comm - compare two sorted files line by line

Synopsis

comm [OPTION]... FILE1 FILE2

Description

Compare sorted files FILE1 and FILE2 line by line.

With no options, produce three-column output. Column one contains lines unique to FILE1, column two contains lines unique to FILE2, and column three contains lines common to both files.

-1
suppress column 1 (lines unique to FILE1)
-2
suppress column 2 (lines unique to FILE2)
-3
suppress column 3 (lines that appear in both files)
--check-order
check that the input is correctly sorted, even if all input lines are pairable
--nocheck-order
do not check that the input is correctly sorted
--output-delimiter=STR
separate columns with STR

//suppress first column using -1//

```
$comm -1 file1.txt file2.txt
        Apaar
        Hemant
 Lucky
 Pranjal Thakral
```

//suppress second column using -2//

```
$comm -2 file1.txt file2.txt
       Apaar
Ayush Rajput
Deepak
       Hemant
```

//suppress third column using -3//

```
$comm -3 file1.txt file2.txt
Ayush Rajput
Deepak
       Lucky
       Pranjal Thakral
```

sed - stream editor for filtering and transforming text

Synopsis

sed [OPTION]... {script-only-if-no-other-script} [input-file]...

Description

Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as ed), sed works by making only one pass over the input(s), and is consequently more efficient. But it is sed's ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

-n, --quiet, --silent
suppress automatic printing of pattern space
-e script, --expression=script
add the script to the commands to be executed
-f script-file, --file=script-file
add the contents of script-file to the commands to be executed
--follow-symlinks
follow symlinks when processing in place; hard links will still be broken.
-i[SUFFIX], --in-place[=SUFFIX]
edit files in place (makes backup if extension supplied). The default operation mode is to break symbolic and hard links. This can be changed with --follow-symlinks and --copy.
-c, --copy
use copy instead of rename when shuffling files in -i mode. While this will avoid breaking links (symbolic or hard), the resulting editing operation is not atomic. This is rarely the desired mode; --follow-symlinks is usually enough, and it is both faster and more secure.
-l N, --line-length=N
specify the desired line-wrap length for the 'l' command
--posix
disable all GNU extensions.
-r, --regexp-extended
use extended regular expressions in the script.
-s, --separate
consider files as separate rather than as a single continuous long stream.
-u, --unbuffered
load minimal amounts of data from the input files and flush the output buffers more often
--help

Notation	Effect
`8d`	Delete 8th line of input.
`/^$/d`	Delete all blank lines.
`1,/^$/d`	Delete from beginning of input up to, and including first blank line.
`/Jones/p`	Print only lines containing "Jones" (with -n option).
`s/Windows/Linux/`	Substitute "Linux" for first instance of "Windows" found in each input line.
`s/BSOD/stability/g`	Substitute "stability" for every instance of "BSOD" found in each input line.
`s/ *$//`	Delete all spaces at the end of every line.
`s/00*/0/g`	Compress all consecutive sequences of zeroes into a single zero.
`echo "Working on it." | sed -e '1i How far are you along?'`	Prints "How far are you along?" as first line, "Working on it" as second.
`5i 'Linux is great.' file.txt`	Inserts 'Linux is great.' at line 5 of the file file.txt.
`/GUI/d`	Delete all lines containing "GUI".
`s/GUI//g`	Delete all instances of "GUI", leaving the remainder of each line intact.

tr - translate or delete characters

Synopsis

tr [OPTION]... SET1 [SET2]

Description

Translate, squeeze, and/or delete characters from standard input, writing to standard output.

-c, -C, --complement
use the complement of SET1
-d, --delete
delete characters in SET1, do not translate
-s, --squeeze-repeats
replace each input sequence of a repeated character that is listed in SET1 with a single occurrence of that character
-t, --truncate-set1
first truncate SET1 to length of SET2
--help
display this help and exit
--version
output version information and exit
SETs are specified as strings of characters. Most represent themselves. Interpreted sequences are:

\NNN
character with octal value NNN (1 to 3 octal digits)
\\
backslash
\a
audible BEL
\b
backspace
\f
form feed
\n
new line

\r
return
\t
horizontal tab
\v
vertical tab
CHAR1-CHAR2
all characters from CHAR1 to CHAR2 in ascending order
[CHAR*]
in SET2, copies of CHAR until length of SET1
[CHAR*REPEAT]
REPEAT copies of CHAR, REPEAT octal if starting with 0
[:alnum:]
all letters and digits
[:alpha:]
all letters
[:blank:]
all horizontal whitespace
[:cntrl:]
all control characters
[:digit:]
all digits
[:graph:]
all printable characters, not including space
[:lower:]
all lower case letters
[:print:]
all printable characters, including space
[:punct:]
all punctuation characters
[:space:]
all horizontal or vertical whitespace
[:upper:]
all upper case letters
[:xdigit:]
all hexadecimal digits
[=CHAR=]
all characters which are equivalent to CHAR

grep, egrep, fgrep - print lines that match patterns

SYNOPSIS top

grep [OPTION...] PATTERNS [FILE...]
grep [OPTION...] -e PATTERNS ... [FILE...]
grep [OPTION...] -f PATTERN_FILE ... [FILE...]

