cloc *Count Lines of Code*

---

cloc counts blank lines, comment lines, and physical lines of source code in many programming languages.

Latest release: v1.82 (May 3, 2019)

Hosted at http://cloc.sourceforge.net/ (http://cloc.sourceforge.net/) since August 2006, cloc began the transition to GitHub in September 2015.

(%7B%7B%7B1) # Quick Start ▲

Step 1: Download cloc (several methods, see below).

Step 2: Open a terminal (`cmd.exe` on Windows).

Step 3: Invoke cloc to count your source files, directories, archives, or git commits. The executable name differs depending on whether you use the development source version (`cloc`), source for a released version (`cloc-1.82.pl`) or a Windows executable (`cloc-1.82.exe`). On this page, `cloc` is the generic term used to refer to any of these.

a file

```
prompt> cloc hello.c
       1 text file.
       1 unique file.
       0 files ignored.

https://github.com/AlDanial/cloc v 1.65  T=0.04 s (28.3 files/s, 340.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                                1              0              7              5
-------------------------------------------------------------------------------
```

a directory

```
prompt> cloc gcc-5.2.0/gcc/c
      16 text files.
      15 unique files.
       3 files ignored.

https://github.com/AlDanial/cloc v 1.65  T=0.23 s (57.1 files/s, 188914.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                               10           4680           6621          30812
C/C++ Header                     3             99            286            496
-------------------------------------------------------------------------------
SUM:                            13           4779           6907          31308
-------------------------------------------------------------------------------
```

an archive

We'll pull cloc's source zip file from GitHub, then count the contents:

```
prompt> wget https://github.com/AlDanial/cloc/archive/master.zip

prompt> cloc master.zip
https://github.com/AlDanial/cloc v 1.65  T=0.07 s (26.8 files/s, 141370.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Perl                             2            725           1103           8713
-------------------------------------------------------------------------------
SUM:                             2            725           1103           8713
-------------------------------------------------------------------------------
```

a git repository, using a specific commit

This example uses code from PuDB (https://pypi.python.org/pypi/pudb), a fantastic Python debugger.

```
prompt> git clone http://git.tiker.net/trees/pudb.git

prompt> cd pudb

prompt> cloc 6be804e07a5db
      48 text files.
      48 unique files.
      15 files ignored.

github.com/AlDanial/cloc v 1.73  T=0.15 s (223.1 files/s, 46159.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                          28           1519            728           4659
YAML                             2              9              2             75
Bourne Shell                     3              6              0             17
make                             1              4              6             10
-------------------------------------------------------------------------------
SUM:                            34           1538            736           4761
-------------------------------------------------------------------------------
```

each subdirectory of a particular directory

Say you have a directory with three different git-managed projects, Project0, Project1, and Project2. You can use your shell's looping capability to count the code in each. This example uses bash:

```
prompt> for d in ./*/ ; do (cd "$d" && echo "$d" && cloc --vcs git); done
./Project0/
7 text files.
       7 unique files.
       1 file ignored.

github.com/AlDanial/cloc v 1.71  T=0.02 s (390.2 files/s, 25687.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
D                                4             61             32            251
Markdown                         1              9              0             38
make                             1              0              0              4
-------------------------------------------------------------------------------
SUM:                             6             70             32            293
-------------------------------------------------------------------------------
./Project1/
       7 text files.
       7 unique files.
       0 files ignored.

github.com/AlDanial/cloc v 1.71  T=0.02 s (293.0 files/s, 52107.1 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Go                               7            165            282            798
-------------------------------------------------------------------------------
SUM:                             7            165            282            798
-------------------------------------------------------------------------------
./Project2/
      49 text files.
      47 unique files.
      13 files ignored.

github.com/AlDanial/cloc v 1.71  T=0.10 s (399.5 files/s, 70409.4 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                          33           1226           1026           3017
C                                4            327            337            888
Markdown                         1             11              0             28
YAML                             1              0              2             12
-------------------------------------------------------------------------------
SUM:                            39           1564           1365           3945
-------------------------------------------------------------------------------
```

(1%7D%7D%7D) (%7B%7B%7B1) # Overview ▲

Translations: Arabic (http://www.garciniacambogiareviews.ca/translations/aldanial-cloc/), Armenian (http://students.studybay.com/?p=34), Belarussian (http://www.besteonderdelen.nl/blog/?p=5426), Bulgarian (http://www.ajoft.com/wpaper/aj-cloc.html), Hungarian (http://www.forallworld.com/cloc-grof-sornyi-kodot/), (%5BPolish%5D(http://www.trevister.com/blog/cloc.html),) (%5BRussian%5D(http://someblogscience.com/cloc.html),) Portuguese (https://www.homeyou.com/~edu/cloc), Serbo-Croatian (http://science.webhostinggeeks.com/cloc), Romanian (http://www.bildelestore.dk/blog/cloc-contele-de-linii-de-cod/), Slovakian (http://newknowledgez.com/cloc.html), Tamil (http://healthcareadministrationdegree.co/socialwork/aldanial-cloc/) (%5BUkrainian%5D(http://blog.kudoybook.com/cloc/))

cloc counts blank lines, comment lines, and physical lines of source code in many programming languages. Given two versions of a code base, cloc can compute differences in blank, comment, and source lines. It is written entirely in Perl with no dependencies outside the standard distribution of Perl v5.6 and higher (code from some external modules is embedded within cloc (https://github.com/AlDanial/cloc#regexp_common)) and so is quite portable. cloc is known to run on many flavors of Linux, FreeBSD, NetBSD, OpenBSD, Mac OS X, AIX, HP-UX, Solaris, IRIX, z/OS, and Windows. (To run the Perl source version of cloc on Windows one needs ActiveState Perl (http://www.activestate.com/activeperl) 5.6.1 or higher, Strawberry Perl (http://strawberryperl.com/), Cygwin (http://www.cygwin.com/), MobaXTerm (http://mobaxterm.mobatek.net/) with the Perl plug-in installed, or a mingw environment and terminal such as provided by Git for Windows (https://gitforwindows.org/). Alternatively one can use the Windows binary of cloc generated with PAR::Packer (http://search.cpan.org/~rschupp/PAR-Packer-1.019/lib/pp.pm) to run on Windows computers that have neither Perl nor Cygwin.)

cloc contains code from David Wheeler's SLOCCount (http://www.dwheeler.com/sloccount/), Damian Conway and Abigail's Perl module Regexp::Common (http://search.cpan.org/%7Eabigail/Regexp-Common-2.120/lib/Regexp/Common.pm), Sean M. Burke's Perl module Win32::Autoglob (http://search.cpan.org/%7Esburke/Win32-Autoglob-1.01/Autoglob.pm), and Tye McQueen's Perl module Algorithm::Diff (http://search.cpan.org/%7Etyemq/Algorithm-Diff-1.1902/lib/Algorithm/Diff.pm). Language scale factors were derived from Mayes Consulting, LLC web site http://softwareestimator.com/IndustryData2.htm (http://softwareestimator.com/IndustryData2.htm). (1%7D%7D%7D) (%7B%7B%7B1)

Install via package manager

Depending your operating system, one of these installation methods may work for you:

```
npm install -g cloc                  # https://www.npmjs.com/package/cloc
sudo apt install cloc                # Debian, Ubuntu
sudo yum install cloc                # Red Hat, Fedora
sudo dnf install cloc                # Fedora 22 or later
sudo pacman -S cloc                  # Arch
sudo emerge -av dev-util/cloc        # Gentoo https://packages.gentoo.org/packages/dev-util/cloc
sudo apk add cloc                    # Alpine Linux
sudo pkg install cloc                # FreeBSD
sudo port install cloc               # Mac OS X with MacPorts
brew install cloc                    # Mac OS X with Homebrew
choco install cloc                   # Windows with Chocolatey
scoop install cloc                   # Windows with Scoop
```

Note: I don't control any of these packages. If you encounter a bug in cloc using one of the above packages, try with cloc pulled from the latest stable release here on github (link follows below) before submitting a problem report. (1%7D%7D%7D) (%7B%7B%7B1) ## Stable release https://github.com/AlDanial/cloc/releases/latest (https://github.com/AlDanial/cloc/releases/latest)

Development version https://github.com/AlDanial/cloc/raw/master/cloc (https://github.com/AlDanial/cloc/raw/master/cloc) (1%7D%7D%7D) (%7B%7B%7B1) # License ▲

cloc is licensed under the GNU General Public License, v 2 (http://www.gnu.org/licenses/gpl-2.0.html), excluding portions which are copied from other sources. Code copied from the Regexp::Common, Win32::Autoglob, and Algorithm::Diff Perl modules is subject to the Artistic License (http://www.opensource.org/licenses/artistic-license-2.0.php). (1%7D%7D%7D) (%7B%7B%7B1) # Why Use cloc? ▲

cloc has many features that make it easy to use, thorough, extensible, and portable:

1. Exists as a single, self-contained file that requires minimal installation effort—just download the file and run it.
2. Can read language comment definitions from a file and thus potentially work with computer languages that do not yet exist.
3. Allows results from multiple runs to be summed together by language and by project.
4. Can produce results in a variety of formats: plain text, SQL, JSON, XML, YAML, comma separated values.
5. Can count code within compressed archives (tar balls, Zip files, Java .ear files).
6. Has numerous troubleshooting options.
7. Handles file and directory names with spaces and other unusual characters.
8. Has no dependencies outside the standard Perl distribution.
9. Runs on Linux, FreeBSD, NetBSD, OpenBSD, Mac OS X, AIX, HP-UX, Solaris, IRIX, and z/OS systems that have Perl 5.6 or higher. The source version runs on Windows with either ActiveState Perl, Strawberry Perl, Cygwin, or MobaXTerm+Perl plugin. Alternatively on Windows one can run the Windows binary which has no dependencies. (1%7D%7D%7D)

(%7B%7B%7B1) # Other Counters ▲

If cloc does not suit your needs here are other freely available counters to consider:

- loc (https://github.com/cgag/loc/)
- gocloc (https://github.com/hhatto/gocloc/)
- Ohcount (https://github.com/blackducksoftware/ohcount/)

- scc (https://github.com/boyter/scc/)
- sclc (https://code.google.com/archive/p/sclc/)
- SLOCCount (http://www.dwheeler.com/sloccount/)
- Sonar (http://www.sonarsource.org/)
- tokei (https://github.com/Aaronepower/tokei/)
- Unified Code Count (http://csse.usc.edu/ucc_new/wordpress/)

Other references:

- QSM's directory (http://www.qsm.com/CodeCounters.html) of code counting tools.
- The Wikipedia entry (http://en.wikipedia.org/wiki/Source_lines_of_code) for source code line counts.

Regexp::Common, Digest::MD5, Win32::Autoglob, Algorithm::Diff

Although `cloc` does not need Perl modules outside those found in the standard distribution, cloc does rely on a few external modules. Code from three of these external modules–Regexp::Common, Win32::Autoglob, and Algorithm::Diff–is embedded within cloc. A fourth module, Digest::MD5, is used only if it is available. If cloc finds Regexp::Common or Algorithm::Diff installed locally it will use those installation. If it doesn't, cloc will install the parts of Regexp::Common and/or Algorithm:Diff it needs to temporary directories that are created at the start of a cloc run then removed when the run is complete. The necessary code from Regexp::Common v2.120 and Algorithm::Diff v1.1902 are embedded within the cloc source code (see subroutines `Install_Regexp_Common()` and `Install_Algorithm_Diff()`).

Only three lines are needed from Win32::Autoglob and these are included directly in cloc.

Additionally, cloc will use Digest::MD5 to validate uniqueness among equally-sized input files if Digest::MD5 is installed locally.

A parallel processing option, `–processes=N`, was introduced with cloc version 1.76 to enable faster runs on multicored machines. However, to use it, one must have the module Parallel::ForkManager installed. This module does not work reliably on Windows so parallel processing will only work on Unix-like operating systems.

The Windows binary is built on a computer that has both Regexp::Common and Digest::MD5 installed locally. (1%7D%7D%7D) (%7B%7B%7B1) # Building a Windows Executable ▲

The Windows downloads `cloc-1.70.exe` and `cloc-1.72.exe` were built with PAR::Packer (http://search.cpan.org/~rschupp/PAR-Packer-1.019/lib/pp.pm) and Strawberry Perl 5.24.0.1 on an Amazon Web Services t2.micro instance running Microsoft Windows Server 2008 (32 bit for 1.70 and 1.72; 64 bit for 1.74).

Releases 1.74 through 1.82 were was built on a 32 bit Windows 7 virtual machine (IE11.Win7.For.Windows.VirtualBox.zip pulled from https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/ (https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/)) using Strawberry Perl 5.26.1.1.

The `cloc-1.66.exe` executable was built with PAR::Packer (http://search.cpan.org/~rschupp/PAR-Packer-1.019/lib/pp.pm) on a 32 bit Windows 7 VirtualBox image pulled from https://dev.windows.com/en-us/microsoft-edge/tools/vms/linux/ (https://dev.windows.com/en-us/microsoft-edge/tools/vms/linux/) and running on an Ubuntu 15.10 host. The virtual machine ran Strawberry Perl (http://strawberryperl.com/) version 5.22.1. Windows executables of cloc versions 1.60 and earlier were built with perl2exe (http://www.indigostar.com/perl2exe.htm) on a 32 bit Windows XP computer. A small modification was made to the cloc source code before passing it to perl2exe; lines 87 and 88 were uncommented:

Windows code

```
85  # Uncomment next two lines when building Windows executable with perl2exe
86  # or if running on a system that already has Regexp::Common.
87  #use Regexp::Common;
88  #$HAVE_Rexexp_Common = 1;
```

Is the Windows executable safe to run? Does it have malware?

