patch includes only the exact
changes that were made. In effect, a patch is a list of all the changes made to a file,
which can be used to re-create those changes on another copy of that file.


Assignment 5
Why are these system calls and not just regular library functions?
read, write, open, close, fstat
System Calls are a way for user programs (running in user mode) to request some service
from Operating System. In other words, system calls allow the user programs to ask OS to
do some stuff on behalf of the user program. User programs are not allowed to work
directly with them since there is a likelihood of our programs corrupting important data
structures in kernel. They are best managed by the OS itself which takes care of
locking, recovery/cleanup and other things that guarantee that data structures are
accessed in correct manner.

Assignment 6
Why are the benefits of single core having illusion of parallelism by switching
processes quickly?
You are able to multitask because modern processors use "multithreading", which allows
you to perform concurrent processing, even if only a single core processor. The more
cores you have, the better the work load is divided up. Each core can execute only one
instruction at a time, however it's so fast it seems as if you are running multiple
threads simultaneously.

What can go wrong in multithreading?
            What are race conditions?
            What is deadlock?

What are some approaches to make multithreading safer? What are the possible advantages
or disadvantgaes of each of these approaches?
Locks:
Synchronizing the critical sections. When you do that, only 1 thread can work with this
part at 1 time, but take care when you use that, only for the source code which all
thread can't run together. This one will make the performance go down.

Use immutable objects. Immutable objects are simply objects whose state (the object's
data) can't change after construction.

Use thread-safe wrappers. This means you put the main class (which isn't thread-safe)
inside a new class that is thread-safe. This way is helpful when the main class from
third party or can't update.

Assignment 8
How are libraries dynamically linked?
What are the advantages and disadvantages of dynamic linking (as opposed to static
linking)?
How does the building process change with dynmaic linking? Dynamic linking at runtime
instead of compile-time.
What computational or data overhead could dynamic linking require?
Load so, resolve address, missing dynamic lib, wrong lib. A DLL can start to pay off
when it is already loaded in another process. Now the code pages of the DLL are simply
shared, startup overhead is very low and memory usage is efficient.

Assignment 9
Why bother with source control?
What are the strengths and weakness of source control?
What would i want to use it? How do I use it?
What is a branch?
Branching means you diverge from the main line of development and continue to do work
without messing with that main line.
What is the difference between a working copy and the repository?
Check out is the term used to describe the process of making a copy of a project from a
repository into your local file system. This checked out copy is called a working copy.

What is a commit? What should be in a commit?
With Git, every time you commit, or save the state of your project, Git basically takes
a picture of what all your files look like at that moment and stores a reference to that
snapshot. Typically, you'll want to start making changes and committing snapshots of
those changes into your repository each time the project reaches a state you want to
record. Commit message.
Stores the current contents of the index in a new commit along with a log message from
the user describing the changes.

How many files could commits contain?
Files added from $ git add .
(modified files)

Why bother having branches? Why can't we just all work on the same single master branch?
What happens when we perform a merge? How does it work?
Join two or more development histories together. Incorporates changes from the named
commits (since the time their histories diverged from the current branch) into the
current branch. This command is used by git pull to incorporate changes from another
repository and can be used by hand to merge changes from one branch into another.
Suppose you've decided that your issue #53 work is complete and ready to be merged into
your master branch. In order to do that, you'll merge your iss53 branch into master,
much like you merged your hotfix branch earlier. All you have to do is check out the
branch you wish to merge into and then run the git merge command:
 $ git checkout master
  Switched to branch 'master'
  $ git merge iss53
  Merge made by the 'recursive' strategy.
  index.html |    1 +
  1 file changed, 1 insertion(+)
Instead of just moving the branch pointer forward, Git creates a new snapshot that
results from this three-way merge and automatically creates a new commit that points to
it. This is referred to as a merge commit, and is special in that it has more than one
parent.
```

Now that your work is merged in, you have no further need for the iss53 branch. You can close the ticket in your ticket-tracking system, and delete the branch:

Assignment 2

*****Part 1******
Execute the following commands and send the outputs into six text files:
```
$ cat assign2.html | tr -c 'A-Za-z' '[\n*]'
```
Replace non-letter characters with new lines.
```
$ cat assign2.html | tr -cs 'A-Za-z' '[\n*]'
```
Replace non-letter characters with new lines and replace a sequence of repeated characters (in this case, new lines) with a single occurence.
```
$ cat assign2.html | tr -cs 'A-Za-z' '[\n*]' | sort
```
Sort the previous command's output.
```
$ cat assign2.html | tr -cs 'A-Za-z' '[\n*]' | sort -u
```
Remove any duplicates from the previous command's output.
```
$ cat assign2.html | tr -cs 'A-Za-z' '[\n*]' | sort -u | comm - words
```
Compare the previous command's output with the list of English words.
Column 1 contains lines unique to assign2.html, column 2 contains lines unique to words, column 3 contains lines common to both files.
```
$ cat assign2.html | tr -cs 'A-Za-z' '[\n*]' | sort -u | comm -23 - words
```
Output the words that are unique to assign2.html and not in words.

*****Part 2******
```
$ wget http://mauimapp.com/moolelo/hwnwdseng.htm
```
Obtain a copy of the web page.

Inside buildwords:
```
#! /bin/sh
# case-insensitive
tr [:upper:] [:lower:] |
```
Translate all upper case letters to lower case letters.
```
tr [\`] \'  |
```
Translate all grave accents to apostrophes.
```
sed 's/?//g' |
```
Remove questions marks.
```
sed 's/^ *//g' |
```
Remove leading whitespaces in each line.
```
sed -n '/<tr>/ {s/.*//; N; N; s/\n//g; p;}' |
```
Delete newlines after every <tr>
```
sed 's/<td>.*<td>//' |
```
English words are between <td> and <td>, remove them.
```
sed 's/<[^>]*.//g' |
```
Delete html brackets.
```
grep ^[pk\'mnwlhaeiou] |
```
Pick words that start with Hawaiian letters.
```
tr , '\n' |
```
Entries with commas contain multiple words.
```
tr - '\n' |
```
Entries with hyphens contain multiple words.
```
sed 's/ /\n/g' |
```
Entries with spaces contain multiple words.
```
sed "/[^pk\'mnwlhaeiou]/d" |
```
Remove improperly formatted hawaiian words
```
sed '/^\s*$/d' |
```
Remove extra empty lines
```
sort -u
```
Sort the words and remove duplicates.

