# Enhancing GNU `grep`

Tony Abou-Assaleh
Faculty of Computer Science
Dalhousie University
Halifax, NS, B3H 1W5, Canada
*taa@acm.org*

Wei Ai
Faculty of Computer Science
Dalhousie University
Halifax, NS, B3H 1W5, Canada
*weia@cs.dal.ca*

April 27, 2004

**Abstract**

The UNIX `grep` utility searches the input files selecting lines matching one or more patterns. Searching for patterns in text is an important operation in a number of domains, including program comprehension and software maintenance, structured text databases, indexing file systems, and searching natural language texts. Such a wide range of uses inspired the development of variations of the original UNIX `grep`. These variations range from adding new features, to employing faster algorithms, to changing the behaviour of pattern matching and printing. GNU `grep` is a successor of the UNIX `grep`. GNU `grep` is enhanced with faster algorithms and a richer set of features than the original UNIX `grep`. This work further enhances GNU `grep` by incorporating new features from the context `grep` utility and fixing some bugs.

# 1 Introduction

Searching for a pattern in a text file is an important task. All modern text editors provide functionality for searching text for a pattern. However, pattern matching is also useful beyond text editing. Users of UNIX found that they constantly use the `ed` command `g/re/p`, which means globally search for the regular expression and print the matching lines, where `re` is a regular expression, that a separate program was developed with the same name—`grep`. In effect, `grep` became an acronym for Global Regular Expression Print [Goe95].

It is believed that `grep` is the single widely used tool in program comprehension. Empirical studies show that software engineers use `grep` extensively in their daily maintenance tasks [SL97]. Though useful and powerful, the original UNIX `grep` [IEE03b] was not sufficient to satisfy all needs. On occasions, users found features and optimizations that could be incorporated into `grep`, such as the GNU extensions found in GNU `grep` [Fre02], and on others, they found new approaches to pattern matching that fundamentally changed the behaviour of `grep`, such as context grep [CC96], structured grep [JK96], nondeterministic reverse grep [Nav01], and approximate grep [WM92].

The latest release of GNU `grep`, version 2.5, includes several features that are borrowed from other `grep` variants. The `TODO` file distributed with GNU `grep` lists some future enhancements that would be desirable. In particular, it invites to "take a look at `cgrep` (Context `grep`) seems like nice work," and then asks: "can we merge?" In this project, we take a step toward accomplishing this quest. We selected five features from `cgrep` that are not present in GNU `grep`. These features are:

- *Macros:* allow macro definition and macro substitution in the patterns.
- *Tags:* mark each match with a start and an end tag.
- *Shortest match:* use shortest match rather than the current longest match approach.
- *No end-of-line:* allow the entire file to be treated as a single stream rather than searching one line at a time.
- *Universes:* allow definition of universes (regions) within which the patterns will be searched.

GNU `grep` supports Perl compatible regular expression (PCRE) [Haz04] which include the special operator '?'. The operator '?' modifies the behaviour of the repetition operators such as '*' and '+' to be ungreedy, which produces the shortest match behaviour. GNU `grep` also introduces the option `-z` or `--null-data` that causes the null character '\0' to be used as an end-of-line character instead of the '\n' character. The effect of this option when searching in text files that the whole file is treated as a single line. However, GNU `grep` does not permit the combination of PCRE with the null data option. Therefore, we introduce a new option, `--force-null-data`, that bypasses this restriction and enables GNU `grep` to mimic `cgrep`'s behaviour of searching across line boundaries using the shortest match.

The rest of the paper is organized as follows. Section 2 surveys all the major general-purpose `grep` variants that are freely available. Section 3 presents a brief overview of the new feathers that we introduced to GNU `grep` and give usage examples for each one. In section 4 we list a number of bugs that we found in GNU `grep`. Our impressions on the project and working with GNU `grep` source-code are given in section 5. Section 6 concludes the paper with a summary of our contributions.

# 2 Related Work

Thompson was the first to implement `grep`—pattern matching based on regular expressions [Tho68]. His implementation was based on building a non-deterministic finite automaton (NFA) for the regular expression. Advances in pattern matching algorithms lead to the UNIX `grep` family [IEE03b] consisting of `grep`, `fgrep`, and `egrep`. UNIX `grep`searches text files for a pattern and prints all lines that contain that pattern. It uses a compact non-deterministic algorithm. By default, the pattern is treated as a POSIX basic regular expression [IEE03a]. Fixed-string `grep` (`fgrep`) is optimized for matching strings, and is equivalent to '`grep -F`'. Extended `grep`(`egrep`) supports full regular expressions and includes operators such as the disjunction operator '|' that acts as *or*, and is equivalent to '`grep -E`'.