Ideally, no one would need the Windows executable because they have a Perl interpreter installed on their machines and can run the cloc source file. On centrally-managed corporate Windows machines, however, this this may be difficult or impossible.

The Windows executable distributed with cloc is provided as a best-effort of a virus and malware-free `.exe` . You are encouraged to run your own virus scanners against the executable and also check sites such https://www.virustotal.com/ (https://www.virustotal.com/) . The entries for recent versions are:

cloc-1.82.exe: https://www.virustotal.com/#/file/2e5fb443fdefd776d7b6b136a25e5ee2048991e735042897dbd0bf92efb16563/detection (https://www.virustotal.com/#/file/2e5fb443fdefd776d7b6b136a25e5ee2048991e735042897dbd0bf92efb16563/detection)

cloc-1.80.exe: https://www.virustotal.com/#/file/9e547b01c946aa818ffad43b9ebaf05d3da08ed6ca876ef2b6847be3bf1cf8be/detection (https://www.virustotal.com/#/file/9e547b01c946aa818ffad43b9ebaf05d3da08ed6ca876ef2b6847be3bf1cf8be/detection)

cloc-1.78.exe: https://www.virustotal.com/#/file/256ade3df82fa92febf2553853ed1106d96c604794606e86efd00d55664dd44f/detection (https://www.virustotal.com/#/file/256ade3df82fa92febf2553853ed1106d96c604794606e86efd00d55664dd44f/detection)

cloc-1.76.exe: https://www.virustotal.com/#/url/c1b9b9fe909f91429f95d41e9a9928ab7c58b21351b3acd4249def2a61acd39d/detection (https://www.virustotal.com/#/url/c1b9b9fe909f91429f95d41e9a9928ab7c58b21351b3acd4249def2a61acd39d/detection)

cloc-1.74_x86.exe: https://www.virustotal.com/#/file/b73dece71f6d3199d90d55db53a588e1393c8dbf84231a7e1be2ce3c5a0ec75b/detection (https://www.virustotal.com/#/file/b73dece71f6d3199d90d55db53a588e1393c8dbf84231a7e1be2ce3c5a0ec75b/detection)

cloc 1.72 exe: https://www.virustotal.com/en/url/8fd2af5cd972f648d7a2d7917bc202492012484c3a6f0b48c8fd60a8d395c98c/analysis/ (https://www.virustotal.com/en/url/8fd2af5cd972f648d7a2d7917bc202492012484c3a6f0b48c8fd60a8d395c98c/analysis/)

cloc 1.70 exe: https://www.virustotal.com/en/url/63edef209099a93aa0be1a220dc7c4c7ed045064d801e6d5daa84ee624fc0b4a/analysis/ (https://www.virustotal.com/en/url/63edef209099a93aa0be1a220dc7c4c7ed045064d801e6d5daa84ee624fc0b4a/analysis/)

cloc 1.68 exe: https://www.virustotal.com/en/file/c484fc58615fc3b0d5569b9063ec1532980281c3155e4a19099b11ef1c24443b/analysis/ (https://www.virustotal.com/en/file/c484fc58615fc3b0d5569b9063ec1532980281c3155e4a19099b11ef1c24443b/analysis/)

cloc 1.66 exe:
https://www.virustotal.com/en/file/54d6662e59b04be793dd10fa5e5edf7747cf0c0cc32f71eb67a3cf8e7a171d81/analysis/1453601367/
(https://www.virustotal.com/en/file/54d6662e59b04be793dd10fa5e5edf7747cf0c0cc32f71eb67a3cf8e7a171d81/analysis/1453601367/)

Why is the Windows executable so large?

Windows executables of cloc versions 1.60 and earlier, created with perl2exe as noted above, are about 1.6 MB, while versions 1.62 and 1.54, created with `PAR::Packer`, are 11 MB. Version 1.66, built with a newer version of `PAR::Packer`, is about 5.5 MB. Why are the `PAR::Packer`, executables so much larger than those built with perl2exe? My theory is that perl2exe uses smarter tree pruning logic than `PAR::Packer`, but that's pure speculation.

Create your own executable

The most robust option for creating a Windows executable of cloc is to use ActiveState's Perl Development Kit (http://www.activestate.com/perl-dev-kit). It includes a utility, `perlapp`, which can build stand-alone Windows, Mac, and Linux binaries of Perl source code.

perl2exe (http://www.indigostar.com/perl2exe.php) will also do the trick. If you do have `perl2exe`, modify lines 84-87 in the cloc source code for a minor code modification that is necessary to make a cloc Windows executable.

Otherwise, to build a Windows executable with `pp` from `PAR::Packer`, first install a Windows-based Perl distribution (for example Strawberry Perl or ActivePerl) following their instructions. Next, open a command prompt, aka a DOS window and install the PAR::Packer module. Finally, invoke the newly installed `pp` command with the cloc source code to create an `.exe` file:

```
C:> cpan -i Digest::MD5
C:> cpan -i Regexp::Common
C:> cpan -i Algorithm::Diff
C:> cpan -i PAR::Packer
C:> pp -M Digest::MD5 -c -x -o cloc-1.82.exe cloc
```

A variation on the instructions above is if you installed the portable version of Strawberry Perl, you will need to run `portableshell.bat` first to properly set up your environment.

(1%7D%7D%7D) (%7B%7B%7B1) # Basic Use ▲

cloc is a command line program that takes file, directory, and/or archive names as inputs. Here's an example of running cloc against the Perl v5.22.0 source distribution:

```
prompt> cloc perl-5.22.0.tar.gz
    5605 text files.
    5386 unique files.
    2176 files ignored.

https://github.com/AlDanial/cloc v 1.65  T=25.49 s (134.7 files/s, 51980.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Perl                          2892         136396         184362         536445
C                              130          24676          33684         155648
C/C++ Header                   148           9766          16569         147858
Bourne Shell                   112           4044           6796          42668
Pascal                           8            458           1603           8592
XML                             33            142              0           2410
YAML                            49             20             15           2078
C++                             10            313            277           2033
make                             4            426            488           1986
Prolog                          12            438              2           1146
JSON                            14              1              0           1037
yacc                             1             85             76            998
Windows Message File             1            102             11            489
DOS Batch                       14             92             41            389
Windows Resource File            3             10              0             85
D                                1              5              7              8
Lisp                             2              0              3              4
-------------------------------------------------------------------------------
SUM:                          3434         176974         243934         903874
-------------------------------------------------------------------------------
```