```
$ cat hwnwdseng.htm | ./buildwords > hwords
```
Write the Hawaiian words into hwords.

```
$ cat assign2.html | tr [:upper:] [:lower:] | tr -cs 'A-Za-z' '[\n*]' | sort -u | comm -23 - words > me
```
Running English spelling checker on assign2.html.
```
$ cat assign2.html | tr [:upper:] [:lower:] | tr -cs 'A-Za-z' '[\n*]' | sort -u | comm -23 - hwords > mh1
```
Running Hawaiian spelling checker on assign2.html, look at all words.
```
$ cat assign2.html | tr [:upper:] [:lower:] | tr -cs "pk\'mnwlhaeiou" '[\n*]' | sort -u | comm -23 - hwords > mh2
```
Running Hawaiian spelling checker on assign2.html, and only look at the valid Hawaiian words.
```
$ cat hwords | tr [:upper:] [:lower:] | tr -cs 'A-Za-z' '[\n*]' | sort -u | comm -23 - hwords > hh1
```
Since tr -cs command replaces all apostrophes, the spelling check generate 45 mispelled words to hh1.
```
$ cat hwords | tr [:upper:] [:lower:] | sort -u | comm -23 - hwords > hh2
```
Running Hawaiian spelling checker on hwords. hh2 is empty because hwords is the dictionary itself.
```
$ diff me mh > m.diff
```
Find the difference between misspelled words in English and misspelled words in Hawaiian.

By examining me, mh1, and mh2:
me: 38 misspelled in assign2 as English
mh1: 405 misspelled in assign2 as Hawaiian if improperly formatted English words are considered misspelled Hawaiian words.
mh2: 196 misspelled in assign2 as Hawaiian if we look at words with only Hawaiian letters.
```
$ comm -12 me hwords
```
Misspelled as English but not as Hawaiian: wiki, lau, halau.
```
$ comm -12 mh1 words
```
Misspelled as Hawaiian but not as English (370):

```
#! /bin/sh
# sameln
# Declare variables
declare -a a
D=$1

IFS=$'\n'
files=$(find $D -maxdepth 1 -type f | sort)
#filesWithDot=$(ls -a $D | grep ^[.] | sort)


# Add files into the array of files
i=0

# Add other files
for f in $files
do
    a[$i]="$f"
    ((i++))
done

# Report the error if the file is not readable
for ((j=0;j<i;j++))
do
    if [ ! -r "${a[$j]}" ]
    then
        echo "ERROR: ${a[$j]} is not readable"
    fi
done


for ((k=0;k<i;k++))
do
            if [[ -L ${a[$k]} ]]
            then
                continue
            fi

    for ((j=k+1;j<i;j++))
    do
        # Only look at the readable regular files
        if [[ -r "${a[$j]}" && -f "${a[$j]}" && -r "${a[$k]}" && -f "${a[$k]}" ]]
        then
            # Find out if the files are duplicates
                if [[ "${a[$k]}"!="${a[$j]}" ]]
                then

            cmp -s "${a[$j]}" "${a[$k]}"
            if [ $? -eq 0 ]
            then
                # Replace the duplicate with hard links to the first file
                echo "Replace ${a[$j]} with ${a[$k]}"
                                echo "Remove ${a[$j]}"
                ln -fP "${a[$k]}" "${a[$j]}"
            fi
                fi
        fi
    done
done
unset IFS
```

Assignment 3
hw3.txt

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.)

```
$ python3 ./randline.py -n 1 assign1.html
  File "./randline.py", line 65
    except IOError as (errno, strerror):
                      ^
SyntaxError: invalid syntax
```

There is a syntax error when handling exceptions due to the difference between python2 and python3. In python3, the correct syntax could be:
```
            except IOError as e:
        parser.error("I/O error({0}): {1}".
                     format(e.errno, e.strerror))
```

```
#!/usr/bin/python
import random, sys
from optparse import OptionParser
import string

class perm:
  def __init__(self, filename):
    if type(filename) is str:
      if filename == "-":
        self.lines = sys.stdin.readlines()
      else:
        f = open(filename, 'r')
        self.lines = f.readlines()
        f.close()
    else:
      self.lines = filename
  def choosePerm(self):
    random.shuffle(self.lines)
    return self.lines
  def chooseRand(self):
    if len(self.lines) != 0:
      return random.choice(self.lines)
    return None

def main():
  version_msg = "%prog 1.0"
  usage_msg = """%prog [OPTION]... FILE

Write a random permutation of the input lines to standard output."""


  parser = OptionParser(version = version_msg, usage = usage_msg)
  parser.add_option("-e", "--echo", action = "store_true", dest = "echo",
                    default = False,
                    help = "treat each ARG as an input file")
  parser.add_option("-n", "--head-count", action = "store", dest = "count",
                    help = "output at most COUNT lines")
  parser.add_option("-r", "--repeat", action = "store_true", dest = "repeat",
                    default = False,
                    help = "output lines can be repeated")
  options, args = parser.parse_args(sys.argv[1:])

  count = 0
  limit = True
  if options.count ==  None:
    limit = False
  else:
    count = int(options.count)
  echo = bool(options.echo)
  repeat = bool(options.repeat)

  if limit and count < 0:
    parser.error("negative count: {0}".format(count))

  if echo:
    input_file = [None] * len(args)
    for index in range(len(args)):
      input_file[index] = args[index] + "\n"
  elif len(args) == 1:
    input_file = args[0]
  elif len(args) > 1:
    parser.error("wrong number of operands")
  else:
    input_file = sys.stdin.readlines()

  try:
    generator = perm(input_file)
    output = []
    if repeat:
      if generator.chooseRand() == None:
        parser.error("no lines to repeat")
    else:
```

```
      output = generator.choosePerm()
      if not limit or len(output) < count:
        count = len(output)
      for index in range(count):
        sys.stdout.write(output[index])
  except IOError as e:
    parser.error("I/O error({0}): {1}".format(e.errno, e.strerror))