DESCRIPTION top

grep searches for PATTERNS in each FILE. PATTERNS is one or patterns
separated by newline characters, and grep prints each line that matches a pattern.

A FILE of “-” stands for standard input. If no FILE is given, recursive searches examine the working directory, and nonrecursive searches read standard input.

In addition, the variant programs egrep and fgrep are the same as grep -E and grep -F, respectively. These variants are deprecated, but are provided for backward compatibility.

OPTIONS top

Generic Program Information

--help Output a usage message and exit.

-V, --version
Output the version number of grep and exit.

Matcher Selection

-E, --extended-regexp
Interpret PATTERNS as extended regular expressions (EREs, see below).

-F, --fixed-strings
Interpret PATTERNS as fixed strings, not regular expressions.

-G, --basic-regexp
Interpret PATTERNS as basic regular expressions (BREs, see below). This is the default.

sort - sort lines of text files

SYNOPSIS top

sort [OPTION]... [FILE]...
sort [OPTION]... --files0-from=F

DESCRIPTION top

Write sorted concatenation of all FILE(s) to standard output.

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

Mandatory arguments to long options are mandatory for short options too. Ordering options:

-b, --ignore-leading-blanks
ignore leading blanks

-d, --dictionary-order
consider only blanks and alphanumeric characters

-f, --ignore-case
fold lower case to upper case characters

-g, --general-numeric-sort
compare according to general numerical value

-i, --ignore-nonprinting
consider only printable characters

-M, --month-sort
compare (unknown) < 'JAN' < ... < 'DEC'

-h, --human-numeric-sort
compare human readable numbers (e.g., 2K 1G)

-n, --numeric-sort
compare according to string numerical value

-R, --random-sort
shuffle, but group identical keys. See shuf(1)

--random-source=FILE
get random bytes from FILE

-r, --reverse
reverse the result of comparisons

--sort=WORD
sort according to WORD: general-numeric -g, human-numeric -h, month -M, numeric -n, random -R, version -V

-V, --version-sort
natural sort of (version) numbers within text

Bash

```
#!/bin/bash
# Counting the number of lines in a list of files
# function version

# function storing list of all files in variable files
get_files () {
  files="`ls *.[ch]`"
}

# function counting the number of lines in a file
count_lines () {
```

```
  f=$1  # 1st argument is filename
  l=`wc -l $f | sed 's/^\([0-9]*\).*$/\1/'` # number of lines
}

# the script should be called without arguments
if [ $# -ge 1 ]
then
  echo "Usage: $0 "
  exit 1
fi

# split by newline
IFS=$'\012'

echo "$0 counts the lines of code"
# don't forget to initialise!
l=0
n=0
s=0
# call a function to get a list of files
get_files
# iterate over this list
for f in $files
do
        # call a function to count the lines
        count_lines $f
        echo "$f: $l"loc
                # store filename in an array
                file[$n]=$f
                # store number of lines in an array
                lines[$n]=$l
                # increase counter
        n=$[ $n + 1 ]
                # increase sum of all lines
        s=$[ $s + $l ]
done

echo "$n files in total, with $s lines in total"
i=5
echo "The $i-th file was ${file[$i]} with ${lines[$i]} lines"'
```

putchar() Declaration: int putchar(int char)
putchar() function is used to write a character on standard output/screen. In a C program, we can use putchar function as below.
putchar(char);

where, char is a character variable/value.

getchar() Declaration: int getchar(void)
getchar() function is used to get/read a character from keyboard input. In a C program, we can use getchar function as below.
getchar(char);

where, char is a character variable/value.

malloc
void* malloc (size_t size);
Allocate memory block
Allocates a block of size bytes of memory, returning a pointer to the beginning of the block.

The content of the newly allocated block of memory is not initialized, remaining with indeterminate values.

If size is zero, the return value depends on the particular library implementation (it may or may not be a null pointer), but the returned pointer shall not be dereferenced.

Parameters
size
Size of the memory block, in bytes.
size_t is an unsigned integral type.

Return Value
On success, a pointer to the memory block allocated by the function.
The type of this pointer is always void*, which can be cast to the desired type of data pointer in order to be dereferenceable.
If the function failed to allocate the requested block of memory, a null pointer is returned.

realloc
void* realloc (void* ptr, size_t size);
Reallocate memory block
Changes the size of the memory block pointed to by ptr.

The function may move the memory block to a new location (whose address is returned by the function).