To run cloc on Windows computers, one must first open up a command (aka DOS) window and invoke cloc.exe from the command line there. (1%7D%7D%7D) (%7B%7B%7B1) # Options ▲

```
prompt> cloc --help

Usage: cloc [options] <file(s)/dir(s)/git hash(es)> | <set 1> <set 2> | <report files>

 Count, or compute differences of, physical lines of source code in the
 given files (may be archives such as compressed tarballs or zip files,
 or git commit hashes or branch names) and/or recursively below the
 given directories.

 Input Options
   --extract-with=<cmd>      This option is only needed if cloc is unable
                             to figure out how to extract the contents of
                             the input file(s) by itself.
                             Use <cmd> to extract binary archive files (e.g.:
                             .tar.gz, .zip, .Z).  Use the literal '>FILE<' as
                             a stand-in for the actual file(s) to be
                             extracted.  For example, to count lines of code
                             in the input files
                                gcc-4.2.tar.gz  perl-5.8.8.tar.gz
                             on Unix use
                               --extract-with='gzip -dc >FILE< | tar xf -'
                             or, if you have GNU tar,
                               --extract-with='tar zxf >FILE<'
                             and on Windows use, for example:
                               --extract-with="\"c:\Program Files\WinZip\WinZip32.exe\" -e -o >FILE< ."
                             (if WinZip is installed there).
   --list-file=<file>        Take the list of file and/or directory names to
                             process from <file>, which has one file/directory
                             name per line.  Only exact matches are counted;
                             relative path names will be resolved starting from
                             the directory where cloc is invoked.
                             See also --exclude-list-file.
   --vcs=<VCS>               Invoke a system call to <VCS> to obtain a list of
                             files to work on.  If <VCS> is 'git', then will
                             invoke 'git ls-files' to get a file list and
                             'git submodule status' to get a list of submodules
                             whose contents will be ignored.  See also --git
                             which accepts git commit hashes and branch names.
                             If <VCS> is 'svn' then will invoke 'svn list -R'.
                             The primary benefit is that cloc will then skip
                             files explicitly excluded by the versioning tool
                             in question, ie, those in .gitignore or have the
                             svn:ignore property.
                             Alternatively <VCS> may be any system command
                             that generates a list of files.
```

```
                           Note:  cloc must be in a directory which can read
                           the files as they are returned by <VCS>.  cloc will
                           not download files from remote repositories.
                           'svn list -R' may refer to a remote repository
                           to obtain file names (and therefore may require
                           authentication to the remote repository), but
                           the files themselves must be local.
  --unicode                Check binary files to see if they contain Unicode
                           expanded ASCII text.  This causes performance to
                           drop noticeably.

Processing Options
  --autoconf               Count .in files (as processed by GNU autoconf) of
                           recognized languages.  See also --no-autogen.
  --by-file                Report results for every source file encountered.
  --by-file-by-lang        Report results for every source file encountered
                           in addition to reporting by language.
  --config <file>          Read command line switches from <file> instead of
                           the default location of /home/al/.config/cloc/options.txt.
                           The file should contain one switch, along with
                           arguments (if any), per line.  Blank lines and lines
                           beginning with '#' are skipped.  Options given on
                           the command line take priority over entries read from
                           the file.
  --count-and-diff <set1> <set2>
                           First perform direct code counts of source file(s)
                           of <set1> and <set2> separately, then perform a diff
                           of these.  Inputs may be pairs of files, directories,
                           or archives.  If --out or --report-file is given,
                           three output files will be created, one for each
                           of the two counts and one for the diff.  See also
                           --diff, --diff-alignment, --diff-timeout,
                           --ignore-case, --ignore-whitespace.
  --diff <set1> <set2>     Compute differences in code and comments between
                           source file(s) of <set1> and <set2>.  The inputs
                           may be any mix of files, directories, archives,
                           or git commit hashes.  Use --diff-alignment to
                           generate a list showing which file pairs where
                           compared.  See also --count-and-diff, --diff-alignment,
                           --diff-timeout, --ignore-case, --ignore-whitespace.
  --diff-timeout <N>       Ignore files which take more than <N> seconds
                           to process.  Default is 10 seconds.  Setting <N>
                           to 0 allows unlimited time.  (Large files with many
                           repeated lines can cause Algorithm::Diff::sdiff()
                           to take hours.)
```

```
--follow-links            [Unix only] Follow symbolic links to directories
                          (sym links to files are always followed).
--force-lang=<lang>[,<ext>]
                          Process all files that have a <ext> extension
                          with the counter for language <lang>.  For
                          example, to count all .f files with the
                          Fortran 90 counter (which expects files to
                          end with .f90) instead of the default Fortran 77
                          counter, use
                            --force-lang="Fortran 90",f
                          If <ext> is omitted, every file will be counted
                          with the <lang> counter.  This option can be
                          specified multiple times (but that is only
                          useful when <ext> is given each time).
                          See also --script-lang, --lang-no-ext.
--force-lang-def=<file>   Load language processing filters from <file>,
                          then use these filters instead of the built-in
                          filters.  Note:  languages which map to the same
                          file extension (for example:
                          MATLAB/Mathematica/Objective C/MUMPS/Mercury;
                          Pascal/PHP; Lisp/OpenCL; Lisp/Julia; Perl/Prolog)
                          will be ignored as these require additional
                          processing that is not expressed in language
                          definition files.  Use --read-lang-def to define
                          new language filters without replacing built-in
                          filters (see also --write-lang-def,
                          --write-lang-def-incl-dup).
--git                     Forces the inputs to be interpreted as git targets
                          (commit hashes, branch names, et cetera) if these
                          are not first identified as file or directory
                          names.  This option overrides the --vcs=git logic
                          if this is given; in other words, --git gets its
                          list of files to work on directly from git using
                          the hash or branch name rather than from
                          'git ls-files'.  This option can be used with
                          --diff to perform line count diffs between git
                          commits, or between a git commit and a file,
                          directory, or archive.  Use -v/--verbose to see
                          the git system commands cloc issues.
--ignore-whitespace       Ignore horizontal white space when comparing files
                          with --diff.  See also --ignore-case.
--ignore-case             Ignore changes in case; consider upper- and lower-
                          case letters equivalent when comparing files with
                          --diff.  See also --ignore-whitespace.
--lang-no-ext=<lang>      Count files without extensions using the <lang>
```

```
                          counter.  This option overrides internal logic
                          for files without extensions (where such files
                          are checked against known scripting languages
                          by examining the first line for #!).  See also
                          --force-lang, --script-lang.
--max-file-size=<MB>      Skip files larger than <MB> megabytes when
                          traversing directories.  By default, <MB>=100.
                          cloc's memory requirement is roughly twenty times
                          larger than the largest file so running with
                          files larger than 100 MB on a computer with less
                          than 2 GB of memory will cause problems.
                          Note:  this check does not apply to files
                          explicitly passed as command line arguments.
--no-autogen[=list]       Ignore files generated by code-production systems
                          such as GNU autoconf.  To see a list of these files
                          (then exit), run with --no-autogen list
                          See also --autoconf.
--original-dir            [Only effective in combination with
                          --strip-comments]  Write the stripped files
                          to the same directory as the original files.
--read-binary-files       Process binary files in addition to text files.
                          This is usually a bad idea and should only be
                          attempted with text files that have embedded
                          binary data.
--read-lang-def=<file>    Load new language processing filters from <file>
                          and merge them with those already known to cloc.
                          If <file> defines a language cloc already knows
                          about, cloc's definition will take precedence.
                          Use --force-lang-def to over-ride cloc's
                          definitions (see also --write-lang-def,
                          --write-lang-def-incl-dup).
--script-lang=<lang>,<s>  Process all files that invoke <s> as a #!
                          scripting language with the counter for language
                          <lang>.  For example, files that begin with
                             #!/usr/local/bin/perl5.8.8
                          will be counted with the Perl counter by using
                             --script-lang=Perl,perl5.8.8
                          The language name is case insensitive but the
                          name of the script language executable, <s>,
                          must have the right case.  This option can be
                          specified multiple times.  See also --force-lang,
                          --lang-no-ext.
--sdir=<dir>              Use <dir> as the scratch directory instead of
                          letting File::Temp chose the location.  Files
                          written to this location are not removed at
```

```
                          the end of the run (as they are with File::Temp).
--skip-uniqueness         Skip the file uniqueness check.  This will give
                          a performance boost at the expense of counting
                          files with identical contents multiple times
                          (if such duplicates exist).
--stdin-name=<file>       Give a file name to use to determine the language
                          for standard input.  (Use - as the input name to
                          receive source code via STDIN.)
--strip-comments=<ext>    For each file processed, write to the current
                          directory a version of the file which has blank
                          and commented lines removed (in-line comments
                          persist).  The name of each stripped file is the
                          original file name with .<ext> appended to it.
                          It is written to the current directory unless
                          --original-dir is on.
--strip-str-comments      Replace comment markers embedded in strings with
                          'xx'.  This attempts to work around a limitation
                          in Regexp::Common::Comment where comment markers
                          embedded in strings are seen as actual comment
                          markers and not strings, often resulting in a
                          'Complex regular subexpression recursion limit'
                          warning and incorrect counts.  There are two
                          disadvantages to using this switch:  1/code count
                          performance drops, and 2/code generated with
                          --strip-comments will contain different strings
                          where ever embedded comments are found.
--sum-reports             Input arguments are report files previously
                          created with the --report-file option.  Makes
                          a cumulative set of results containing the
                          sum of data from the individual report files.
--processes=NUM           [Available only on systems with a recent version
                          of the Parallel::ForkManager module.  Not
                          available on Windows.] Sets the maximum number of
                          cores that cloc uses.  The default value of 0
                          disables multiprocessing.
--unix                    Override the operating system autodetection
                          logic and run in UNIX mode.  See also
                          --windows, --show-os.
--use-sloccount           If SLOCCount is installed, use its compiled
                          executables c_count, java_count, pascal_count,
                          php_count, and xml_count instead of cloc's
                          counters.  SLOCCount's compiled counters are
                          substantially faster than cloc's and may give
                          a performance improvement when counting projects
                          with large files.  However, these cloc-specific
```

```
                            features will not be available: --diff,
                            --count-and-diff, --strip-comments, --unicode.
  --windows                 Override the operating system autodetection
                            logic and run in Microsoft Windows mode.
                            See also --unix, --show-os.

Filter Options
  --exclude-dir=<D1>[,D2,]  Exclude the given comma separated directories
                            D1, D2, D3, et cetera, from being scanned.  For
                            example  --exclude-dir=.cache,test  will skip
                            all files and subdirectories that have /.cache/
                            or /test/ as their parent directory.
                            Directories named .bzr, .cvs, .hg, .git, .svn,
                            and .snapshot are always excluded.
                            This option only works with individual directory
                            names so including file path separators is not
                            allowed.  Use --fullpath and --not-match-d=<regex>
                            to supply a regex matching multiple subdirectories.
  --exclude-ext=<ext1>[,<ext2>[...]]
                            Do not count files having the given file name
                            extensions.
  --exclude-lang=<L1>[,L2[...]]
                            Exclude the given comma separated languages
                            L1, L2, L3, et cetera, from being counted.
  --exclude-list-file=<file>  Ignore files and/or directories whose names
                            appear in <file>.  <file> should have one file
                            name per line.  Only exact matches are ignored;
                            relative path names will be resolved starting from
                            the directory where cloc is invoked.
                            See also --list-file.
  --fullpath                Modifies the behavior of --match-f, --not-match-f,
                            and --not-match-d to include the file's path
                            in the regex, not just the file's basename.
                            (This does not expand each file to include its
                            absolute path, instead it uses as much of
                            the path as is passed in to cloc.)
                            Note:  --match-d always looks at the full
                            path and therefore is unaffected by --fullpath.
  --include-ext=<ext1>[,ext2[...]]
                            Count only languages having the given comma
                            separated file extensions.  Use --show-ext to
                            see the recognized extensions.
  --include-lang=<L1>[,L2[...]]
                            Count only the given comma separated languages
                            L1, L2, L3, et cetera.  Use --show-lang to see
```

```
                            the list of recognized languages.
  --match-d=<regex>         Only count files in directories matching the Perl
                            regex.  For example
                              --match-d='/(src|include)/'
                            only counts files in directories containing
                            /src/ or /include/.  Unlike --not-match-d,
                            --match-f, and --not-match-f, --match-d always
                            compares the fully qualified path against the
                            regex.
  --not-match-d=<regex>     Count all files except those in directories
                            matching the Perl regex.  Only the trailing
                            directory name is compared, for example, when
                            counting in /usr/local/lib, only 'lib' is
                            compared to the regex.
                            Add --fullpath to compare parent directories to
                            the regex.
                            Do not include file path separators at the
                            beginning or end of the regex.
  --match-f=<regex>         Only count files whose basenames match the Perl
                            regex.  For example
                              --match-f='^[Ww]idget'
                            only counts files that start with Widget or widget.
                            Add --fullpath to include parent directories
                            in the regex instead of just the basename.
  --not-match-f=<regex>     Count all files except those whose basenames
                            match the Perl regex.  Add --fullpath to include
                            parent directories in the regex instead of just
                            the basename.
  --skip-archive=<regex>    Ignore files that end with the given Perl regular
                            expression.  For example, if given
                              --skip-archive='(zip|tar(.(gz|Z|bz2|xz|7z))?)'
                            the code will skip files that end with .zip,
                            .tar, .tar.gz, .tar.Z, .tar.bz2, .tar.xz, and
                            .tar.7z.
  --skip-win-hidden         On Windows, ignore hidden files.

Debug Options
  --categorized=<file>      Save names of categorized files to <file>.
  --counted=<file>          Save names of processed source files to <file>.
  --diff-alignment=<file>   Write to <file> a list of files and file pairs
                            showing which files were added, removed, and/or
                            compared during a run with --diff.  This switch
                            forces the --diff mode on.
  --explain=<lang>          Print the filters used to remove comments for
                            language <lang> and exit.  In some cases the
```

```
                           filters refer to Perl subroutines rather than
                           regular expressions.  An examination of the
                           source code may be needed for further explanation.
  --help                   Print this usage information and exit.
  --found=<file>           Save names of every file found to <file>.
  --ignored=<file>         Save names of ignored files and the reason they
                           were ignored to <file>.
  --print-filter-stages    Print processed source code before and after
                           each filter is applied.
  --show-ext[=<ext>]       Print information about all known (or just the
                           given) file extensions and exit.
  --show-lang[=<lang>]     Print information about all known (or just the
                           given) languages and exit.
  --show-os                Print the value of the operating system mode
                           and exit.  See also --unix, --windows.
  -v[=<n>]                 Verbose switch (optional numeric value).
  -verbose[=<n>]           Long form of -v.
  --version                Print the version of this program and exit.
  --write-lang-def=<file>  Writes to <file> the language processing filters
                           then exits.  Useful as a first step to creating
                           custom language definitions. Note: languages which
                           map to the same file extension will be excluded.
                           (See also --force-lang-def, --read-lang-def).
  --write-lang-def-incl-dup=<file>
                           Same as --write-lang-def, but includes duplicated
                           extensions.  This generates a problematic language
                           definition file because cloc will refuse to use
                           it until duplicates are removed.

Output Options
  --3                      Print third-generation language output.
                           (This option can cause report summation to fail
                           if some reports were produced with this option
                           while others were produced without it.)
  --by-percent  X          Instead of comment and blank line counts, show
                           these values as percentages based on the value
                           of X in the denominator:
                              X = 'c'   -> # lines of code
                              X = 'cm'  -> # lines of code + comments
                              X = 'cb'  -> # lines of code + blanks
                              X = 'cmb' -> # lines of code + comments + blanks
                           For example, if using method 'c' and your code
                           has twice as many lines of comments as lines
                           of code, the value in the comment column will
                           be 200%.  The code column remains a line count.
```

```
--csv                    Write the results as comma separated values.
--csv-delimiter=<C>      Use the character <C> as the delimiter for comma
                         separated files instead of ,.  This switch forces
--file-encoding=<E>      Write output files using the <E> encoding instead of
                         the default ASCII (<E> = 'UTF-7').  Examples: 'UTF-16',
                         'euc-kr', 'iso-8859-16'.  Known encodings can be
                         printed with
                           perl -MEncode -e 'print join("\n", Encode->encodings(":all")), "\n"'
--hide-rate              Do not show line and file processing rates in the
                         output header. This makes output deterministic.
--json                   Write the results as JavaScript Object Notation
                         (JSON) formatted output.
--md                     Write the results as Markdown-formatted text.
--out=<file>             Synonym for --report-file=<file>.
--progress-rate=<n>      Show progress update after every <n> files are
                         processed (default <n>=100).  Set <n> to 0 to
                         suppress progress output (useful when redirecting
                         output to STDOUT).
--quiet                  Suppress all information messages except for
                         the final report.
--report-file=<file>     Write the results to <file> instead of STDOUT.
--sql=<file>             Write results as SQL create and insert statements
                         which can be read by a database program such as
                         SQLite.  If <file> is -, output is sent to STDOUT.
--sql-append             Append SQL insert statements to the file specified
                         by --sql and do not generate table creation
                         statements.  Only valid with the --sql option.
--sql-project=<name>     Use <name> as the project identifier for the
                         current run.  Only valid with the --sql option.
--sql-style=<style>      Write SQL statements in the given style instead
                         of the default SQLite format.  Styles include
                         'Oracle' and 'Named_Columns'.
--sum-one                For plain text reports, show the SUM: output line
                         even if only one input file is processed.
--xml                    Write the results in XML.
--xsl=<file>             Reference <file> as an XSL stylesheet within
                         the XML output.  If <file> is 1 (numeric one),
                         writes a default stylesheet, cloc.xsl (or
                         cloc-diff.xsl if --diff is also given).
                         This switch forces --xml on.
--yaml                   Write the results in YAML.
```