The GNU `grep` utility [Fre02] provides two advantages over the UNIX `grep`. Firstly, it is based on a fast lazy-state deterministic matcher hybridized with a Boyer-Moore-Gosper search for a fixed string that eliminates impossible text from being considered by the full regular expression matcher without having to look at every character. The result is usually many times faster

than the UNIX `grep` and `egrep`. Secondly, GNU `grep` includes an extended set of options such as printing lines preceding and following the matching line, printing only the matching text, colouring the match, recursively traversing directories, and using Perl compatible regular expressions.

Approximate `grep` (`agrep`) [WM92] is a fuzzy string searching tool. It was developed as the search engine for Global Implicit Search (GLIMPSE) tool for searching and indexing whole file systems, which is part of the HARVEST Information Discovery and Access System. `agrep` has three new features. Firstly, it allows for approximate string matching where the number of mismatched characters can be specified. Secondly, `agrep` is record oriented rather than line oriented. The user can define records by specifying a pattern that marks the beginning of a new record. Records can overlap and nest. Thirdly, `agrep` allows multiple pattern search using the logical AND and OR operators.

Context `grep` (`cgrep`) [CC96] is a part of the MultiText information retrieval system and was developed to search structured texts where the line boundary does not present useful partitioning of the text. `cgrep` treats the new line character as an ordinary character, permitting the search pattern to span multiple lines, as well as permitting multiple matches within the same line. `cgrep` prints substrings that match the pattern. The matches may overlap, but may not nest. Clarke and Cormack implemented the shortest-match principle in `cgrep` and they argue, with supportive examples, that this principle is more useful than the longest-match principle used in `grep` when treating text as a continuous stream [CC97]. `cgrep` supports union and intersection of matches, creating macros, and defining universes over the input to search within.

Structured `grep` (`sgrep`) [JK96] was developed to search highly structured files, such as source-code, mail folders, news folders, HTML and SGML files, and so on. The file is divided into regions using delimiters, such as SGML tags. Regions can overlap and nest. `sgrep` does not use regular expressions; instead, it introduces its own language for indexing and querying. This language, while powerful and expressive, is difficult to learn and to use. Macro definition can simplify some of the common tasks, but only to a limited extent. The power of the `sgrep` query language is most evident when making complex queries on SGML like tagged documents.

Nondeterministic reverse `grep` (`nrgrep`) [Nav01] is the newest addition to the `grep` tools. It was developed to facilitate searching natural language text. At the high level, `nrgrep` provides similar functionality as `agrep`. However,

its implementation is completely different. Its use of a single and uniform algorithmic concept, as opposed multiple specialized algorithms as in `agrep` and GNU `grep`, allows it to perform simple, sophisticated, and approximate pattern matching with an efficiency that degrades smoothly as the complexity of the pattern increases. `nrgrep` also incorporates some of the useful options found in GNU `grep` such as printing context lines and colouring matches.

Perl compatible regular expressions (PCRE) [Haz04] are the most expressive of the `grep` regular expression implementations. In addition to the standard regular expression features, they include operators for shortest matches, define an extended set of character classes, introduce new positional matchers (e.g., end of word), allow references for subpattern matches, and can match across new line characters.

# 3 Enhancements

Four new features are added to GNU `grep`. These features are macros, tags, force null data, and universes. Each one of the these features is described in the following subsections with examples of usage.

## 3.1 Macros

A macro is a symbol or an identifies that is associated with an expression. Macro processing is a string substitution process where each occurrence of a macro identifier is replaced by the associated expression. Macros help formulate search patterns that involve complex or frequent subexpressions.

The macro feature is introduced by adding to new options to the GNU `grep` command line specification: the short option '`-M FILE`' and the long option '`--macro-def=FILE`', where `FILE` is the name of the file that contains the macro definitions. The syntax of the macro definition file is:

name=pattern

where a macro name must start with an alphabetic character, and may include only alphanumeric characters and the underscore character '_'. Each line may contain only one macro definition. The macro usage in a search pattern has the form:

(@name)

**Limitations** The pattern string is processed only once for macro substitution. Therefore, the macro definitions may not use other macros. Also passing parameters to macros is not supported.