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

```
#!/usr/bin/python
import random, sys
from optparse import OptionParser
class randline:
    def __init__(self, filename):
        f = open(filename, 'r')
        self.lines = f.readlines()
        f.close()
def chooseline(self):
        return random.choice(self.lines)
def main():
    version_msg = "%prog 2.0"
    usage_msg = """%prog [OPTION]... FILE
Output randomly selected lines from FILE."""
parser = OptionParser(version=version_msg,
                      usage=usage_msg)
    parser.add_option("-n", "--numlines",
                      action="store", dest="numlines", default=1,
                      help="output NUMLINES lines (default 1)")

    options, args = parser.parse_args(sys.argv[1:])
try:
        numlines = int(options.numlines)
    except:
        parser.error("invalid NUMLINES: {0}".
                     format(options.numlines))
    if numlines < 0:
        parser.error("negative count: {0}".
                     format(numlines))
    if len(args) != 1:
        parser.error("wrong number of operands")
    input_file = args[0]
try:
        generator = randline(input_file)
        for index in range(numlines):
            sys.stdout.write(generator.chooseline())
    except IOError as (errno, strerror):
        parser.error("I/O error({0}): {1}".
                     format(errno, strerror))
if __name__ == "__main__":
    main()
```

```
lab4.txt
================================================================================
1. 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?
$ wget https://web.cs.ucla.edu/classes/spring18/cs35L/assign/
coreutils-with-bug.tar.gz
$ tar xvf coreutils-with-bug.tar.gz
            Fetch the old version of coreutils.

Install the old version of coreutils:
$ cd
$ mkdir lab4
$ realpath lab4
/w/home.20/cs/ugrad/huimin/lab4
            Create a directory to install coreutils.

$ cd ~/coreutils-with-bug
$ ./configure --prefix=/w/home.20/cs/ugrad/huimin/lab4
$ make
In file included from utimecmp.c:41:0:
utimens.h:2:5: error: conflicting types for 'futimens'
 int futimens (int, char const *, struct timespec const [2]);
     ^~~~~~~~
In file included from utimecmp.h:25:0,
                 from utimecmp.c:25:
/usr/include/sys/stat.h:373:12: note: previous declaration of 'futimens' was
here
 extern int futimens (int __fd, const struct timespec __times[2]) __THROW;
            ^~~~~~~
make[3]: *** [Makefile:659: utimecmp.o] Error 1
make[3]: Leaving directory
'/w/home.20/cs/ugrad/huimin/35LA4/coreutils-with-bug/lib'
make[2]: *** [Makefile:414: all] Error 2
make[2]: Leaving directory
'/w/home.20/cs/ugrad/huimin/35LA4/coreutils-with-bug/lib'
make[1]: *** [Makefile:419: all-recursive] Error 1
make[1]: Leaving directory
'/w/home.20/cs/ugrad/huimin/35LA4/coreutils-with-bug'
make: *** [Makefile:357: all] Error 2
            Encounter an error in utimens.h and stat.h.

$ make clean

Apply the patch:
$ wget https://web.cs.ucla.edu/classes/spring18/cs35L/assign/coreutils.diff
$ patch -p0 < coreutils.diff
patching file lib/utimens.c
patching file lib/utimens.h
patching file src/copy.c
patching file src/tee.c
patching file src/touch.c
$ ./configure --prefix=/w/home.20/cs/ugrad/huimin/lab4
$ make
$ make install
$ ls ~/lab4
bin  share

I could not build the old version of coreutils as-is, as I encountered the
error which states that the previous declaration in stat.h for the function
'futimens' has a conflict with the function 'futimens' in utimens.h. The old
version of coreutils is outdated (version 5.93), so it might contain functions
that conflict with the updated C standards. Therefore, the patch renames
'futimens' and some other functions by adding prefix 'coreutils_' and corrects
the places where these functions are called, such as in touch.c, copy.c, and
main.


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

$ cd ~/lab4/bin
$ tmp=$(mktemp -d)
$ cd $tmp
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ TZ=UTC0 ~/lab4/bin/ls -lt --full-time wwi-armistice now now1
-rw-r--r-- 1 huimin csugrad 0 1918-11-11 11:00:00.000000000 +0000 wwi-armistice
-rw-r--r-- 1 huimin csugrad 0 2018-05-02 23:53:38.750511073 +0000 now1
-rw-r--r-- 1 huimin csugrad 0 2018-05-02 23:53:33.233417906 +0000 now

I also tried to use the installed version of coreutils:
$ TZ=UTC0 ls -lt --full-time wwi-armistice now now1
-rw-r--r-- 1 huimin csugrad 0 2018-05-02 23:53:38.750511073 +0000 now1
-rw-r--r-- 1 huimin csugrad 0 2018-05-02 23:53:33.233417906 +0000 now
-rw-r--r-- 1 huimin csugrad 0 1918-11-11 11:00:00.000000000 +0000 wwi-armistice
It works correctly without the bug.


Inside $tmp, I use a debugger to figure out the buggy source file:
$ gdb ~/lab4/bin/ls
(gdb) run -lt --full-time wwi-armistice now
            Running ls with wwi-artimistice with one other file
(gdb) break sort_files
Breakpoint 1 at 0x4042f0: file ls.c, line 2954.
(gdb) run -lt --full-time wwi-armistice now
Starting program: /w/home.20/cs/ugrad/huimin/lab4/bin/ls -lt --full-time wwi-armistice
now

Breakpoint 1, sort_files () at ls.c:2954
2954        {
(gdb) step
2962          if (! setjmp (failed_strcoll))
(gdb) step
2964              switch (sort_type)
(gdb) step
2969                      switch (time_type)
(gdb) step
2975                          func = sort_reverse ? rev_cmp_mtime : compare_mtime;
(gdb) break compare_mtime
Breakpoint 2 at 0x4067c0: file ../lib/timespec.h, line 48.
(gdb) disable 1
(gdb) run -lt --full-time wwi-armistice now
Starting program: /w/home.20/cs/ugrad/huimin/lab4/bin/ls -lt --full-time wwi-armistice
now

Breakpoint 2, compare_mtime (a=0x6170d0, b=0x617180) at ls.c:2884
2884        static int compare_mtime (V a, V b) { return cmp_mtime (a, b, xstrcoll); }
(gdb) step
cmp_mtime (cmp=0x4045f0 <xstrcoll>, b=0x617180, a=0x6170d0) at ls.c:2884
2884        static int compare_mtime (V a, V b) { return cmp_mtime (a, b, xstrcoll); }
(gdb) step
timespec_cmp (b=..., a=...) at ../lib/timespec.h:48
48            int diff = a.tv_sec - b.tv_sec;
(gdb) print diff
$1 = -1155827283
(gdb) list compare_time
43          /* Return negative, zero, positive if A < B, A == B, A > B, respectively.
44             Assume the nanosecond components are in range, or close to it.  */
45          static inline int
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          }
51
52          # if ! HAVE_DECL_NANOSLEEP
(gdb)
(gdb) quit
A debugging session is active.

            Inferior 1 [process 13300] will be killed.

Quit anyway? (y or n) y


I used gdb to find out that there might be a problem with 'diff' calculated in
timespec_cmp function, which appears to be in the lib/timespec.h:48. The number
is very large to the negative side, so it might cause a problem of overflowing,
or not being recognized by later functions.


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

$ cd ~/coreutils-with-bug/lib
$ find times*
timespec.h
$ cp timespec.h timespec2.h
            Make a copy of the older version in order to make the patch.
$ mv timespec2.h ~
$ emacs timespec.h
            Change the timespec_cmp function in timespec.h.
$ cd
$ diff -u timespec2.h ~/coreutils-with-bug/lib/timespec.h > lab4.diff


================================================================================
4.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?
$ cd
$ 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
-rw-r--r-- 1 huimin csugrad 0 2054-12-17 17:28:16.000000000 +0000 wwi-armistice
-rw-r--r-- 1 huimin csugrad 0 2018-05-03 04:31:56.081174000 +0000 now1
-rw-r--r-- 1 huimin csugrad 0 2018-05-03 04:31:48.210436000 +0000 now

Instead of creating a file in 1918-11-11, the touch -d command creates a file
in 2054-12-17. Unix Epoch defines a point in time as the number of seconds
elapsed since 1970-1-1 00:00. SEASnet NFS filesystem uses unsigned time stamps,
whereas local file system on Linux server uses signed time stamps. Therefore,
the time before 1970-1-1 00:00 will round to the future like an overflow
behavior in SEASnet NFS filesystem.
```