(1%7D%7D%7D) (%7B%7B%7B1) # Recognized Languages ▲

```
prompt> cloc --show-lang

ABAP                          (abap)
ActionScript                  (as)
Ada                           (ada, adb, ads, pad)
ADSO/IDSM                     (adso)
Agda                          (agda, lagda)
AMPLE                         (ample, dofile, startup)
Ant                           (build.xml, build.xml)
ANTLR Grammar                 (g, g4)
Apex Class                    (cls)
Apex Trigger                  (trigger)
Arduino Sketch                (ino, pde)
AsciiDoc                      (adoc, asciidoc)
ASP                           (asa, asp)
ASP.NET                       (asax, ascx, asmx, aspx, master, sitemap, webinfo)
AspectJ                       (aj)
Assembly                      (asm, S, s)
AutoHotkey                    (ahk)
awk                           (awk)
Blade                         (blade.php)
Bourne Again Shell            (bash)
Bourne Shell                  (sh)
BrightScript                  (brs)
builder                       (xml.builder)
C                             (c, ec, pgc)
C Shell                       (csh, tcsh)
C#                            (cs)
C++                           (C, c++, cc, CPP, cpp, cxx, inl, pcc)
C/C++ Header                  (h, H, hh, hpp, hxx)
CCS                           (ccs)
Chapel                        (chpl)
Clean                         (dcl, icl)
Clojure                       (clj)
ClojureC                      (cljc)
ClojureScript                 (cljs)
CMake                         (cmake, CMakeLists.txt)
COBOL                         (cbl, CBL, cob, COB)
CoffeeScript                  (coffee)
ColdFusion                    (cfm)
ColdFusion CFScript           (cfc)
Coq                           (v)
Crystal                       (cr)
CSON                          (cson)
CSS                           (css)
```

```
Cucumber                  (feature)
CUDA                      (cu, cuh)
Cython                    (pyx)
D                         (d)
DAL                       (da)
Dart                      (dart)
DIET                      (dt)
diff                      (diff)
DITA                      (dita)
DOORS Extension Language  (dxl)
DOS Batch                 (bat, BAT, BTM, btm, cmd, CMD)
Drools                    (drl)
DTD                       (dtd)
dtrace                    (d)
ECPP                      (ecpp)
EEx                       (eex)
EJS                       (ejs)
Elixir                    (ex, exs)
Elm                       (elm)
Embedded Crystal          (ecr)
ERB                       (erb, ERB)
Erlang                    (erl, hrl)
Expect                    (exp)
F#                        (fsi, fs, fs)
F# Script                 (fsx)
Fennel                    (fnl)
Fish Shell                (fish)
Focus                     (focexec)
Forth                     (4th, e4, f83, fb, forth, fpm, fr, frt, ft, fth, rx, fs, f, for)
Fortran 77                (F, f77, F77, FOR, ftn, FTN, pfo, f, for)
Fortran 90                (f90, F90)
Fortran 95                (f95, F95)
Freemarker Template       (ftl)
FXML                      (fxml)
GDScript                  (gd)
Gencat NLS                (msg)
Glade                     (glade, ui)
GLSL                      (comp, frag, geom, glsl, tesc, tese, vert)
Go                        (go)
Gradle                    (gradle, gradle.kts)
Grails                    (gsp)
GraphQL                   (gql, graphql)
Groovy                    (gant, groovy)
Haml                      (haml)
Handlebars                (handlebars, hbs)
```

```
Harbour                 (hb)
Haskell                 (hs, lhs)
Haxe                    (hx)
HCL                     (hcl, nomad, tf)
HLSL                    (cg, cginc, hlsl, shader)
Hoon                    (hoon)
HTML                    (htm, html)
IDL                     (idl, pro)
Idris                   (idr)
Igor Pro                (ipf)
INI                     (ini)
InstallShield           (ism)
Java                    (java)
JavaScript              (es6, js)
JavaServer Faces        (jsf)
JCL                     (jcl)
JSON                    (json)
JSON5                   (json5)
JSP                     (jsp, jspf)
JSX                     (jsx)
Julia                   (jl)
Jupyter Notebook        (ipynb)
Kermit                  (ksc)
Korn Shell              (ksh)
Kotlin                  (kt, kts)
Lean                    (lean)
LESS                    (less)
lex                     (l)
LFE                     (lfe)
liquid                  (liquid)
Lisp                    (asd, el, lisp, lsp, cl, jl)
Literate Idris          (lidr)
LiveLink OScript        (oscript)
Logtalk                 (lgt, logtalk)
Lua                     (lua)
m4                      (ac, m4)
make                    (am, gnumakefile, Gnumakefile, Makefile, makefile, mk)
Mako                    (mako)
Markdown                (md)
Mathematica             (mt, wl, wlt, m)
MATLAB                  (m)
Maven                   (pom, pom.xml)
Modula3                 (i3, ig, m3, mg)
MSBuild script          (csproj, vbproj, vcproj, wdproj, wixproj)
MUMPS                   (mps, m)
```

```
Mustache                    (mustache)
MXML                        (mxml)
NAnt script                 (build)
NASTRAN DMAP                (dmap)
Nemerle                     (n)
Nim                         (nim)
Nix                         (nix)
Objective C                 (m)
Objective C++               (mm)
OCaml                       (ml, mli, mll, mly)
OpenCL                      (cl)
Oracle Forms                (fmt)
Oracle PL/SQL               (bod, fnc, prc, spc, trg)
Oracle Reports              (rex)
Pascal                      (dpr, p, pas)
Pascal/Puppet               (pp)
Patran Command Language     (pcl, ses)
Perl                        (perl, plh, plx, pm, pm6, pl)
PHP                         (php, php3, php4, php5, phtml)
PHP/Pascal                  (inc)
Pig Latin                   (pig)
PL/I                        (pl1)
PL/M                        (lit, plm)
PO File                     (po)
PowerBuilder                (sra, srf, srm, srs, sru, srw)
PowerShell                  (ps1, psd1, psm1)
ProGuard                    (pro)
Prolog                      (P, pl, pro)
Protocol Buffers            (proto)
Pug                         (pug)
PureScript                  (purs)
Python                      (py, pyw)
QML                         (qml)
Qt                          (ui)
Qt Linguist                 (ts)
Qt Project                  (pro)
R                           (r, R)
Racket                      (rkt, rktl, scrbl)
RAML                        (raml)
RapydScript                 (pyj)
Razor                       (cshtml)
ReasonML                    (re, rei)
reStructuredText            (rst)
Rexx                        (rexx)
Rmd                         (Rmd)
```

```
RobotFramework             (robot, tsv)
Ruby                       (rake, rb)
Ruby HTML                  (rhtml)
Rust                       (rs)
SAS                        (sas)
Sass                       (sass, scss)
Scala                      (scala)
Scheme                     (sc, sch, scm, sld, sls, ss)
sed                        (sed)
SKILL                      (il)
SKILL++                    (ils)
Slice                      (ice)
Slim                       (slim)
Smalltalk                  (st, cs)
Smarty                     (smarty, tpl)
Softbridge Basic           (sbl, SBL)
Solidity                   (sol)
SparForte                  (sp)
Specman e                  (e)
SQL                        (psql, sql, SQL)
SQL Data                   (data.sql)
SQL Stored Procedure       (spc.sql, spoc.sql, sproc.sql, udf.sql)
Standard ML                (fun, sig, sml)
Starlark                   (bzl)
Stata                      (do, DO)
Stylus                     (styl)
SVG                        (svg, SVG)
Swift                      (swift)
SWIG                       (i)
Tcl/Tk                     (itk, tcl, tk)
Teamcenter met             (met)
Teamcenter mth             (mth)
TeX                        (bst, dtx, sty, tex, cls)
TITAN Project File Information (tpd)
Titanium Style Sheet       (tss)
TOML                       (toml)
TTCN                       (ttcn, ttcn2, ttcn3, ttcnpp)
Twig                       (twig)
TypeScript                 (tsx, ts)
Unity-Prefab               (mat, prefab)
Vala                       (vala)
Vala Header                (vapi)
Velocity Template Language (vm)
Verilog-SystemVerilog      (sv, svh, v)
VHDL                       (VHD, vhd, vhdl, VHDL)
```

```
vim script                (vim)
Visual Basic              (bas, ctl, dsr, frm, vb, VB, vba, VBA, VBS, vbs, cls)
Visual Fox Pro            (SCA, sca)
Visualforce Component     (component)
Visualforce Page          (page)
Vuejs Component           (vue)
WebAssembly               (wast, wat)
Windows Message File      (mc)
Windows Module Definition (def)
Windows Resource File     (rc, rc2)
WiX include               (wxi)
WiX source                (wxs)
WiX string localization   (wxl)
XAML                      (xaml)
xBase                     (prg)
xBase Header              (ch)
XHTML                     (xhtml)
XMI                       (XMI, xmi)
XML                       (xml, XML)
XQuery                    (xq, xquery)
XSD                       (xsd, XSD)
XSLT                      (xsl, XSL, XSLT, xslt)
Xtend                     (xtend)
yacc                      (y)
YAML                      (yaml, yml)
zsh                       (zsh)
```

The above list can be customized by reading language definitions from a file with the `--read-lang-def` or `--force-lang-def` options.

These file extensions map to multiple languages:

- `cl` files could be Lisp or OpenCL
- `cls` files could be Visual Basic, TeX or Apex Class
- `cs` files could be C# or Smalltalk
- `d` files could be D or dtrace
- `f` files could be Fortran 77 or Forth
- `fnc` files could be Oracle PL or SQL
- `for` files could be Fortran 77 or Forth
- `fs` files could be F# or Forth
- `inc` files could be PHP or Pascal
- `itk` files could be Tcl or Tk
- `jl` files could be Lisp or Julia
- `lit` files could be PL or M
- `m` files could be MATLAB, Mathematica, Objective C, MUMPS or Mercury

- `p6` files could be Perl or Prolog
- `pl` files could be Perl or Prolog
- `PL` files could be Perl or Prolog
- `pp` files could be Pascal or Puppet
- `pro` files could be IDL, Qt Project, Prolog or ProGuard
- `ts` files could be TypeScript or Qt Linguist
- `ui` files could be Qt or Glade
- `v` files could be Verilog-SystemVerilog or Coq

cloc has subroutines that attempt to identify the correct language based on the file's contents for these special cases. Language identification accuracy is a function of how much code the file contains; .m files with just one or two lines for example, seldom have enough information to correctly distinguish between MATLAB, Mercury, MUMPS, or Objective C.

Languages with file extension collisions are difficult to customize with `--read-lang-def` or `--force-lang-def` as they have no mechanism to identify languages with common extensions. In this situation one must modify the cloc source code.

(1%7D%7D%7D) (%7B%7B%7B1)

How It Works ▲

`cloc` 's method of operation resembles SLOCCount's: First, create a list of files to consider. Next, attempt to determine whether or not found files contain recognized computer language source code. Finally, for files identified as source files, invoke language-specific routines to count the number of source lines.

A more detailed description:

1. If the input file is an archive (such as a .tar.gz or .zip file), create a temporary directory and expand the archive there using a system call to an appropriate underlying utility (tar, bzip2, unzip, etc) then add this temporary directory as one of the inputs. (This works more reliably on Unix than on Windows.)
2. Use File::Find (File::Find) to recursively descend the input directories and make a list of candidate file names. Ignore binary and zero-sized files.
3. Make sure the files in the candidate list have unique contents (first by comparing file sizes, then, for similarly sized files, compare MD5 hashes of the file contents with Digest::MD5). For each set of identical files, remove all but the first copy, as determined by a lexical sort, of identical files from the set. The removed files are not included in the report. (The `--skip-uniqueness` switch disables the uniqueness tests and forces all copies of files to be included in the report.) See also the `--ignored=` switch to see which files were ignored and why.
4. Scan the candidate file list for file extensions which `cloc` associates with programming languages (see the `--show-lang` and `--show-ext` options). Files which match are classified as containing source code for that language. Each file without an extensions is opened and its first line read to see if it is a Unix shell script (anything that begins with #!). If it is shell script, the file is classified by that scripting language (if the language is recognized). If the file does not have a recognized extension or is not a recognzied scripting language, the file is ignored.
5. All remaining files in the candidate list should now be source files for known programming languages. For each of these files:
    1. Read the entire file into memory.
    2. Count the number of lines (= $L_{original}$).

3. Remove blank lines, then count again (= L_{non_blank}).
4. Loop over the comment filters defined for this language. (For example, C++ has two filters: (1) remove lines that start with optional whitespace followed by // and (2) remove text between /* and */) Apply each filter to the code to remove comments. Count the left over lines (= L_{code}).
5. Save the counts for this language:
    - blank lines = $L_{original}$ - L_{non_blank}
    - comment lines = L_{non_blank} - L_{code}
    - code lines = L_{code}

The options modify the algorithm slightly. The `--read-lang-def` option for example allows the user to read definitions of comment filters, known file extensions, and known scripting languages from a file. The code for this option is processed between Steps 2 and 3.

(1%7D%7D%7D) (%7B%7B%7B1) # Advanced Use ▲ (1%7D%7D%7D) (%7B%7B%7B1) ## Remove Comments from Source Code ▲

How can you tell if cloc correctly identifies comments? One way to convince yourself cloc is doing the right thing is to use its `--strip-comments` option to remove comments and blank lines from files, then compare the stripped-down files to originals.

Let's try this out with the SQLite amalgamation, a C file containing all code needed to build the SQLite library along with a header file:

```
prompt> tar zxf sqlite-amalgamation-3.5.6.tar.gz
prompt> cd sqlite-3.5.6/
prompt> cloc --strip-comments=nc sqlite.c
       1 text file.
       1 unique file.
Wrote sqlite3.c.nc
       0 files ignored.

http://cloc.sourceforge.net v 1.03  T=1.0 s (1.0 files/s, 82895.0 lines/s)
-------------------------------------------------------------------------------
Language          files     blank   comment      code    scale   3rd gen. equiv
-------------------------------------------------------------------------------
C                     1      5167     26827     50901 x   0.77 =       39193.77
-------------------------------------------------------------------------------
```

The extension argument given to –strip-comments is arbitrary; here nc was used as an abbreviation for "no comments".

`cloc` removed over 31,000 lines from the file:

```
prompt> wc -l sqlite3.c sqlite3.c.nc
  82895 sqlite3.c
  50901 sqlite3.c.nc
 133796 total
prompt> echo "82895 - 50901" | bc
31994
```

We can now compare the original file, sqlite3.c and the one stripped of comments, sqlite3.c.nc with tools like diff or vimdiff and see what exactly `cloc` considered comments and blank lines. A rigorous proof that the stripped-down file contains the same C code as the original is to compile these files and compare checksums of the resulting object files.