The content of the memory block is preserved up to the lesser of the new and old sizes, even if the block is moved to a new location. If the new size is larger, the value of the newly allocated portion is indeterminate.

In case that ptr is a null pointer, the function behaves like malloc, assigning a new block of size bytes and returning a pointer to its beginning.

Otherwise, if size is zero, the memory previously allocated at ptr is deallocated as if a call to free was made, and a null pointer is returned.

If the function fails to allocate the requested block of memory, a null pointer is returned, and the memory block pointed to by argument ptr is not deallocated (it is still valid, and with its contents unchanged).

Parameters
ptr
Pointer to a memory block previously allocated with malloc, calloc or realloc.
Alternatively, this can be a null pointer, in which case a new block is allocated (as if malloc was called).
size
New size for the memory block, in bytes.
size_t is an unsigned integral type.

Return Value
A pointer to the reallocated memory block, which may be either the same as ptr or a new location.
The type of this pointer is void*, which can be cast to the desired type of data pointer in order to be dereferenceable.

free
void free (void* ptr);
Deallocate memory block
A block of memory previously allocated by a call to malloc, calloc or realloc is deallocated, making it available again for further allocations.

If ptr does not point to a block of memory allocated with the above functions, it causes undefined behavior.

If ptr is a null pointer, the function does nothing.

Notice that this function does not change the value of ptr itself, hence it still points to the same (now invalid) location.

Parameters
ptr
Pointer to a memory block previously allocated with malloc, calloc or realloc.

Return Value
None

open - open a file
SYNOPSIS
[OH] [Option Start] #include <sys/stat.h> [Option End]

#include <fcntl.h>

int open(const char *path, int oflag, ...);
DESCRIPTION
The open() function shall establish the connection between a file and a file descriptor. It shall create an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The path argument points to a pathname naming the file.

The open() function shall return a file descriptor for the named file that is the lowest file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor shall not share it with any other process in the system. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared.

The file offset used to mark the current position within the file shall be set to the beginning of the file.

The file status flags and file access modes of the open file description shall be set according to the value of oflag.

Values for oflag are constructed by a bitwise-inclusive OR of flags from the following list, defined in <fcntl.h>. Applications shall specify exactly one of the first three values (file access modes) below in the value of oflag:

O_RDONLY
Open for reading only.
O_WRONLY
Open for writing only.
O_RDWR
Open for reading and writing. The result is undefined if this flag is applied to a FIFO.
Any combination of the following may be used:

O_APPEND
If set, the file offset shall be set to the end of the file prior to each write.

read - read from a file
SYNOPSIS
#include <unistd.h>

ssize_t read(int fildes, void *buf, size_t nbyte);
DESCRIPTION
The read() function shall attempt to read nbyte bytes from the file associated with the open file descriptor, fildes, into the buffer pointed to by buf. The behavior of multiple concurrent reads on the same pipe, FIFO, or terminal device is unspecified.

Before any action described below is taken, and if nbyte is zero, the read() function may detect and return errors as described below. In the absence of errors, or if error detection is not performed, the read() function shall return zero and have no other results.

On files that support seeking (for example, a regular file), the read() shall start at a position in the file given by the file offset associated with fildes. The file offset shall be incremented by the number of bytes actually read.

RETURN VALUE
Upon successful completion, read() [XSI] [Option Start] and pread() [Option End] shall return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions shall return -1 and set errno to indicate the error.

write - write on a file
SYNOPSIS
#include <unistd.h>

ssize_t write(int fildes, const void *buf, size_t nbyte);
DESCRIPTION
The write() function shall attempt to write nbyte bytes from the buffer pointed to by buf to the file associated with the open file descriptor, fildes.

Before any action described below is taken, and if nbyte is zero and the file is a regular file, the write() function may detect and return errors as described below. In the absence of errors, or if error detection is not performed, the write() function shall return zero and have no other results. If nbyte is zero and the file is not a regular file, the results are unspecified.

On a regular file or other file capable of seeking, the actual writing of data shall proceed from the position in the file indicated by the file offset associated with fildes. Before successful return from write(), the file offset shall be incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file shall be set to this file offset.

If the O_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file prior to each write and no intervening file modification operation shall occur between changing the file offset and the write operation.

If a write() requests that more bytes be written than there is room for (for example, [XSI] [Option Start] the process' file size limit or [Option End] the physical end of a medium), only as many bytes as there is room for shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would give a failure return (except as noted below).

RETURN VALUE
Upon successful completion, write() [XSI] [Option Start] shall return the number of bytes actually written to the file associated with fildes. This number shall never be greater than nbyte. Otherwise, -1 shall be returned and errno set to indicate the error.

```
edit : main.o kbd.o command.o display.o \
       insert.o search.o files.o utils.o
        cc -o edit main.o kbd.o command.o display.o \
                  insert.o search.o files.o utils.o

main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
```