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

int frobcmp(const void* a, const void* b)
{
  if (a && b) {
```

```
  // compare two char arrays
  const char* charArrA = *(const char**)a;
  const char* charArrB = *(const char**)b;

  while ((*charArrA) != ' ' && (*charArrB) != ' ')
  {
    // decode each char
    char ca = (*charArrA)^42;
    char cb = (*charArrB)^42;

    if (ca > cb)
      return 1;
    else if (ca < cb)
      return -1;
    else
    {
      charArrA++;
      charArrB++;
    }
  }

  // if A ends earlier than B
  if ((*charArrA) == ' ' && (*charArrB) != ' ')
    return -1;
  // if B enes earlier than A
  else if ((*charArrB) == ' ' && (*charArrA) != ' ')
    return 1;
  // if they are the same
  else return 0;
  }
  else {
    fprintf(stderr, "frobcmp failed");
    exit(1);
  }
}

int main()
{
  int c = getchar();
  if (c == EOF || c == '\0')
    return 0;
  if (c == '\n')
    c = getchar();
  char** arr = malloc(sizeof(char*));
  char* phrase = malloc(sizeof(char));
  int psize = 1;
  int asize = 1;
  if (arr == NULL || phrase == NULL)
  {
    fprintf(stderr, "malloc failed");
    exit(1);
  }
  char prev = '0';
  while (!ferror(stdin))
  {
    if (c == ' ')
    {
      phrase[psize-1] = ' ';
      arr[asize-1] = phrase;
      c = getchar();
      if (c!=EOF) {


      char* nextphrase = (char*) malloc(sizeof(char));
      if (nextphrase != NULL) {
            phrase = nextphrase;
      }
      else {
            fprintf(stderr, "malloc failed");
            exit(1);
      }
      psize = 1;
            asize++;
            char** newarr = (char**)realloc(arr, asize*sizeof(char*));
            if (newarr != NULL)
              {
                arr = newarr;
              }
            else
              {
                fprintf(stderr, "realloc failed");
                exit(1);
              }
      }
      else {
            break;
      }
    }
    else if (c == EOF)
    {
      if (prev != ' ')
      {
            phrase[psize-1] = ' ';
      }
      arr[asize-1] = phrase;
      break;
      }
    else
    {
      char cchar = c;
      phrase[psize-1] = cchar;
      psize++;
      char* newphrase = realloc(phrase, (psize+1)*sizeof(char));
      if (newphrase != NULL)
      {
        phrase = newphrase;
      }
      else
      {
        fprintf(stderr, "realloc failed");
        exit(1);
      }
      c = getchar();
    }

    prev = c;
  }
  if (ferror(stdin)) {
    fprintf(stderr, "read error");
    exit(1);
  }

  //    printf("arr size: %d\n", asize);

  qsort(arr, asize, sizeof(char*), frobcmp);

  // Output
  size_t i, k;
  for (k = 0; k < asize; k++) {
    for (i = 0; ;i++) {
      char ch = arr[k][i];
      if (ch != ' ')
            putchar(ch);
      else {
            putchar(ch);
            break;
      }
    }
  }


  for (k = 0; k < asize; k++){
    if (arr[k]) {
      free(arr[k]);
      arr[k] = NULL;
    }
  }
  if (arr) {
    free(arr);
    arr = NULL;
  }
  return 0;
}
```

Assignment 5
lab.txt
1. 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.

Copying one file to another takes longer than copying a file to the terminal. One possible reason for this is that strace might not track
the processes of writing to a separate file after stdout, while it tracks writing to the terminal. tr2u is slower than tr2b. Approximately,
tr2u performs one system call for each byte. The privilege switch for each read() and write() causes tr2u to run slower and perfomr more
system calls than tr2b.

2. Use the time command to measure how much faster one program is, compared to the other, when copying the same amount of data.

tr2u takes longer time than tr2b because tr2u uses system calls, which cause a switch from user mode to kernel mode. System calls like read()
and write() cause a trap to interrupt the user process and the kernel takes control of the processor. Then the kernel executes the system
call and returns the processor to the user process. Therefore, system calls are expensive to use due to the privilege switch.

sfrobu.c