First, the original source file:

```
prompt> gcc -c sqlite3.c
prompt> md5sum sqlite3.o
cce5f1a2ea27c7e44b2e1047e2588b49  sqlite3.o
```

Next, the version without comments:

```
prompt> mv sqlite3.c.nc sqlite3.c
prompt> gcc -c sqlite3.c
prompt> md5sum sqlite3.o
cce5f1a2ea27c7e44b2e1047e2588b49  sqlite3.o
```

`cloc` removed over 31,000 lines of comments and blanks but did not modify the source code in any significant way since the resulting object file matches the original. (1%7D%7D%7D) (%7B%7B%7B1) ## Work with Compressed Archives ▲ Versions of `cloc` before v1.07 required an `--extract-with=CMD` option to tell `cloc` how to expand an archive file. Beginning with v1.07 this is extraction is attempted automatically. At the moment the automatic extraction method works reasonably well on Unix-type OS's for the following file types: `.tar.gz` , `.tar.bz2` , `.tar.xz` , `.tgz` , `.zip` , `.ear` , `.deb` . Some of these extensions work on Windows if one has WinZip installed in the default location (`C:\Program Files\WinZip\WinZip32.exe`). Additionally, with newer versions of WinZip, the http://www.winzip.com/downcl.htm (command%20line%20add-on) is needed for correct operation; in this case one would invoke `cloc` with something like

```
 --extract-with="\"c:\Program Files\WinZip\wzunzip\" -e -o >FILE< ."
```

Ref. http://sourceforge.net/projects/cloc/forums/forum/600963/topic/4021070?message=8938196 (http://sourceforge.net/projects/cloc/forums/forum/600963/topic/4021070?message=8938196)

In situations where the automatic extraction fails, one can try the `--extract-with=CMD` option to count lines of code within tar files, Zip files, or other compressed archives for which one has an extraction tool. `cloc` takes the user-provided extraction command and expands the archive to a temporary directory (created with File::Temp (File::Temp)), counts the lines of code in the temporary directory, then removes that directory. While not especially helpful when dealing with a single compressed archive (after all, if you're going to type the extraction command anyway why not just manually expand the archive?) this option is handy for working with several archives at once.

For example, say you have the following source tarballs on a Unix machine

```
perl-5.8.5.tar.gz
Python-2.4.2.tar.gz
```

and you want to count all the code within them. The command would be

```
cloc --extract-with='gzip -dc >FILE< | tar xf -' perl-5.8.5.tar.gz Python-2.4.2.tar.gz
```

If that Unix machine has GNU tar (which can uncompress and extract in one step) the command can be shortened to

```
cloc --extract-with='tar zxf >FILE<' perl-5.8.5.tar.gz Python-2.4.2.tar.gz
```

On a Windows computer with WinZip installed in `c:\Program Files\WinZip` the command would look like

```
cloc.exe --extract-with="\"c:\Program Files\WinZip\WinZip32.exe\" -e -o >FILE< ." perl-5.8.5.tar.gz Python-2.4.2.ta
```

Java `.ear` files are Zip files that contain additional Zip files. cloc can handle nested compressed archives without difficulty–provided all such files are compressed and archived in the same way. Examples of counting a Java `.ear` file in Unix and Windows:

```
Unix> cloc --extract-with="unzip -d . >FILE< " Project.ear
DOS> cloc.exe --extract-with="\"c:\Program Files\WinZip\WinZip32.exe\" -e -o >FILE< ." Project.ear
```

(1%7D%7D%7D) (%7B%7B%7B1) ## Differences ▲ The `--diff` switch allows one to measure the relative change in source code and comments between two versions of a file, directory, or archive. Differences reveal much more than absolute code counts of two file versions. For example, say a source file has 100 lines and its developer delivers a newer version with 102 lines. Did the developer add two comment lines, or delete seventeen source lines and add fourteen source lines and five comment lines, or did the developer do a complete rewrite, discarding all 100 original lines and adding 102 lines of all new source? The diff option tells how many lines of source were added, removed, modified or stayed the same, and how many lines of comments were added, removed, modified or stayed the same.

Differences in blank lines are handled much more coarsely because these are stripped by cloc early on. Unless a file pair is identical, cloc will report only differences in absolute counts of blank lines. In other words, one can expect to see only entries for ‘added’ if the second file has more blanks than the first, and ‘removed’ if the situation is reversed. The entry for ‘same’ will be non-zero only when the two files are identical.

In addition to file pairs, one can give cloc pairs of directories, or pairs of file archives, or a file archive and a directory. cloc will try to align file pairs within the directories or archives and compare diffs for each pair. For example, to see what changed between GCC 4.4.0 and 4.5.0 one could do

```
cloc --diff gcc-4.4.0.tar.bz2  gcc-4.5.0.tar.bz2
```

Be prepared to wait a while for the results though; the `--diff` option runs much more slowly than an absolute code count.

To see how cloc aligns files between the two archives, use the `--diff-alignment` option

```
cloc --diff-aligment=align.txt gcc-4.4.0.tar.bz2  gcc-4.5.0.tar.bz2
```

to produce the file `align.txt` which shows the file pairs as well as files added and deleted. The symbols `==` and `!=` before each file pair indicate if the files are identical (`==`) or if they have different content (`!=`).

Here’s sample output showing the difference between the Python 2.6.6 and 2.7 releases:

```
cloc --diff Python-2.7.9.tgz Python-2.7.10.tar.xz
    4315 text files.
    4313 text files.s
    2173 files ignored.

4 errors:
Diff error, exceeded timeout:  /tmp/8ToGAnB9Y1/Python-2.7.9/Mac/Modules/qt/_Qtmodule.c
Diff error, exceeded timeout:  /tmp/M6ldvsGaoq/Python-2.7.10/Mac/Modules/qt/_Qtmodule.c
Diff error (quoted comments?):  /tmp/8ToGAnB9Y1/Python-2.7.9/Mac/Modules/qd/qdsupport.py
Diff error (quoted comments?):  /tmp/M6ldvsGaoq/Python-2.7.10/Mac/Modules/qd/qdsupport.py

https://github.com/AlDanial/cloc v 1.65  T=298.59 s (0.0 files/s, 0.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Visual Basic
 same                            2              0              1             12
 modified                        0              0              0              0
 added                           0              0              0              0
 removed                         0              0              0              0
make
 same                           11              0            340           2952
 modified                        1              0              0              1
 added                           0              0              0              0
 removed                         0              0              0              0
diff
 same                            1              0             87            105
 modified                        0              0              0              0
 added                           0              0              0              0
 removed                         0              0              0              0
CSS
 same                            0              0             19            327
 modified                        1              0              0              1
 added                           0              0              0              0
 removed                         0              0              0              0
Objective C
 same                            7              0             61            635
 modified                        0              0              0              0
 added                           0              0              0              0
 removed                         0              0              0              0
NAnt script
 same                            2              0              0             30
 modified                        0              0              0              0
 added                           0              0              0              0
 removed                         0              0              0              0
```

```
XML
 same                          3               0               2              72
 modified                      1               0               0               1
 added                         0               0               0               1
 removed                       0               1               0               0
Windows Resource File
 same                          3               0              56             206
 modified                      1               0               0               1
 added                         0               0               0               0
 removed                       0               0               0               0
Expect
 same                          6               0             161             565
 modified                      0               0               0               0
 added                         0               0               0               0
 removed                       0               0               0               0
HTML
 same                         14               0              11            2344
 modified                      0               0               0               0
 added                         0               0               0               0
 removed                       0               0               0               0
vim script
 same                          1               0               7             106
 modified                      0               0               0               0
 added                         0               0               0               0
 removed                       0               0               0               0
C++
 same                          2               0              18             128
 modified                      0               0               0               0
 added                         0               0               0               0
 removed                       0               0               0               0
Windows Module Definition
 same                          7               0             187            2080
 modified                      2               0               0               0
 added                         0               0               0               1
 removed                       0               1               0               2
Prolog
 same                          1               0               0              24
 modified                      0               0               0               0
 added                         0               0               0               0
 removed                       0               0               0               0
Javascript
 same                          3               0              49             229
 modified                      0               0               0               0
 added                         0               0               0               0
 removed                       0               0               0               0
```

```
Assembly
 same                    51               0            6794           12298
 modified                 0               0               0               0
 added                    0               0               0               0
 removed                  0               0               0               0
Bourne Shell
 same                    41               0            7698           45024
 modified                 1               0               0               3
 added                    0              13               2              64
 removed                  0               0               0               0
DOS Batch
 same                    29               0             107             494
 modified                 1               0               0               9
 added                    0               1               0               3
 removed                  0               0               0               0
MSBuild script
 same                    77               0               3           38910
 modified                 0               0               0               0
 added                    0               0               0               0
 removed                  0               0               0               0
Python
 same                  1947               0          109012          430335
 modified               192               0              94             950
 added                    2             323             283            2532
 removed                  2              55              58             646
m4
 same                    18               0             191           15352
 modified                 1               0               0               2
 added                    1              31               0             205
 removed                  0               0               0               0
C
 same                   505               0           37439          347837
 modified                45               0              13             218
 added                    0              90              33             795
 removed                  0               9               2             148
C/C++ Header
 same                   255               0           10361           66635
 modified                 5               0               5               7
 added                    0               1               3             300
 removed                  0               0               0               0
-------------------------------------------------------------------------------
SUM:
 same                  2986               0          172604          966700
 modified               251               0             112            1193
 added                    3             459             321            3901
```

```
 removed                       2             66             60            796
-------------------------------------------------------------------------------
```

A pair of errors occurred.

The first pair was caused by timing out when computing diffs of the file `Python-X/Mac/Modules/qt/_Qtmodule.c` in each Python version.

This file has > 26,000 lines of C code and takes more than 10 seconds–the default maximum duration for diff'ing a single file–on my slow computer. (Note: this refers to performing differences with the `sdiff()` function in the Perl `Algorithm::Diff` module, not the command line `diff` utility.) This error can be overcome by raising the time to, say, 20 seconds with `--diff-timeout 20`.

The second error is more problematic. The files `Python-X/Mac/Modules/qd/qdsupport.py` include Python `docstring` (text between pairs of triple quotes) containing C comments. `cloc` treats `docstrings` as comments and handles them by first converting them to C comments, then using the C comment removing regular expression. Nested C comments yield erroneous results however.

(1%7D%7D%7D) (%7B%7B%7B1)

Create Custom Language Definitions ▲

`cloc` can write its language comment definitions to a file or can read comment definitions from a file, overriding the built-in definitions. This can be useful when you want to use `cloc` to count lines of a language not yet included, to change association of file extensions to languages, or to modify the way existing languages are counted.

The easiest way to create a custom language definition file is to make `cloc` write its definitions to a file, then modify that file:

```
Unix> cloc --write-lang-def=my_definitions.txt
```

creates the file `my_definitions.txt` which can be modified then read back in with either the `--read-lang-def` or `--force-lang-def` option. The difference between the options is former merges language definitions from the given file in with `cloc`'s internal definitions with `cloc`'s taking precedence if there are overlaps. The `--force-lang-def` option, on the other hand, replaces `cloc`'s definitions completely. This option has a disadvantage in preventing `cloc` from counting

```
<a class="u" href="#extcollision" name="extcollision">
```

languages whose extensions map to multiple languages as these languages require additional logic that is not easily expressed in a definitions file.

```
Unix> cloc --read-lang-def=my_definitions.txt  file1 file2 dir1 ...
```

Each language entry has four parts: * The language name starting in column 1. * One or more comment *filters* starting in column 5. * One or more filename extensions starting in column 5. * A 3rd generation scale factor starting in column 5. This entry must be provided but its value is not important unless you want to compare your language to a hypothetical third generation programming language.

A filter defines a method to remove comment text from the source file. For example the entry for C++ looks like this

```
    C++
    filter call_regexp_common C++
    filter remove_inline //.*$
    extension C
    extension c++
    extension cc
    extension cpp
    extension cxx
    extension pcc
    3rd_gen_scale 1.51
    end_of_line_continuation \\$
```

C++ has two filters: first, remove lines matching Regexp::Common's C++ comment regex. The second filter using remove_inline is currently unused. Its intent is to identify lines with both code and comments and it may be implemented in the future.

A more complete discussion of the different filter options may appear here in the future. The output of `cloc` 's `--write-lang-def` option should provide enough examples for motivated individuals to modify or extend `cloc` 's language definitions.

(1%7D%7D%7D) (%7B%7B%7B1) ## Combine Reports ▲

If you manage multiple software projects you might be interested in seeing line counts by project, not just by language. Say you manage three software projects called MariaDB, PostgreSQL, and SQLite. The teams responsible for each of these projects run `cloc` on their source code and provide you with the output.