**Examples** In the following examples, the macros are defined in a the file `macro.def`. Four macros are defined as follows:

```
PRINT=pri.*t
MAIN=ma.*n
LO=l.*o
D=aaa.*bd
```

The command

```
grep -M macro.def '(@PRINT)' *
```

uses the macro `PRINT` to extracts all the lines containing the pattern '`pri.*t`' from all the files in the current directory. Similarly, the command

```
grep -M macro.def 'v.*d.*(@MAIN)' test.txt
```

extracts all the lines containing the pattern '`v.*d.*ma.*n`' from the file `test.txt` and the command

```
grep --macro-def macro.def '(@PRINT).*(@LO)' *
```

extracts all the lines containing the pattern '`pri.*t.*.l.*o`'.

Macro processing is implemented independent of the other features of GNU `grep`. The macro substitution is applied to the patterns string after all the command line arguments have been processed and before the pattern is compiled into a DFA. As a result, the macro option can be combined with all the other options of GNU `grep` without conflicts.

## 3.2 Tags

Tags are used to format the output and facilitate identifying the matching strings. Two new options are added to GNU `grep`: '`--start-tag=BEFORE`' and '`--end-tag=AFTER`', where `BEFORE` is the text that will be printed before each matching string and `AFTER` is the text that will be printed after each matching string.

Upon parsing the start and end tag options, backslash character substitution is applied to enable the inclusion of escaped characters in the tag strings. The supported escape sequences are: \\, \a, \b, \f, \n, \r, \t, \', and \".

**Limitations** Tags are implemented in the same fashion as the colour option. The current bugs and limitations in using the colour option are the same for the tags options.

**Examples** The start tag option can be used to add backslashes to special characters. For instance, the command

```
echo "Hello, world!" | grep --start-tag='\\' '!'
```

produces "`Hello, world\!`" where the exclamation mark '!' is preceded by a backslash. More interestingly, the use of both start and end tags can be combined to add opening and closing HTML tags tags around a phrase in an HTML document. The command

```
grep --start-tag='<b>' --end-tag='</b>' 'Canada' index.html
```

adds `<b>...</b>` tags around each occurrence of the word "Canada" when matching lines are printed form the file `index.html`. This particular usage is useful if the output of GNU `grep` is to be viewed in a web browser.

## 3.3 Force Null Data (No End-of-Line)

GNU `grep` forbids the combination of the null data option ('`-z`' or '`--null-data`') with the Perl regular expressions option ('`-P`' or '`--perl-regexp`'). The reason for this restriction is that the PCRE library does not handle properly null characters in the search pattern, and thus may produce results that are inconsistent with the other pattern matchers in GNU `grep`.

The null data option is implemented by setting the end-of-line byte within the matcher to the null character rather than to the new line character. The original purpose of introducing the null data option is to process strings in a binary file since these strings are normally terminated with a null character. However, if the input file is a text file, then it should not contain any null characters, and setting this option results in treating the entire file as a continuous stream. Combined with the power of PCRE, namely the '?' operator that modifies the behaviour of the repetition operators such as '*' and '+' to favour shortest-matches rather than longest-matches, this option becomes invaluable when searching across multiple lines of text.

We introduce a new option to GNU `grep`, '`--force-null-data`', that bypasses the restriction mentioned above and enables the user to use the

null data option with PCRE. By default, the dot operator in PCRE '.' does not match a new line. Since the main purpose of forcing null data is to enable searching across line boundaries, the `PCRE_DOTALL` option is passed to the PCRE matcher, which causes the dot operator to match new line characters as well.

**Examples** We use the the option '`-o`' that instructs GNU `grep` to print only the matching string. Without using this option, the entire file would be printed when using the force null data option if a match is found.

Let the file `hello.c` contain the following:

```
#include <stdio.h>
grep -P --force-null -o '^main.*?(.*?).*?{.*}' hello.c

int
main (int argc, char ** argv)
{
        char * text;
        text = "Hello, world!"
        printf("text=%s\n", text);
        return 0;
}
```

The command

```
grep -P --force-null -o '^main.*?{.*?^}' hello.c
```

prints out the entire main function. The use of shortest-match operator ensures that the matching stops when encountering the first closing parenthesis as opposed to the last one. Using a similar approach, the command

```
grep -P --force-null -o '<title>.*?</title>' *.html
```

extracts all the `<title>...</title>` tags from all the HTML documents, even if the tags span more than one line.