```
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <string.h>
#include <sys/stat.h>
#include <ctype.h>
int cmp(const void* a, const void* b, int upper)
{
  if (a && b) {
  // compare two char arrays
  const char* charArrA = *(const char**)a;
  const char* charArrB = *(const char**)b;

  while ((*charArrA) != ' ' && (*charArrB) != ' ')
  {
    // decode each char
    char ca = (*charArrA)^42;
    char cb = (*charArrB)^42;

    if (upper) {
      if (ca >= -1 && ca <= UCHAR_MAX)
            ca = toupper(ca);
      if (cb >= -1 && cb <= UCHAR_MAX)
        cb = toupper(cb);
    }

    if (ca > cb)
      return 1;
    else if (ca < cb)
      return -1;
    else
    {
      charArrA++;
      charArrB++;
    }
  }

  // if A ends earlier than B
  if ((*charArrA) == ' ' && (*charArrB) != ' ')
    return -1;
  // if B enes earlier than A
  else if ((*charArrB) == ' ' && (*charArrA) != ' ')
    return 1;
  // if they are the same
  else return 0;
  }
  else {
    write(2, "frobcmp failed\n", 15);
    exit(1);
  }
}


int frobcmpu(const void* a, const void* b)
{
  return cmp(a, b, 1);
}
int frobcmp(const void* a, const void* b)
{
  return cmp(a, b, 0);
}


int main(int argc, char **argv)
{
  struct stat buf;
  int f = fstat(0,&buf);
  off_t size = buf.st_size;
  if (size < 0) {
    write(2, "fstat failed\n",13);
    exit(1);
  }

  //  printf("size: %d\n", size);

  char* text = malloc(size*sizeof(char));
```

```
  int upper = 0;
  if (argc > 2) {
    write(2, "#arg error\n", 11);
    exit(1);
  }
  else if (argc == 2) {
    if (strcmp(argv[1], "-f") == 0) {
      upper = 1;
    }
    else {
      write(2, "wrong arg\n", 10);
      exit(1);
    }
  }

  //printf("file size: %d\n", size);

  int it = 0;
  while (1) {
    int c;
    int bytes = read(0, &c, 1);
    if (bytes == 0) {
      break;
    }
    if (!S_ISREG(0) && it >= size) {
      size++;
      char* newarr = realloc(text, size*sizeof(char));
      if (!newarr) {
        write(2, "realloc failed\n", 15);
        exit(1);
      }
      text = newarr;
    }

    text[it] = c;
    it++;
  }

  //printf("%s", text);

  if (text[size-1] != ' ') {
    size++;
    char* newarr = realloc(text, size*sizeof(char));
    if (!newarr) {
      write(2, "realloc failed\n", 15);
      exit(1);
    }
    text = newarr;
    text[size-1] = ' ';
  }
  int nPhrases = 0;
  for (size_t i = 0; i < size; i++) {
    if (text[i] == ' ')
      nPhrases++;
  }

  //  printf("\n%d phrases\n", nPhrases);

  char** arr = malloc(nPhrases*sizeof(char*));
  if (!arr) {
    write(2, "malloc failed\n", 14);
    exit(1);
  }


  arr[0] = &(text[0]);
  size_t asize = 1;
  for (size_t i = 0; i < size-1; i++) {
    if (text[i] == ' ')
      arr[asize++] = &(text[i+1]);
  }

  if (!upper)
    qsort(arr, asize, sizeof(char*), frobcmp);
  else
    qsort(arr, asize, sizeof(char*), frobcmpu);


  for (size_t i = 0; i < asize; i++) {
    for (size_t j = 0; ; j++) {
      char ch = arr[i][j];
      if (ch == ' ') {
            write(1, &ch, 1);
            break;
      }
      else
            write(1, &ch, 1);
    }
  }

  if (text) {
    free(text);
    text = NULL;
  }

  if (arr) {
    free(arr);
    arr = NULL;
  }
}
```

tr2b.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int length(char *a) {
  int length = 0;
  for (size_t j = 0; a[j] != '\0'; j++)
    length++;
  return length;
}

int main(int argc, char** argv) {
  if (argc != 3) {
    fprintf(stderr, "wrong number of arguments\n");
    exit(1);
  }

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

  if (length(from) != length(to)) {
    fprintf(stderr, "length error\n");
    exit(1);
  }

  int len = length(from);
  for (size_t i = 0; i < len; i++) {
    for (size_t j = i+1; j < len; j++) {
      if (from[i] == from[j]) {
            fprintf(stderr, "duplicate bytes\n");
            exit(1);
      }
    }
  }


  while (!ferror(stdin)) {
    int c = getchar();
    if (c == EOF)
      break;
    for (size_t i = 0; i < len; i++) {
      if (c == from[i]) {
            c = to[i];
            break;
      }
    }
    putchar(c);
  }
  return 0;
}
```

tr2u.c

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

int length(char *a) {
  int length = 0;
  for (size_t j = 0; a[j] != '\0'; j++)
    length++;
  return length;
}

int main(int argc, char** argv) {
  if (argc != 3) {
    write(2, "wrong number of arguments\n", 26);
    exit(1);
  }

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

  if (length(from) != length(to)) {
    write(2, "length error\n", 13);
    exit(1);
  }

  int len = length(from);
  for (size_t i = 0; i < len; i++) {
    for (size_t j = i+1; j < len; j++) {
      if (from[i] == from[j]) {
            write(2, "duplicate bytes\n", 16);
            exit(1);
      }
    }
  }


  while (1) {
    int c = 0;
    int bytes = read(0, &c, 1);
    if (bytes == 0) {
      break;
    }
    for (size_t i = 0; i < len; i++) {
      if (c == from[i])
            c = to[i];
    }
    write(1, &c, 1);
  }
  return 0;
}
```

sfrobs

```
#!/bin/sh
# sfrob

export LC_ALL='C'

arg=${1:-default}

encrypt=$(awk 'BEGIN { for (i = 0; i <= 255; i++) printf "\\%o", i }')

decrypt=$(awk 'BEGIN { for (i = 0; i <= 255; i++) printf "\\%o", xor(i, 42) }')

if [ "$arg" = "default" ]
then
    cat | tr "$encrypt" "$decrypt" | sort | tr "$decrypt" "$encrypt"
elif [ "$arg" = "-f" ]
then
    cat | tr "$encrypt" "$decrypt" | sort --ignore-case | tr "$decrypt" "$encrypt"
fi
```

Assignment 6

log.txt

```
1. Run the command sort --version to make sure you're using a new-enough version.
$ sort --version
sort (GNU coreutils) 8.29
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Mike Haertel and Paul Eggert.