For example MariaDB team does

```
cloc --out mariadb-10.1.txt mariadb-server-10.1.zip
```

and provides you with the file `mariadb-10.1.txt` . The contents of the three files you get are

```
cat mariadb-10.1.txt
https://github.com/AlDanial/cloc v 1.65  T=45.36 s (110.5 files/s, 66411.4 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                           1613         225338         290077         983026
C                              853          62442          73017         715018
C/C++ Header                  1327          48300         114577         209394
Bourne Shell                   256          10224          10810          61943
Perl                           147          10342           8305          35562
Pascal                         107           4907           5237          32541
HTML                            56            195              6          16489
Javascript                       5           3309           3019          15540
m4                              30           1599            359          14215
CMake                          190           1919           4097          12206
XML                             35            648             56           5210
Ruby                            59            619            184           4998
Puppet                          10              0              1           3848
make                           134            724            360           3631
SQL                             23            306            377           3405
Python                          34            371            122           2545
Bourne Again Shell              27            299            380           1604
Windows Module Definition       37             27             13           1211
lex                              4            394            166            991
yacc                             2            152             64            810
DOS Batch                       19             89             82            700
Prolog                           1              9             40            448
RobotFramework                   1              0              0            441
CSS                              2             33            155            393
JSON                             5              0              0            359
dtrace                           9             59            179            306
Windows Resource File           10             61             89            250
Assembly                         2             70            284            237
WiX source                       1             18             10            155
Visual Basic                     6              0              0             88
YAML                             2              4              4             65
PHP                              1             11              2             24
SKILL                            1              8             15             16
sed                              2              0              0             16
Windows Message File             1              2              8              6
diff                             1              1              4              4
D                                1              4             11              4
-------------------------------------------------------------------------------
SUM:                          5014         372484         512110        2127699
-------------------------------------------------------------------------------
```

```
<i>Unix&gt;</i> cat sqlite-3081101.txt
https://github.com/AlDanial/cloc v 1.65  T=1.22 s (3.3 files/s, 143783.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                                2          11059          53924         101454
C/C++ Header                     2            211           6630           1546
-------------------------------------------------------------------------------
SUM:                             4          11270          60554         103000
-------------------------------------------------------------------------------

<i>Unix&gt;</i> cat postgresql-9.4.4.txt
```

```
https://github.com/AlDanial/cloc v 1.65  T=22.46 s (172.0 files/s, 96721.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
HTML                          1254           3725              0         785991
C                             1139         139289         244045         736519
C/C++ Header                   667          12277          32488          57014
SQL                            410          13400           8745          51926
yacc                             8           3163           2669          28491
Bourne Shell                    41           2647           2440          17170
Perl                            81           1702           1308           9456
lex                              9            792           1631           4285
make                           205           1525           1554           4114
m4                              12            218             25           1642
Windows Module Definition       13              4             17           1152
XSLT                             5             76             55            294
DOS Batch                        7             29             30             92
CSS                              1             20              7             69
Assembly                         3             17             38             69
D                                1             14             14             66
Windows Resource File            3              4              0             62
Lisp                             1              1              1             16
sed                              1              1              7             15
Python                           1              5              0             13
Bourne Again Shell               1              8              6             10
Windows Message File             1              0              0              5
-------------------------------------------------------------------------------
SUM:                          3864         178917         295080        1698471
-------------------------------------------------------------------------------
```

While these three files are interesting, you also want to see the combined counts from all projects. That can be done with cloc's `--sum_reports` option:

```
$ --sum-reports --out=databases mariadb-10.1.txt  sqlite-3081101.txt  postgresql-9.4.4.txt
Wrote databases.lang
Wrote databases.file
```

The report combination produces two output files, one for sums by programming language (`databases.lang`) and one by project (`databases.file`). Their contents are

```
https://github.com/AlDanial/cloc v 1.65
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                             1994         212790         370986        1552991
C++                           1613         225338         290077         983026
HTML                          1310           3920              6         802480
C/C++ Header                  1996          60788         153695         267954
Bourne Shell                   297          12871          13250          79113
SQL                            433          13706           9122          55331
Perl                           228          12044           9613          45018
Pascal                         107           4907           5237          32541
yacc                            10           3315           2733          29301
m4                              42           1817            384          15857
Javascript                       5           3309           3019          15540
CMake                          190           1919           4097          12206
make                           339           2249           1914           7745
lex                             13           1186           1797           5276
XML                             35            648             56           5210
Ruby                            59            619            184           4998
Puppet                          10              0              1           3848
Python                          35            376            122           2558
Windows Module Definition       50             31             30           2363
Bourne Again Shell              28            307            386           1614
DOS Batch                       26            118            112            792
CSS                              3             53            162            462
Prolog                           1              9             40            448
RobotFramework                   1              0              0            441
JSON                             5              0              0            359
Windows Resource File           13             65             89            312
Assembly                         5             87            322            306
dtrace                           9             59            179            306
XSLT                             5             76             55            294
WiX source                       1             18             10            155
Visual Basic                     6              0              0             88
D                                2             18             25             70
YAML                             2              4              4             65
sed                              3              1              7             31
PHP                              1             11              2             24
SKILL                            1              8             15             16
Lisp                             1              1              1             16
Windows Message File             2              2              8             11
diff                             1              1              4              4
-------------------------------------------------------------------------------
```

```
SUM:                            8882         562671         867744        3929170
-------------------------------------------------------------------------------
```

```
cat databases.file
-------------------------------------------------------------------------------
File                            files          blank        comment           code
-------------------------------------------------------------------------------
mariadb-10.1.txt                 5014         372484         512110        2127699
postgresql-9.4.4.txt             3864         178917         295080        1698471
sqlite-3081101.txt                  4          11270          60554         103000
-------------------------------------------------------------------------------
SUM:                             8882         562671         867744        3929170
-------------------------------------------------------------------------------
```

Report files themselves can be summed together. Say you also manage development of Perl and Python and you want to keep track of those line counts separately from your database projects. First create reports for Perl and Python separately:

```
cloc --out perl-5.22.0.txt   perl-5.22.0.tar.gz
cloc --out python-2.7.10.txt Python-2.7.10.tar.xz
```

then sum these together with

```
cloc --sum-reports --out script_lang perl-5.22.0.txt python-2.7.10.txt
Wrote script_lang.lang
Wrote script_lang.file
```

```
cat script_lang.lang
https://github.com/AlDanial/cloc v 1.65
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Perl                          2892         136396         184362         536445
C                              680          75566          71211         531203
Python                        2141          89642         109524         434015
C/C++ Header                   408          16433          26938         214800
Bourne Shell                   154          11088          14496          87759
MSBuild script                  77              0              3          38910
m4                              20           1604            191          15559
Assembly                        51           3775           6794          12298
Pascal                           8            458           1603           8592
make                            16            897            828           4939
XML                             37            198              2           2484
HTML                            14            393             11           2344
C++                             12            338            295           2161
Windows Module Definition        9            171            187           2081
YAML                            49             20             15           2078
Prolog                          12            438              2           1146
JSON                            14              1              0           1037
yacc                             1             85             76            998
DOS Batch                       44            199            148            895
Objective C                      7             98             61            635
Expect                           6            104            161            565
Windows Message File             1            102             11            489
CSS                              1             98             19            328
Windows Resource File            7             55             56            292
Javascript                       3             31             49            229
vim script                       1             36              7            106
diff                             1             17             87            105
NAnt script                      2              1              0             30
IDL                              1              0              0             24
Visual Basic                     2              1              1             12
D                                1              5              7              8
Lisp                             2              0              3              4
-------------------------------------------------------------------------------
SUM:                          6674         338250         417148        1902571
-------------------------------------------------------------------------------

<i>Unix&gt;</i> cat script_lang.file
-------------------------------------------------------------------------------
File                         files          blank        comment           code
-------------------------------------------------------------------------------
```

```
python-2.7.10.txt              3240          161276          173214          998697
perl-5.22.0.txt                3434          176974          243934          903874
-------------------------------------------------------------------------------
SUM:                           6674          338250          417148         1902571
-------------------------------------------------------------------------------
```

Finally, combine the combination files:

```
cloc --sum-reports --report_file=everything databases.lang script_lang.lang
Wrote everything.lang
Wrote everything.file

cat everything.lang
https://github.com/AlDanial/cloc v 1.65
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                             2674         288356         442197        2084194
C++                           1625         225676         290372         985187
HTML                          1324           4313             17         804824
Perl                          3120         148440         193975         581463
C/C++ Header                  2404          77221         180633         482754
Python                        2176          90018         109646         436573
Bourne Shell                   451          23959          27746         166872
SQL                            433          13706           9122          55331
Pascal                         115           5365           6840          41133
MSBuild script                  77              0              3          38910
m4                              62           3421            575          31416
yacc                            11           3400           2809          30299
Javascript                       8           3340           3068          15769
make                           355           3146           2742          12684
Assembly                        56           3862           7116          12604
CMake                          190           1919           4097          12206
XML                             72            846             58           7694
lex                             13           1186           1797           5276
Ruby                            59            619            184           4998
Windows Module Definition       59            202            217           4444
Puppet                          10              0              1           3848
YAML                            51             24             19           2143
DOS Batch                       70            317            260           1687
Bourne Again Shell              28            307            386           1614
Prolog                          13            447             42           1594
JSON                            19              1              0           1396
CSS                              4            151            181            790
Objective C                      7             98             61            635
Windows Resource File           20            120            145            604
Expect                           6            104            161            565
Windows Message File             3            104             19            500
RobotFramework                   1              0              0            441
dtrace                           9             59            179            306
XSLT                             5             76             55            294
WiX source                       1             18             10            155
diff                             2             18             91            109
```

```
vim script                       1             36              7            106
Visual Basic                     8              1              1            100
D                                3             23             32             78
sed                              3              1              7             31
NAnt script                      2              1              0             30
IDL                              1              0              0             24
PHP                              1             11              2             24
Lisp                             3              1              4             20
SKILL                            1              8             15             16
-------------------------------------------------------------------------------
SUM:                         15556         900921        1284892        5831741
-------------------------------------------------------------------------------
```

```
cat everything.file
-------------------------------------------------------------------------------
File                          files          blank        comment           code
-------------------------------------------------------------------------------
databases.lang                 8882         562671         867744        3929170
script_lang.lang               6674         338250         417148        1902571
-------------------------------------------------------------------------------
SUM:                          15556         900921        1284892        5831741
-------------------------------------------------------------------------------
```

One limitation of the `--sum-reports` feature is that the individual counts must be saved in the plain text format. Counts saved as XML, JSON, YAML, or SQL will produce errors if used in a summation.

(1%7D%7D%7D) (%7B%7B%7B1) ## SQL ▲

`cloc` can write results in the form of SQL table create and insert statements for use with relational database programs such as SQLite, MySQL, PostgreSQL, Oracle, or Microsoft SQL. Once the code count information is in a database, the information can be interrogated and displayed in interesting ways.

A database created from `cloc` SQL output has two tables,

metadata and **t**:

Table **metadata**:

Field	Type
timestamp	text
project	text
elapsed_s	text

Table **t**:

Field	Type
project	text
language	text
file	text
nBlank	integer
nComment	integer
nCode	integer
nScaled	real

The **metadata** table contains information about when the `cloc` run was made. The `--sql-append` switch allows one to combine many runs in a single database; each run adds a row to the metadata table. The code count information resides in table **t**.

Let's repeat the code count examples of Perl, Python, SQLite, MySQL and PostgreSQL tarballs shown in the Combine Reports example above, this time using the SQL output options and the SQLite (http://www.sqlite.org/) database engine.

The `--sql` switch tells `cloc` to generate output in the form of SQL table `create` and `insert` commands. The switch takes an argument of a file name to write these SQL statements into, or, if the argument is 1 (numeric one), streams output to STDOUT.

Since the SQLite command line program, `sqlite3`, can read commands from STDIN, we can dispense with storing SQL statements to a file and use `--sql 1` to pipe data directly into the SQLite executable:

```
cloc --sql 1 --sql-project mariadb mariadb-server-10.1.zip | sqlite3 code.db
```

The `--sql-project mariadb` part is optional; there's no need to specify a project name when working with just one code base. However, since we'll be adding code counts from four other tarballs, we'll only be able to identify data by input source if we supply a project name for each run.

Now that we have a database we will need to pass in the `--sql-append` switch to tell cloc not to wipe out this database but instead add more data:

```
cloc --sql 1 --sql-project postgresql --sql-append postgresql-9.4.4.tar.bz2        | sqlite3 code.db
cloc --sql 1 --sql-project sqlite     --sql-append sqlite-amalgamation-3081101.zip | sqlite3 code.db
cloc --sql 1 --sql-project python     --sql-append Python-2.7.10.tar.xz            | sqlite3 code.db
cloc --sql 1 --sql-project perl       --sql-append perl-5.22.0.tar.gz              | sqlite3 code.db
```

Now the fun begins–we have a database, `code.db`, with lots of information about the five projects and can query it for all manner of interesting facts.

Which is the longest file over all projects?

```
prompt> sqlite3 code.db 'select project,file,nBlank+nComment+nCode as nL from t
                                 where nL = (select max(nBlank+nComment+nCode) from t)'

sqlite|sqlite-amalgamation-3081101/sqlite3.c|161623
```

`sqlite3` 's default output format leaves a bit to be desired. We can add an option to the program's `rc` file, `~/.sqliterc` , to show column headers:

```
  .header on
```

One might be tempted to also include

```
  .mode column
```

in `~/.sqliterc` but this causes problems when the output has more than one row since the widths of entries in the first row govern the maximum width for all subsequent rows. Often this leads to truncated output–not at all desirable. One option is to write a custom SQLite output formatter such as `sqlite_formatter` , included with `cloc` .

To use it, simply pass `sqlite3` 's STDOUT into `sqlite_formatter` via a pipe:

```
prompt> sqlite3 code.db 'select project,file,nBlank+nComment+nCode as nL from t
                         where nL = (select max(nBlank+nComment+nCode) from t)' | ./sqlite_formatter

  -- Loading resources from ~/.sqliterc
  Project File                                  nL
  _______ _____________________________________ ______
  sqlite  sqlite-amalgamation-3081101/sqlite3.c 161623
```

If the "Project File" line doesn't appear, add `.header on` to your `~/.sqliterc` file as explained above.