## 3.4 Universes

We introduce a new option to GNU `grep`, '`--universe=PATTERN`', where '`PATTERN`' is an expression that defines a universe (region) within which to search for the matches.

Universes are implemented in `cgrep` by constructing two NFAs, one for the universe and one for the pattern, and executing them in parallel on the input symbols. After each input symbol is scanned, the two NFAs are compared to ensure that the matching boundaries of the pattern lie within the matching boundaries of the universe.

GNU `grep` has several wrapper functions for compiling and executing the pattern-matching based on the matcher specified by the user. These wrappers handle setting the environment based on the options given by the user. Therefore, implementing universes in a manner similar to `cgrep`'s implementation would require a complete rewrite of a big portion of the GNU `grep` source-code.

The alternative approach that we follow is to first search for the string that matches the universe, and then search within that string for a match to the patterns. If a match is found, the entire universe is printed. The the loop controlling pattern matching within universes follows the implementation of the only matching option ('`-o`' or '`--only-matching`').

GNU `grep`implementation uses global data structures to retain the compiled expression across the execution calls to the matchers. A second set of these global data is declared. One of the sets stores information pertaining to matching universes while the other set stores information pertaining to matching patterns. The main steps in the algorithm for matching universes and patterns are:

1. Obtain a list of universe expressions (universes) from the command line arguments
2. Obtain a list of pattern expressions (patterns) from the command line arguments
3. If only universes or only patterns are specified, process these expressions as GNU `grep`version 2.5 does.
4. Compile the patterns
5. Save patterns global data
6. Compile the universes
7. Execute the matcher to find a string matching the universes
8. if a match is found

(a) Save the universes global data and restore the pattern globals data

(b) Execute the matcher to find a substring matching the patterns in the universes' matched string

(c) if a match is found then print the entire universes' string while applying tags and colours to the patterns' substrings.

9. Save the patterns global data and restore the universes global data

10. Go to step 7

Multiple universes can be specified using multiple '`--universe`' options similar to using multiple '`-e`' options to define multiple patterns.

**Limitations** The main loop that prints matches within universes is based on the loop for printing matches when the only match option is used. Therefore, the limitations of the only matches option also apply to the universe option including the relevant bugs is section 4.

**Examples** If the file canada.txt contained the line

```
'<b>Canada</b> <i>Canada</i> <b>USA</b>'
```

then the command

```
grep -P --universe '<b>.*?</b>' 'Canada'
```

searches for universes that are marked by the `<b>...</b>` tags and that contain the word 'Canada'. Thus, the output of this command is '`<b>Canada</b>`'.

# 4 Bugs

During our implementation of the features described in section 3 we came across a number of bug in GNU `grep`. These bugs are discussed in the following subsection.

## 4.1 Multiple Patterns with `--colour`

When multiple patters are used with the colour option, if later pattern in the patterns list appears in the text before an earlier pattern, then the earlier pattern is not coloured.

**Example** The command

```
echo 'a b c' | grep --colour -e 'b' -e 'a'
```

fails to colour the letter 'a' and colour only 'b'.

## 4.2 Multiple Patterns with `--only-matching`

When multiple patters are used with the only matching option, if later pattern in the patterns list appears in the text before an earlier pattern, then the earlier pattern is not printed.

**Example** The command

```
echo 'a b c' | grep --only-matching -e 'b' -e 'a'
```

fails to print the letter 'a' and prints only 'b'.

## 4.3 `--only-matching --ignore-case`

When the only matching option is used, the ignore case option does not function properly and in most situation has no effect.

## 4.4 `--colour --ignore-case`

When the ignore case option is used with the colour option, matches are not coloured if the pattern contains an upper case letter.

## 4.5 `--colour --ignore-case --line-buffered`

The line buffered option cause the output to be flushed after each line is printed. This option was not implemented when the colour and ignore case options are combined. We fixed this bug by flushing the output for the combination of colour and ignore case options.

### 4.6 `--perl-regexp --line-regexp --word-regexp`

If a single word appeared on a line with leading whitespace but no trailing whitespace, then it is matched when using Perl regular expressions with the match word and match line options even though it should not be matched (because it violates the match line option). We fixed this bug by making the match line option override the match word option when processing Perl regular expressions, similar to the way this combination of options is handled for non-Perl expressions.