2. Generate a file containing 10,000,000 random single-precision floating point numbers,
in textual form, one per line with no white space.
            $ od -An -f -N 40000000 < /dev/urandom | tr -s ' ' '\n' > random.txt
                        Generate a file with 10000000 floating point numbers. Since each
float is 4 bytes, the file needs to be 40000000 bytes. Transform all spaces to new
lines.
            $ cat random.txt | wc -l
            10000001
                        Check the number of lines. The output should be 10000000.
            $ cat random.txt | sed '/^[[:space:]]*$/d' > myFile.txt
                        Remove the additional new line.

3. time -p to time the command sort -g on that data, with the output sent to /dev/null.

4. Invoke sort with the --parallel option as well as the -g option, and run your
benchmark with 1, 2, 4, and 8 threads, in each case recording the real, user, and system
time.

            $ echo $PATH
            /usr/local/cs/bin/:/usr/lib64/qt-
3.3/bin:/u/cs/ugrad/huimin/perl5/bin:/usr/lib64/ccache:/usr/local/bin:/usr/bin:/usr/X11R
6/bin:/usr/local/cs/bin:/u/cs/ugrad/huimin/bin
```

```
$ time -p sort -g myFile.txt > /dev/null
real 21.91
user 114.87
sys 0.49
$ time -p sort -g --parallel=1 myFile.txt > /dev/null
real 109.57
user 109.27
sys 0.29
$ time -p sort -g --parallel=2 myFile.txt > /dev/null
real 60.27
user 114.60
sys 0.34
$ time -p sort -g --parallel=4 myFile.txt > /dev/null
real 33.71
user 113.41
sys 0.36
$ time -p sort -g --parallel=8 myFile.txt > /dev/null
real 21.89
user 114.54
sys 0.45
```

The real time gets shorter as the number of threads increases due to parallelism.
I don't see an obvious changing pattern with respect to system time, and user time almost stays constant.
When we can divide tasks into various independent steps with shared memory by running multiple cores together, we can get a better performance, as long as the overhead is still small.

sfrobu.c

```
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#include <string.h>
#include <sys/stat.h>
#include <ctype.h>
int cmp(const void* a, const void* b, int upper)
{
  if (a && b) {
  // compare two char arrays
  const char* charArrA = *(const char**)a;
  const char* charArrB = *(const char**)b;

  while ((*charArrA) != ' ' && (*charArrB) != ' ')
  {
    // decode each char
    char ca = (*charArrA)^42;
    char cb = (*charArrB)^42;

    if (upper) {
      if (ca >= -1 && ca <= UCHAR_MAX)
            ca = toupper(ca);
      if (cb >= -1 && cb <= UCHAR_MAX)
        cb = toupper(cb);
    }

    if (ca > cb)
      return 1;
    else if (ca < cb)
      return -1;
    else
    {
      charArrA++;
      charArrB++;
    }
  }

  // if A ends earlier than B
  if ((*charArrA) == ' ' && (*charArrB) != ' ')
    return -1;
  // if B enes earlier than A
  else if ((*charArrB) == ' ' && (*charArrA) != ' ')
    return 1;
  // if they are the same
  else return 0;
  }
  else {
    write(2, "frobcmp failed\n", 15);
    exit(1);
  }
}


int frobcmpu(const void* a, const void* b)
{
  return cmp(a, b, 1);
}
int frobcmp(const void* a, const void* b)
{
  return cmp(a, b, 0);
}


int main(int argc, char **argv)
{
  struct stat buf;
  int f = fstat(0,&buf);
  int size = buf.st_size;
  if (size < 0) {
    write(2, "fstat failed\n",13);
    exit(1);
  }

  //  printf("size: %d\n", size);

  char* text = malloc(size*sizeof(char));

  int upper = 0;
  if (argc > 2) {
    write(2, "#arg error\n", 11);
    exit(1);
  }
  else if (argc == 2) {
    if (strcmp(argv[1], "-f") == 0) {
      upper = 1;
    }
    else {
      write(2, "wrong arg\n", 10);
      exit(1);
    }
  }

  //printf("file size: %d\n", size);

  int it = 0;
  int c = 0;
  int bytes = read(0, &c, 1);
  if (bytes == 0)
    return 0;
  while (1) {
    if (bytes == 0) {
      break;
    }
    if (!S_ISREG(0) && it >= size) {
      size++;
      char* newarr = realloc(text, size*sizeof(char));
      if (!newarr) {
        write(2, "realloc failed\n", 15);
        exit(1);
      }
      text = newarr;
    }
    text[it] = c;
    it++;
    bytes = read(0, &c, 1);
  }

  //printf("%s", text);

  if (text[size-1] != ' ') {
    size++;
    char* newarr = realloc(text, size*sizeof(char));
    if (!newarr) {
      write(2, "realloc failed\n", 15);
      exit(1);
    }
    text = newarr;
    text[size-1] = ' ';
  }
  int nPhrases = 0;
  for (size_t i = 0; i < size; i++) {
    if (text[i] == ' ')
      nPhrases++;
  }

  //  printf("\n%d phrases\n", nPhrases);

  char** arr = malloc(nPhrases*sizeof(char*));
  if (!arr) {
    write(2, "malloc failed\n", 14);
    exit(1);
  }


  arr[0] = &(text[0]);
  size_t asize = 1;
  for (size_t i = 0; i < size-1; i++) {
    if (text[i] == ' ')
      arr[asize++] = &(text[i+1]);
  }

  if (!upper)
    qsort(arr, asize, sizeof(char*), frobcmp);
  else
    qsort(arr, asize, sizeof(char*), frobcmpu);


  for (size_t i = 0; i < asize; i++) {
    for (size_t j = 0; ; j++) {
      char ch = arr[i][j];
      if (ch == ' ') {
            write(1, &ch, 1);
            break;
      }
      else
            write(1, &ch, 1);
    }
  }

  if (text) {
    free(text);
    text = NULL;
  }

  if (arr) {
    free(arr);
    arr = NULL;
  }
}
```

```
Assignment 6
//
//  main.c
//  srt
//
//  Created by vector on 11/2/10.
//  Copyright (c) 2010 Brian F. Allen.

// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

#include "raymath.h"
#include "shaders.h"

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <pthread.h>
...
scene_t scene;
int nthreads = 0;
float color[width][height][3];

void* ThreadFunction(void *arg) {
  int thread = *(int*)arg;
  Vec3 camera_pos;
    set( camera_pos, 0., 0., -4. );
    Vec3 camera_dir;
    set( camera_dir, 0., 0., 1. );
    const double camera_fov = 75.0 * (PI/180.0);
    Vec3 bg_color;
    set( bg_color, 0.8, 0.8, 1 );

    const double pixel_dx = tan( 0.5*camera_fov ) / ((double)width*0.5);
    const double pixel_dy = tan( 0.5*camera_fov ) / ((double)height*0.5);
    const double subsample_dx
        = halfSamples  ? pixel_dx / ((double)halfSamples*2.0)
                       : pixel_dx;
    const double subsample_dy
        = halfSamples ? pixel_dy / ((double)halfSamples*2.0)
                      : pixel_dy;
```

```
    /* for every pixel */
    for( int px=thread; px<width; px+=nthreads )
    {
        const double x = pixel_dx * ((double)( px-(width/2) ));
        for( int py=0; py<height; ++py )
        {
            const double y = pixel_dy * ((double)( py-(height/2) ));
            Vec3 pixel_color;
            set( pixel_color, 0, 0, 0 );

            for( int xs=-halfSamples; xs<=halfSamples; ++xs )
            {
                for( int ys=-halfSamples; ys<=halfSamples; ++ys )
                {
                    double subx = x + ((double)xs)*subsample_dx;
                    double suby = y + ((double)ys)*subsample_dy;

                    /* construct the ray coming out of the camera, through
                     * the screen at (subx,suby)
                     */
                    ray_t pixel_ray;
                    copy( pixel_ray.org, camera_pos );
                    Vec3 pixel_target;
                    set( pixel_target, subx, suby, z );
                    sub( pixel_ray.dir, pixel_target, camera_pos );
                    norm( pixel_ray.dir, pixel_ray.dir );

                    Vec3 sample_color;
                    copy( sample_color, bg_color );
                    /* trace the ray from the camera that
                     * passes through this pixel */
                    trace( &scene, sample_color, &pixel_ray, 0 );
                    /* sum color for subpixel AA */
                    add( pixel_color, pixel_color, sample_color );
                }
            }

            /* at this point, have accumulated (2*halfSamples)^2 samples,
             * so need to average out the final pixel color
             */
            if( halfSamples )
            {
                mul( pixel_color, pixel_color,
                     (1.0/( 4.0 * halfSamples * halfSamples ) ) );
            }

            /* done, final floating point color values are in pixel_color */
            float scaled_color[3];
            scaled_color[0] = gamma( pixel_color[0] ) * max_color;
            scaled_color[1] = gamma( pixel_color[1] ) * max_color;
            scaled_color[2] = gamma( pixel_color[2] ) * max_color;

            /* enforce caps, replace with real gamma */
            for( int i=0; i<3; i++)
                scaled_color[i] = max( min(scaled_color[i], 255), 0);

            /* write this pixel out to disk. ppm is forgiving about whitespace,
             * but has a maximum of 70 chars/line, so use one line per pixel
             */
                color[px][py][0] = scaled_color[0];
                color[px][py][1] = scaled_color[1];
                color[px][py][2] = scaled_color[2];
            }
    }
    return NULL;
}

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

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

    scene = create_sphereflake_scene( sphereflake_recursion );

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

    pthread_t threadID[nthreads];
    int threadArgs[nthreads];

    /* Create threads */
    for (long t = 0; t < nthreads; t++) {
      threadArgs[t] = t;
      if (pthread_create(&threadID[t], NULL, ThreadFunction, &threadArgs[t])) {
            fprintf(stderr, "Creation error\n");
            exit(1);
      }
    }

    /* Join threads */
    for (long t = 0; t < nthreads; t++) {
      void* retVal;
      if (pthread_join(threadID[t], &retVal)) {
            fprintf(stderr, "Joining error\n");
            exit(1);
      }
    }

    /* Print */
    for (int i = 0; i < width; i++) {
      for (int j = 0; j < height; j++) {
            printf( "%.0f %.0f %.0f\n",
                        color[i][j][0], color[i][j][1], color[i][j][2] );
      }
      printf( "\n" );
    }

    free_scene( &scene );

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

    return 0;
}
```

Assignment 7
hw.txt

1.

(1) Suppose the other teams really had been observing all the bytes going across the network. Is your resulting network still secure?
I believe the network is still secure, because the file is encrypted by the sender's public key and has to be decrpted by the sender's private key, or vice versa. If the observer only knows the bytes go across the network, he will not be able to decrypt the encrypted message without the sender's key.

(2) you assumed the other teams had also tapped your keyboards and had observed all of your team's keystrokes
I think the resulting network is still secure since we disable password authentication. The verfication is done by assymetric encryption, in which a message is encrypted by my public key and will only be decrypted by my private key. The message is encrypted by symmetric encryption, or a session key. Without password authentication, I do not need to type my passphrase to get access to the message. Therefore, the observer would not know my passphrase. The session key is not typed in by my team but generated by the processor and will change frequently, so the observer would not be able to find out the session key to decrypt the message.

(3) you are booting off USB and you assume the other teams temporarily had physical control of the USB
If the other team had physical control of the beaglebone, I don't think the network will still be secure. They could use the board to log in as a root user and get access to the remote host server without entering the password since we disabled password authentication.

2. Explain why the gpg --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?
We can be sure
that the digital signature was sent by the sender because only the sender's public key can decrypt the digital signature. gpg --verify only verifies if I personally created the signature of the file, but not created the file itself. If someone else created eeprom, but I signed it with my key, the receiver would not know I am not the creator. One way to fix the problem is to attach the creator's public key with the file, so the receiver can verify signature using only the creator's public key.

Assignment 8