What is the longest file over all projects?

```
prompt> sqlite3 code.db 'select project,file,nBlank+nComment+nCode as nL from t
                         where nL = (select max(nBlank+nComment+nCode) from t)' | sqlite_formatter

Project File                                  nL
_______ _____________________________________ ______
sqlite  sqlite-amalgamation-3081101/sqlite3.c 161623
```

What is the longest file in each project?

```
prompt> sqlite3 code.db 'select project,file,max(nBlank+nComment+nCode) as nL from t
                         group by project order by nL;' | sqlite_formatter

Project    File                                                              nL
__________ ________________________________________________________________ ______
python     Python-2.7.10/Mac/Modules/qt/_Qtmodule.c                          28091
postgresql postgresql-9.4.4/src/interfaces/ecpg/preproc/preproc.c            54623
mariadb    server-10.1/storage/mroonga/vendor/groonga/lib/nfkc.c             80246
perl       perl-5.22.0/cpan/Locale-Codes/lib/Locale/Codes/Language_Codes.pm 100747
sqlite     sqlite-amalgamation-3081101/sqlite3.c                             161623
```

Which files in each project have the most code lines?

```
prompt> sqlite3 code.db 'select project,file,max(nCode) as nL from t
                        group by project order by nL desc;' | sqlite_formatter

Project    File                                                              nL
__________ ________________________________________________________________ ______
perl       perl-5.22.0/cpan/Locale-Codes/lib/Locale/Codes/Language_Codes.pm 100735
sqlite     sqlite-amalgamation-3081101/sqlite3.c                              97469
mariadb    server-10.1/storage/mroonga/vendor/groonga/lib/nfkc.c              80221
postgresql postgresql-9.4.4/src/interfaces/ecpg/preproc/preproc.c            45297
python     Python-2.7.10/Mac/Modules/qt/_Qtmodule.c                          26705
```

Which C source files with more than 300 lines have a comment ratio below 1%?

```
prompt> sqlite3 code.db 'select project, file, nCode, nComment,
                         (100.0*nComment)/(nComment+nCode) as comment_ratio from t
                         where language="C" and nCode > 300 and
                         comment_ratio < 1 order by comment_ratio;' | sqlite_formatter

Project    File                                                                                  nCode nC
_________ ____________________________________________________________________________________ _____ __
mariadb    server-10.1/storage/mroonga/vendor/groonga/lib/nfkc.c                                80221
python     Python-2.7.10/Python/graminit.c                                                       2175
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_turkish.c                 2095
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_french.c                  1211
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_french.c             1201
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_hungarian.c               1182
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_hungarian.c          1178
mariadb    server-10.1/strings/ctype-eucjpms.c                                                  67466
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_english.c                 1072
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_english.c            1064
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_spanish.c                 1053
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_spanish.c            1049
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_italian.c                 1031
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_italian.c            1023
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_portuguese.c              981
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_portuguese.c         975
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_romanian.c                967
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_2_romanian.c           961
mariadb    server-10.1/strings/ctype-ujis.c                                                     67177
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_finnish.c                 720
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_porter.c                  717
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_finnish.c            714
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_porter.c             711
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_KOI8_R_russian.c                660
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_russian.c                 654
python     Python-2.7.10/Mac/Modules/qt/_Qtmodule.c                                             26705
python     Python-2.7.10/Mac/Modules/icn/_Icnmodule.c                                           1521
mariadb    server-10.1/strings/ctype-extra.c                                                    8282
postgresql postgresql-9.4.4/src/bin/psql/sql_help.c                                             3576
mariadb    server-10.1/strings/ctype-sjis.c                                                     34006
python     Python-2.7.10/Python/Python-ast.c                                                    6554
mariadb    server-10.1/strings/ctype-cp932.c                                                    34609
perl       perl-5.22.0/keywords.c                                                               2815
python     Python-2.7.10/Mac/Modules/menu/_Menumodule.c                                         3263
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_dutch.c                   596
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_dutch.c              586
mariadb    server-10.1/strings/ctype-gbk.c                                                      10684
python     Python-2.7.10/Mac/Modules/qd/_Qdmodule.c                                             6694
```

```
python     Python-2.7.10/Mac/Modules/win/_Winmodule.c                                          3056
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_german.c                  476
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_german.c             470
mariadb    server-10.1/strings/ctype-euc_kr.c                                                   9956
postgresql postgresql-9.4.4/src/backend/utils/fmgrtab.c                                         4815
python     Python-2.7.10/Mac/Modules/ctl/_Ctlmodule.c                                           5442
python     Python-2.7.10/Mac/Modules/ae/_AEmodule.c                                             1347
python     Python-2.7.10/Mac/Modules/app/_Appmodule.c                                           1712
mariadb    server-10.1/strings/ctype-gb2312.c                                                   6377
mariadb    server-10.1/storage/tokudb/ft-index/third_party/xz-4.999.9beta/src/liblzma/lzma/fastpos_table.c   516
python     Python-2.7.10/Mac/Modules/evt/_Evtmodule.c                                           504
python     Python-2.7.10/Modules/expat/xmlrole.c                                                1256
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_UTF_8_danish.c                  312
postgresql postgresql-9.4.4/src/backend/snowball/libstemmer/stem_ISO_8859_1_danish.c             310
python     Python-2.7.10/Mac/Modules/res/_Resmodule.c                                           1621
python     Python-2.7.10/Mac/Modules/drag/_Dragmodule.c                                         1046
python     Python-2.7.10/Mac/Modules/list/_Listmodule.c                                         1021
python     Python-2.7.10/Mac/Modules/te/_TEmodule.c                                             1198
python     Python-2.7.10/Mac/Modules/cg/_CGmodule.c                                             1190
python     Python-2.7.10/Modules/clmodule.c                                                     2379
python     Python-2.7.10/Mac/Modules/folder/_Foldermodule.c                                      306
```

What are the ten longest files (based on code lines) that have no comments at all? Exclude header, .html, and YAML files.

```
prompt> sqlite3 code.db 'select project, file, nCode from t
                         where nComment = 0 and
                         language not in ("C/C++ Header", "YAML", "HTML")
                         order by nCode desc limit 10;' | sqlite_formatter

Project File                                                                 nCode
_______ ____________________________________________________________________ _____
perl    perl-5.22.0/cpan/Unicode-Collate/Collate/Locale/ja.pl                1938
python  Python-2.7.10/PCbuild/pythoncore.vcproj                               1889
python  Python-2.7.10/PC/VS8.0/pythoncore.vcproj                              1889
mariadb server-10.1/mysql-test/extra/binlog_tests/mysqlbinlog_row_engine.inc  1862
perl    perl-5.22.0/cpan/Unicode-Collate/Collate/Locale/zh_strk.pl            1589
perl    perl-5.22.0/cpan/Unicode-Collate/Collate/Locale/zh_zhu.pl             1563
mariadb server-10.1/storage/mroonga/vendor/groonga/configure.ac               1526
perl    perl-5.22.0/cpan/Unicode-Collate/Collate/Locale/zh_pin.pl             1505
mariadb server-10.1/mysql-test/suite/funcs_1/storedproc/storedproc_02.inc     1465
python  Python-2.7.10/PC/VS8.0/_bsddb.vcproj                                  1463
```

What are the most popular languages (in terms of lines of code) in each project?

```
prompt> sqlite3 code.db 'select project, language, sum(nCode) as SumCode from t
                         group by project,language
                         order by project,SumCode desc;' | sqlite_formatter
Project    Language                   SumCode
__________ _____________________________ _______
mariadb    C++                          983026
mariadb    C                            715018
mariadb    C/C++ Header                 209394
mariadb    Bourne Shell                  61943
mariadb    Perl                          35562
mariadb    Pascal                        32541
mariadb    HTML                          16489
mariadb    Javascript                    15540
mariadb    m4                            14215
mariadb    CMake                         12206
mariadb    XML                            5210
mariadb    Ruby                           4998
mariadb    Puppet                         3848
mariadb    make                           3631
mariadb    SQL                            3405
mariadb    Python                         2545
mariadb    Bourne Again Shell             1604
mariadb    Windows Module Definition      1211
mariadb    lex                             991
mariadb    yacc                            810
mariadb    DOS Batch                       700
mariadb    Prolog                          448
mariadb    RobotFramework                  441
mariadb    CSS                             393
mariadb    JSON                            359
mariadb    dtrace                          306
mariadb    Windows Resource File           250
mariadb    Assembly                        237
mariadb    WiX source                      155
mariadb    Visual Basic                     88
mariadb    YAML                             65
mariadb    PHP                              24
mariadb    SKILL                            16
mariadb    sed                              16
mariadb    Windows Message File              6
mariadb    D                                 4
mariadb    diff                              4
perl       Perl                         536445
perl       C                            155648
perl       C/C++ Header                 147858
```

```
perl       Bourne Shell                42668
perl       Pascal                       8592
perl       XML                          2410
perl       YAML                         2078
perl       C++                          2033
perl       make                         1986
perl       Prolog                       1146
perl       JSON                         1037
perl       yacc                          998
perl       Windows Message File          489
perl       DOS Batch                     389
perl       Windows Resource File          85
perl       D                               8
perl       Lisp                            4
postgresql HTML                       785991
postgresql C                          736519
postgresql C/C++ Header                57014
postgresql SQL                         51926
postgresql yacc                        28491
postgresql Bourne Shell                17170
postgresql Perl                         9456
postgresql lex                          4285
postgresql make                         4114
postgresql m4                           1642
postgresql Windows Module Definition    1152
postgresql XSLT                          294
postgresql DOS Batch                      92
postgresql Assembly                       69
postgresql CSS                            69
postgresql D                              66
postgresql Windows Resource File          62
postgresql Lisp                           16
postgresql sed                            15
postgresql Python                         13
postgresql Bourne Again Shell             10
postgresql Windows Message File            5
python     Python                     434015
python     C                          375555
python     C/C++ Header                66942
python     Bourne Shell                45091
python     MSBuild script              38910
python     m4                          15559
python     Assembly                    12298
python     make                         2953
python     HTML                         2344
```

```
python     Windows Module Definition    2081
python     Objective C                   635
python     Expect                        565
python     DOS Batch                     506
python     CSS                           328
python     Javascript                    229
python     Windows Resource File         207
python     C++                           128
python     vim script                    106
python     diff                          105
python     XML                            74
python     NAnt script                    30
python     Prolog                         24
python     Visual Basic                   12
sqlite     C                          101454
sqlite     C/C++ Header                 1546
```

(1%7D%7D%7D) (%7B%7B%7B1) ## Custom Column Output ▲ Cloc's default output is a text table with five columns: language, file count, number of blank lines, number of comment lines and number of code lines. The switches `--by-file` , `--3` , and `--by-percent` generate additional information but sometimes even those are insufficient.

The `--sql` option described in the previous section offers the ability to create custom output. This section has a pair of examples that show how to create custom columns. The first example includes an extra column, **Total**, which is the sum of the numbers of blank, comment, and code lines. The second shows how to include the language name when running with `--by-file` .

Example 1: Add a "Totals" column.

The first step is to run cloc and save the output to a relational database, SQLite in this case:

```
cloc --sql 1 --sql-project x yaml-cpp-yaml-cpp-0.5.3.tar.gz | sqlite3 counts.db
```

(the tar file comes from the YAML-C++ (https://github.com/jbeder/yaml-cpp) project).

Second, we craft an SQL query that returns the regular cloc output plus an extra column for totals, then save the SQL statement to a file, `query_with_totals.sql` :

```
-- file query_with_totals.sql
select Language, count(File)   as files                      ,
                 sum(nBlank)   as blank                      ,
                 sum(nComment) as comment                    ,
                 sum(nCode)    as code                       ,
                 sum(nBlank)+sum(nComment)+sum(nCode) as Total
    from t group by Language order by code desc;
```

Third, we run this query through SQLite using the `counts.db` database. We'll include the `-header` switch so that SQLite prints the column names:

```
> cat query_with_totals.sql | sqlite3 -header counts.db
Language|files|blank|comment|code|Total
C++|141|12786|17359|60378|90523
C/C++ Header|110|8566|17420|51502|77488
Bourne Shell|10|6351|6779|38264|51394
m4|11|2037|260|17980|20277
Python|30|1613|2486|4602|8701
MSBuild script|11|0|0|1711|1711
CMake|7|155|285|606|1046
make|5|127|173|464|764
Markdown|2|30|0|39|69
```

The extra column for **Total** is there but the format is unappealing. Running the output through `sqlite_formatter` yields the desired result:

```
> cat query_with_totals.sql | sqlite3 -header counts.db | sqlite_formatter
Language       files blank comment code  Total
_______________ _____ _____ _______ _____ _____
C++              141 12786   17359 60378 90523
C/C++ Header     110  8566   17420 51502 77488
Bourne Shell      10  6351    6779 38264 51394
m4                11  2037     260 17980 20277
Python            30  1613    2486  4602  8701
MSBuild script    11     0       0  1711  1711
CMake              7   155     285   606  1046
make               5   127     173   464   764
Markdown           2    30       0    39    69
```

The next section, Wrapping cloc in other scripts, shows one way these commands can be combined into a new utility program.

Example 2: Include a column for "Language" when running with `--by-file`.

Output from `--by-file` omits each file's language to save screen real estate; file paths for large projects can be long and including an extra 20 or so characters for a Language column can be excessive.