## 5 Impressions

In order to add new command line options, we familiarized ourselves with the GNU Lib C [Fre01] method of defining short (single letter) and long options, and how to parse them and retrieve their arguments. We consulted many of the source-code files of GNU `grep` during the implementation of our enhancements. However, the changes where done only to `grep.h`, `grep.c`, and `search.c` files, which originally contained 42, 1736, and 724 lines of code, respectively.

GNU `grep` consists of two modules: the regular expression library and a wrapper. GNU `grep` source-code relies heavily on the use of global data to pass information within each module, which complicated the implementation of the universes feature.

It is evident that GNU's extensions to `grep` have not been tested thoroughly. The numerous bugs that we discovered all deal with GNU's extensions. Each option seems to work on its own. However, option combinations are implemented in an ad-hoc way creating a lot of unhandled special cases.

An efficient implementation of universes requires executing several DFAs in parallel. The current implementation permits finding unions of matches. Finding intersections of matchings, or applying the logical AND operator, which what universes attempt to simulate, requires a complete rewrite of the wrapper module, and possibly numerous modifications in the library module.

Most of the code is either self-documenting or includes brief comments on the high-level operation that is being performed, though few procedures had their purpose stated even if they performed a substantial amount of computation. In the two files that we worked with, the biggest stones in the way of comprehending the code were the use of global data and lack of

description for procedures.

It was interesting to discover that even popular open-source software can be poorly written in terms of modularization, and can contain numerous bugs that are due to a poor architectural design.

# 6 Conclusion

In an effort to enhance a successful tool—GNU `grep`—we incorporated several new features borrowed from the `grep`-variant `cgrep`. These features are macros, tags, force null data, and universes. The macros facilitate storing frequent expressions and simplifying complex ones. The tags are useful to format the matches and improve their presentation. The force null data option enable combining the use of Perl regular expressions with the null data option, resulting in capabilities similar to `cgrep`'s multi-line search with shortest-match scheme. The universes define regions within which to search for matches of patterns. In addition to introducing new futures, we identified several bugs in the current features and fixed two of them. Our hope is that our contributions will enhance the usefulness of GNU `grep`.

# References


[CC96] Charles L. A. Clarke and Gordon V. Cormack. Context `grep`. Technical Report Technical Report CS-96-41, School of Computer Science, University of Waterloo, Ontario, Canada, 1996.

[CC97] Charles L. A. Clarke and Gordon V. Cormack. On the use of regular expressions for searching text. *ACM Transactions on Programming Languages and Systems*, 19(3):413–426, 1997.

[Fre01] Free Software Foundation, Inc. *The GNU C Library Reference Manual*, 0.10 edition, 2001. For version 2.3.x of the GNU C Library.

[Fre02] Free Software Foundation, Inc. *GNU* `grep` *Documentation*, 2002.

[Goe95] Eric Goedelbecker. Using `grep`. *Linux Journal*, 1995.

[Haz04] Philip Hazel. *PCRE - Perl Compatible Regular Expressions*, 2004.

[IEE03a] The IEEE and The Open Group. *Regular Expressions. The Open Group Base Specifications Issue 6*, IEEE Std 1003.1, 2003 edition, 2003.

[IEE03b] The IEEE and The Open Group. *Utilities:* `grep`. *The Open Group Base Specifications Issue 6*, IEEE Std 1003.1, 2003 edition, 2003.

[JK96] Jani Jaakkola and Pekka Kilpelinen. Using `sgrep` for querying structured text files. Technical Report Technical Report C-1996-83, Department of Computer Science, University of Helsinki, Helsinki, Finland, 1996.

[Nav01] G. Navarro. Nr-grep: a fast and flexible pattern-matching tool. *Software - Practice and Experience*, 31(13):1265–1312, 2001.

[SL97] Janice Singer and Timothy C. Lethbridge. What's so great about '`grep`'? implications for program comprehension tools. WWW: `http://wwwsel.iit.nrc.ca/~singer/grep/greptxt.html`, 1997.

[Tho68] Ken Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[WM92] S. Wu and U. Manber. AGREP—a fast approximate pattern-matching tool. In *The USENIX Winter 1992 Technical Conference*, pages 153–162, 1992.