```
#include "randcpuid.h"
#include <stdio.h>
#include <dlfcn.h>
#include <stdbool.h>
#include <errno.h>
#include <stdlib.h>
static bool
writebytes (unsigned long long x, int nbytes)
{
  int ndigits = nbytes * 2;
  do
    {
      if (putchar ("0123456789abcdef"[x & 0xf]) < 0)
  return false;
      x >>= 4;
      ndigits--;
    }
  while (0 < ndigits);

  return 0 <= putchar ('\n');
}

/* Main program, which outputs N bytes of random data.  */
int
main (int argc, char **argv)
{
  /* Check arguments.  */
  bool valid = false;
  long long nbytes;
  if (argc == 2)
    {
      char *endptr;
      errno = 0;
      nbytes = strtoll (argv[1], &endptr, 10);
      if (errno)
  perror (argv[1]);
      else
  valid = !*endptr && 0 <= nbytes;
    }
  if (!valid)
    {
      fprintf (stderr, "%s: usage: %s NBYTES\n", argv[0], argv[0]);
      return 1;
    }

  /* If there's no work to do, don't worry about which library to use.  */
  if (nbytes == 0)
    return 0;

  /* Dynamic loading */
  void* dl_handle;
  unsigned long long (*rand64) (void);
  char* error;
  if (rdrand_supported()) {
    dl_handle = dlopen("randlibhw.so", RTLD_NOW);
    if (!dl_handle){
      printf("dlopen() error - %s\n", dlerror());
      return 1;
    }

    rand64 = dlsym(dl_handle, "hardware_rand64");
    error = dlerror();
    if (error != NULL) {
      printf("dlsym rand64 error - %s\n", error);
      return 1;
    }
  }

  else {
    dl_handle = dlopen("randlibsw.so", RTLD_NOW);
    if (!dl_handle) {
       printf("dlopen() error - %s\n", dlerror());
       return 1;
    }
    rand64 = dlsym(dl_handle, "software_rand64");
    error = dlerror();
    if (error != NULL) {
      printf("dlsym rand64 error - %s\n", error);
      return 1;
    }
  }


  int wordsize = sizeof rand64 ();
  int output_errno = 0;

  do
    {
      unsigned long long x = rand64 ();
      int outbytes = nbytes < wordsize ? nbytes : wordsize;
      if (!writebytes (x, outbytes))
```

```
  {
    output_errno = errno;
    break;
  }
      nbytes -= outbytes;
    }
  while (0 < nbytes);

  if (fclose (stdout) != 0)
    output_errno = errno;

  if (output_errno)
    {
      errno = output_errno;
      perror ("output");
      return 1;
    }

  dlclose(dl_handle);
  return 0;
}
```

lab.txt
ldd errors:
The errors might happen due to the following reasons:
(1) The program is a shell script and does not depend on any shared libraries.
(2) ldd does not work on a.out shared libraries.
(3) ldd does not work with some extremely old a.out programs which were built before ldd support was added to the compiler releases.

Assignment 9
lab9.txt
1. Get a copy of the Diffutils repository from the file ~eggert/src/gnu/diffutils:
$ git clone ~eggert/src/gnu/diffutils

2. Get git log and tags:
$ cd diffutils
$ git log > ~/git-log.txt
$ git tag > ~/git-tags.txt

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.
$ emacs ~/git-log.txt
C-s maint: quote 'like this'
<copy the ID> 62ca21c8c1a5aa3488589dcb191a4ef04ae9ed4f
$ git format-patch -1 62ca21c8c1a5aa3488589dcb191a4ef04ae9ed4f --stdout > quote-patch.txt

3. Check out version 3.0 of Diffutils from your repository.
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.
Use the git status command to get an overview of what happened.
$ git checkout v3.0
HEAD is now at 022cd5c... version 3.0
$ patch -p1 quote-patch.txt
skipped all patches with problems three times

```
$ 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
  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")
```

...

7. Remove all untracked files
$ git clean -f

...

$ git diff > quote-3.0-patch.txt
The file contains 327 lines.

9. Build the resulting modified version of Diffutils, using the commands described in the file README-hacking
./bootstrap
./configure
make
make check

10. Verify that Diffutils does the right thing with the “diff . -” scenario, as well as with “diff --help”.
$ diff . -
./src/diff: cannot compare '-' to a directory
$ diff --help
  All the apostrophes are correct.

11. Do a sanity test using the modified version of Diffutils that you just built
$ mkdir diffutils-3.0
$ mkdir diffutils-3.0-patch
$ mkdir diffutils-old
$ git clone ~eggert/src/gnu/diffutils diffutils-old
$ cd diffutils-old
$ git checkout v3.0
$ cd
$ cp diffutils-old/src/*.c diffutils-3.0
$ cp diffutils/src/*.c diffutils-3.0-patch
$ diffutils/src/diff -pru diffutils-3.0 diffutils-3.0-patch >quote-3.0-test.txt

12. 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?
$ diff -u quote-3.0-test.txt diffutils/quote-3.0-patch.txt
  The differences are mainly file paths, the positions of diff3.c are different in two patches. The actual contents are not different, therefore the differences are innocuous.

hw9.txt
1. Check out version 3.0 of Diffutils from your repository, into a new branch named “quote”.
$ git clone ~eggert/src/gnu/diffutils diffutilshw
$ cd diffutilshw
$ git checkout v3.0 -b 'quote'
Switched to a new branch 'quote'

2. Install your change into this new branch, by running the patch command with your patch quote-3.0-patch.txt as input.
$ patch -p1 <~/diffutils/quote-3.0-patch.txt

3. Learn how to use the Emacs function add-change-log-entry-other-window (C-x 4 a). Use this Emacs function to compose an appropriate ChangeLog entry for your patch, by adapting the change log from the original patch.
I open the six files and write the change log for each one using (C-x 4 a), and move ChangeLog from diffutils/src to diffutils
4. Commit your changes to the new branch, using the ChangeLog entry as the commit message.
$ git add .
$ git commit -F ChangeLog

5. 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.
$ git format-patch -1 --stdout > formatted-patch.txt

6. 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”.
$ git checkout v3.0 -b 'partner'
$ git am < ~/partner-patch.txt
$ git log
  My partner's log entry shows up in the log.
$ ./bootstrap
$ ./configure
$ make
$ make check
$ src/diff - .
src/diff: cannot compare '-' to a directory