As an example, here are the first few lines of output using the same code base as in Example 1:

```
> cloc --by-file yaml-cpp-yaml-cpp-0.5.3.tar.gz
github.com/AlDanial/cloc v 1.81  T=1.14 s (287.9 files/s, 221854.9 lines/s)
-------------------------------------------------------------------------------------------------------------------------------
File                                                                                                                  blank
-------------------------------------------------------------------------------------------------------------------------------
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/configure                                                                      2580
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/configure                                                               2541
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/fused-src/gtest/gtest.h                                                 1972
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/fused-src/gmock/gmock.h                                                       1585
yaml-cpp-yaml-cpp-0.5.3/test/integration/gen_emitter_test.cpp                                                           999
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/aclocal.m4                                                                     987
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/m4/libtool.m4                                                            760
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/build-aux/ltmain.sh                                                      959
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/fused-src/gmock-gtest-all.cc                                                  1514
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/fused-src/gtest/gtest-all.cc                                            1312
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/test/gtest_unittest.cc                                                  1226
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/include/gtest/internal/gtest-param-util-generated.h                      349
```

The absence of language identification for each file is a bit disappointing, but this can be remedied with a custom column solution.

The first step, creating a database, matches that from Example 1 so we'll go straight to the second step of creating the desired SQL query. We'll store this one in the file `by_file_with_language.sql`:

```
-- file by_file_with_language.sql
select File, Language, nBlank   as blank  ,
                       nComment as comment,
                       nCode    as code
    from t order by code desc;
```

Our desired extra column appears when we pass this custom SQL query through our database:

```
> cat by_file_with_language.sql | sqlite3 -header counts.db | sqlite_formatter
File                                                                                                  Language       b
____________________________________________________________________________________________________  _____________  _
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/configure                                                     Bourne Shell    1
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/configure                                              Bourne Shell    1
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/fused-src/gtest/gtest.h                                C/C++ Header
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/fused-src/gmock/gmock.h                                      C/C++ Header
yaml-cpp-yaml-cpp-0.5.3/test/integration/gen_emitter_test.cpp                                         C++
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/aclocal.m4                                                   m4
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/m4/libtool.m4                                          m4
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/build-aux/ltmain.sh                                    Bourne Shell
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/fused-src/gmock-gtest-all.cc                                 C++
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/fused-src/gtest/gtest-all.cc                           C++
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/test/gtest_unittest.cc                                 C++
yaml-cpp-yaml-cpp-0.5.3/test/gmock-1.7.0/gtest/include/gtest/internal/gtest-param-util-generated.h    C/C++ Header
```

(1%7D%7D%7D) (%7B%7B%7B1) * ## Wrapping cloc in other scripts ▲

More complex code counting solutions are possible by wrapping cloc in scripts or programs. The “total lines” column from example 1 of Custom Column Output could be simplified to a single command with this shell script (on Linux):

```
#!/bin/sh
#
# These commands must be in the user's $PATH:
#   cloc
#   sqlite3
#   sqlite_formatter
#
if test $# -eq 0 ; then
    echo "Usage: $0  [cloc arguments]"
    echo "       Run cloc to count lines of code with an additional"
    echo "       output column for total lines (code+comment+blank)."
    exit
fi
DBFILE=`tempfile`
cloc --sql 1 --sql-project x $@ | sqlite3 ${DBFILE}
SQL="select Language, count(File)   as files                      ,
                      sum(nBlank)   as blank                      ,
                      sum(nComment) as comment                    ,
                      sum(nCode)    as code                       ,
                      sum(nBlank)+sum(nComment)+sum(nCode) as Total
         from t group by Language order by code desc;
"
echo ${SQL} | sqlite3 -header ${DBFILE} | sqlite_formatter
rm ${DBFILE}
```

Saving the lines above to `total_columns.sh` and making it executable (`chmod +x total_columns.sh`) would let us do

```
./total_columns.sh yaml-cpp-yaml-cpp-0.5.3.tar.gz
```

to directly get

```
Language       files blank comment code  Total
_______________ _____ _____ _______ _____ _____
C++              141 12786   17359 60378 90523
C/C++ Header     110  8566   17420 51502 77488
Bourne Shell      10  6351    6779 38264 51394
m4                11  2037     260 17980 20277
Python            30  1613    2486  4602  8701
MSBuild script    11     0       0  1711  1711
CMake              7   155     285   606  1046
make               5   127     173   464   764
Markdown           2    30       0    39    69
```

Other examples: * Count code from a specific branch of a web-hosted git repository and send the results as a .csv email attachment: https://github.com/dannyloweatx/checkmarx (https://github.com/dannyloweatx/checkmarx)

(1%7D%7D%7D) (%7B%7B%7B1) ## Third Generation Language Scale Factors ▲

cloc versions before 1.50 by default computed, for the provided inputs, a rough estimate of how many lines of code would be needed to write the same code in a hypothetical third-generation computer language. To produce this output one must now use the `--3` switch.

Scale factors were derived from the 2006 version of language gearing ratios listed at Mayes Consulting web site, http://softwareestimator.com/IndustryData2.htm (http://softwareestimator.com/IndustryData2.htm), using this equation:

cloc scale factor for language X = 3rd generation default gearing ratio / language X gearing ratio

For example, cloc 3rd generation scale factor for DOS Batch = 80 / 128 = 0.625.

The biggest flaw with this approach is that gearing ratios are defined for logical lines of source code not physical lines (which cloc counts). The values in cloc's 'scale' and '3rd gen. equiv.' columns should be taken with a large grain of salt.

(1%7D%7D%7D) (%7B%7B%7B1) # Complex regular subexpression recursion limit ▲ cloc relies on the Regexp::Common module's regular expressions to remove comments from source code. If comments are malformed, for example the `/*` start comment marker appears in a C program without a corresponding `*/` marker, the regular expression engine could enter a recursive loop, eventually triggering the warning `Complex regular subexpression recursion limit`.

The most common cause for this warning is the existence of comment markers in string literals. While language compilers and interpreters are smart enough to recognize that `"/*"` (for example) is a string and not a comment, cloc is fooled. File path globs, as in this line of JavaScript

```
var paths = globArray("**/*.js", {cwd: srcPath});
```

are frequent culprits.

In an attempt to overcome this problem, a different algorithm which removes comment markers in strings can be enabled with the `--strip-str-comments` switch. Doing so, however, has drawbacks: cloc will run more slowly and the output of `--strip-comments` will contain strings that no longer match the input source.

(1%7D%7D%7D) (%7B%7B%7B1) # Limitations ▲ Identifying comments within source code is trickier than one might expect. Many languages would need a complete parser to be counted correctly. cloc does not attempt to parse any of the languages it aims to count and therefore is an imperfect tool. The following are known problems:

1. Lines containing both source code and comments are counted as lines of code.
2. Comment markers within strings or here-documents (http://www.faqs.org/docs/abs/HTML/here-docs.html) are treated as actual comment markers and not string literals. For example the following lines of C code

   ```
   printf(" /* ");
   for (i = 0; i < 100; i++) {
       a += i;
   }
   printf(" */ ");
   ```

   look to cloc like this:

```
printf(" xxxxxxx
xxxxxxx
xxxxxxx
xxxxxxx
xxxxxxx     ");
```

where `xxxxxxx` represents cloc's view of commented text. Therefore cloc counts the five lines as two lines of C code and three lines of comments (lines with both code and comment are counted as code).

If you suspect your code has such strings, use the switch `--strip-str-comments` to switch to the algorithm which removes embedded comment markers. Its use will render the five lines above as

```
printf("  ");
for (i = 0; i < 100; i++) {
    a += i;
}
printf("  ");
```

and therefore return a count of five lines of code. See the previous section on drawbacks to using `--strip-str-comments`.

3. Embedded languages are not recognized. For example, an HTML file containing JavaScript will be counted entirely as HTML.
4. Python docstrings can serve several purposes. They may contain documentation, comment out blocks of code, or they can be regular strings (when they appear on the right hand side of an assignment or as a function argument). cloc is unable to infer the meaning of docstrings by context; by default cloc treats all docstrings as comments. The switch `--docstring-as--code` treats all docstrings as code.

(1%7D%7D%7D) (%7B%7B%7B1) # How to Request Support for Additional Languages ▲

If cloc does not recognize a language you are interested in counting, create a GitHub issue (https://github.com/AlDanial/cloc/issues) requesting support for your language. Include this information:

1. File extensions associated with the language. If the language does not rely on file extensions and instead works with fixed file names or with `#!` style program invocations, explain what those are.
2. A description of how comments are defined.
3. Links to sample code.

(1%7D%7D%7D) (%7B%7B%7B1) ## Features Currently in Development ▲

Here, in no particular order and with no promise of future delivery, are features and capabilities currently in development:

1. produce reStructuredText output with `--rst`
2. count code (e.g. Javascript) embedded in HTML files

Pull requests for these features will receive extra consideration.

(1%7D%7D%7D) (%7B%7B%7B1) # Acknowledgments ▲ Wolfram Rösler provided most of the code examples in the test suite. These examples come from his Hello World collection (http://helloworldcollection.de/).

Ismet Kursunoglu found errors with the MUMPS counter and provided access to a computer with a large body of MUMPS code to test cloc.

Tod Huggins gave helpful suggestions for the Visual Basic filters.

Anton Demichev found a flaw with the JSP counter in cloc v0.76 and wrote the XML output generator for the `--xml` option.

Reuben Thomas pointed out that ISO C99 allows `//` as a comment marker, provided code for the `--no3` and `--stdin-name` options, counting the m4 language, and suggested several user-interface enhancements.

Michael Bello provided code for the `--opt-match-f` , `--opt-not-match-f` , `--opt-match-d` , and `--opt-not-match-d` options.

Mahboob Hussain inspired the `--original-dir` and `--skip-uniqueness` options, found a bug in the duplicate file detection logic and improved the JSP filter.

Randy Sharo found and fixed an uninitialized variable bug for shell scripts having only one line.

Steven Baker found and fixed a problem with the YAML output generator.

Greg Toth provided code to improve blank line detection in COBOL.

Joel Oliveira provided code to let `--exclude-list-file` handle directory name exclusion.

Blazej Kroll provided code to produce an XSLT file, `cloc-diff.xsl` , when producing XML output for the `--diff` option.

Denis Silakov enhanced the code which generates `cloc.xsl` when using `--by-file` and `--by-file-by-lang` options, and provided an XSL file that works with `--diff` output.

Andy (awalshe@sf.net (mailto:awalshe@sf.net)) provided code to fix several bugs: correct output of `--counted` so that only files that are used in the code count appear and that results are shown by language rather than file name; allow `--diff` output from multiple runs to be summed together with `--sum-reports` .

Jari Aalto created the initial version of `cloc.1.pod` and maintains the Debian package for cloc.

Mikkel Christiansen (mikkels@gmail.com (mailto:mikkels@gmail.com)) provided counter definitions for Clojure and ClojureScript.

Vera Djuraskovic from Webhostinggeeks.com (http://webhostinggeeks.com/) provided the Serbo-Croatian (http://science.webhostinggeeks.com/cloc) translation.

Gill Ajoft of Ajoft Softwares (http://www.ajoft.com) provided the Bulgarian (http://www.ajoft.com/wpaper/aj-cloc.html) translation.

The Knowledge Team (http://newknowledgez.com/) provided the Slovakian (http://newknowledgez.com/cloc.html) translation.

Erik Gooven Arellano Casillas provided an update to the MXML counter to recognize Actionscript comments.

Gianluca Casati (http://g14n.info) created the cloc CPAN package (https://metacpan.org/pod/App::cloc).

Mary Stefanova provided the Polish (http://www.trevister.com/blog/cloc.html) translation.

Ryan Lindeman implemented the `--by-percent` feature.

Kent C. Dodds, [@kentcdodds](https://twitter.com/kentcdodd (https://twitter.com/kentcdodd)), created and maintains the npm package of cloc.

Viktoria Parnak (http://kudoybook.com) provided the Ukrainian (http://blog.kudoybook.com/cloc/) translation.

Natalie Harmann provided the Belarussian (http://www.besteonderdelen.nl/blog/?p=5426) translation.

Nithyal at Healthcare Administration Portal (http://healthcareadministrationdegree.co/) provided the Tamil (http://healthcareadministrationdegree.co/socialwork/aldanial-cloc/) translation.

Patricia Motosan provided the Romanian (http://www.bildelestore.dk/blog/cloc-contele-de-linii-de-cod/) translation.

The Garcinia Cambogia Review Team (http://www.garciniacambogiareviews.ca/) provided the Arabic translation (http://www.garciniacambogiareviews.ca/translations/aldanial-cloc/).

Gajk Melikyan provided the provided the Armenian translation (http://students.studybay.com/?p=34) for http://studybay.com (http://studybay.com).

Hungarian translation (http://www.forallworld.com/cloc-grof-sornyi-kodot/) courtesy of Zsolt Boros (http://www.forallworld.com/).

Sietse Snel (https://github.com/stsnel) implemented the parallel processing capability available with the `--processes=N` switch.

The development of cloc was partially funded by the Northrop Grumman Corporation.

(1%7D%7D%7D) (%7B%7B%7B1) # Copyright ▲ Copyright (c) 2006-2018, Al Danial (https://github.com/AlDanial) (1%7D%7D%7D)