

# 'Filter': A Grep-like Search Utility

Joel Polowin and Mark Wroth

April 17, 2000

# Chapter 1

## User Documentation

### 1.1 Introduction

This is `filter` revision 2.0 dated 28 February 1994, a `grep`-like text searcher for multiple simultaneous keyword tests.

Copyright © 1994 by Joel Polowin, Department of Chemistry, Queen's University, Kingston, Ontario, Canada. Permission granted for free use and distribution; I want credit/blame for writing it.

E-mail: `polowin@silicon.chem.queensu.ca`,  
`polowinj@qucdn.queensu.ca`,  
`Joel.Polowin@p4.f107.n249.z1.fidonet.org`.

The Nuweb version of this file should be blamed on Mark Wroth, `mark@astrid.upland.ca.us`. Joel was foolish enough to allow me to port his program to Nuweb on the Amiga, a permission which I have extended to a Windows NT version. Kudos for the program goes to him; blame for the port belongs to me.

This is Revision 1.4, dated April 18, 2000 of the `literate` programming version.

Literate programming documentation copyright © 1995, 2000 by Mark Wroth. Permission is granted for use and distribution under the terms set above by the code author.

If you see something wrong with it or it fails to work, *please* let one of us know!

### 1.2 Syntax

`filter` [`filename`] [`filename ...`] `string` [`string ...`]

where each `string` (default max. of 2000) is a term to be searched for in lines (default max. 600 chars) in file(s) `filename`, prefixed by one of the following characters:

- + to show lines which contain string
- to show lines which do not contain string
- = to show lines which contain string, case sensitive
- \_ (underscore) to show lines which do not contain string, case sensitive

A string as above may be further prefixed with the letter `o` to print the line if the current OR the preceding condition is true.

A string including blanks and the prefix may be enclosed in double quotes on most systems. Your operating system may have other ways of dealing with special characters.

`filter` determines the first string which is a search term instead of a file name by its beginning with one of the characters `+ -= _`. For files whose names begin with one of these characters, see below. Otherwise, the first search term must begin with one of these, as that first term cannot be or-linked to a preceding term.

For strings that begin with `"$"` or `"&"` (usually names of files of search terms), see below.

### 1.2.1 Examples

```
filter * +hawk +handsaw o+hound
```

searches all files in the current directory and prints lines that contain the string `"hawk"` and at least one of `"handsaw"` and `"hound"`. This assumes that the operating system and compiler accept wild-carded file names; else `filter` will be looking for a file named `"*"`. For DOS, one would use `*.*`.

```
filter armorial =Vert +argent -gules _Or -azure -purpur +foil > tempfile.txt
```

searches the file `"armorial"` for lines that contain the string `"Vert"` (case-sensitive) and `'argent'` (upper or lower case) but not `'gules'` (upper or lower case) and not `'Or'` (case-sensitive) and not `'azure'` or `'purpur'` (upper or lower case) and DO contain `'foil'` (upper or lower case); the resulting lines are saved in file `"tempfile.txt"`.

```
type temp1.txt | filter +aardvark "o+winged pig" o+wombat "|B|"
```

The file `"temp1.txt"` is fed through the `filter` program, which passes lines that contain `"aardvark"` (upper or lower case) or the string `"winged pig"` or `"wombat"` (upper or lower case) and do *not* contain `"|B|"`. The result is printed on the screen. Note use of quotation marks in the command line to include the space in `"winged pig"` and the special character `|` in `"|B|"`.

### 1.2.2 File names beginning with one of "+ -= \_"

If you absolutely *must* use text file names that begin with one of these characters, use the character twice when specifying the file name to `filter`. Thus, the file name `-stdev.c` would be written `--stdev.c`; `++junk.c` would be written `+++junk.c`.

What `filter` does is to go through each term in the command line and count the number of identical flag characters beginning each; that number is reduced by half, rounded down. An even number specifies a text file name; an odd number designates a search term. `+junk` has one flag character, is not changed (shortened by  $1/2 \rightarrow 0$  characters), and is a search term: print lines containing `junk`. `++junk` has two flag characters, is shortened to `+junk`, and is read as a text file name. `+++junk` is shortened to `++junk`, and is a search term: print lines containing the string `+junk`. `+=junk` has one flag character and is a search term: print lines containing the string `=junk`.

This means that wild-carded file names that match files whose names begin with one of `"+-=_"` will cause trouble. I'm sorry; the telepathic monitors of most computer systems are not software-addressable. A compulsive urge to use files whose names begin with punctuation or mathematical symbols can now be treated successfully in a majority of cases.

Search terms specified in files (see below) are not themselves in the command line, and if they begin with one of `"+-=_"` those characters should not be doubled. Search-term file expansion takes place after `filter` determines which command-line strings are file names.

### 1.2.3 '\$' and '&' usually flag search-term file names

A string which begins with `"$"` or `"&"` will be expanded as the name of a file containing a list of search terms. For example, the string `+$critters` tells `filter` to look for a file `critters`; lines from that file are taken as search terms. `filter` adds the prefix to each term, depending on whether the file name is specified with `"$"` or `"&"` and which of `"+-=_"` precedes it. With `"$"`, `+` and `=` give or-linked terms, so that text file lines will be printed if any search-term file line is matched; `-` and `_` are not or-linked, so that text file lines are printed only if no search-term file line is matched. With `"&"`, `+` and `=` are not or-linked, so that *all* search-term lines must be matched to print a text line; while `-` and `_` are or-linked, so that any search-term line *not* matched will allow text-line printing.

To search for actual text strings beginning with `'$'` or `'&'`, double the flag characters. Thus, to search for the string `"$100"`, use the search string `+$100`. To expand file names beginning with those characters, use three of them: search-term file `"&&junk"` would be specified with something like `-&&&junk`. In general, when a search term begins with a flag character, double each flag character of that kind beginning the term, and if the term is a file name, add an extra flag character.

Search-term files may contain file names, which will be expanded in turn. For this reason, initial `'$'` and `'&'` characters must be doubled even in nested search-term files.

Note that the or-linking logic can get seriously messed up when terms beginning with `'-'` or `'_'` are expanded carelessly, as `filter` has no good sense of logical precedence. If file `"human"` contains `"man"` and `"woman"`, then `"o-$human"` would expand as `"o-man -woman"`.

Examples:

```
filter armorial +$animal o+$vegetable _$mineral
```

If file animalreads

```
\$human
```

```
reptile
```

```
amphibian
```

and file human reads

```
man
```

```
woman
```

and file vegetable reads

```
tree
```

```
grain
```

and file mineral reads

```
rock
```

```
dirt
```

then the above will be expanded to:

```
filter armorial +man o+woman o+reptile o+amphibian o+tree o+grain
```

```
_rock _dirt
```

```
filter armorial +$&beastie +&&&doggie -$dragon
```

If file &beastie reads

```
unicorn
```

```
\$dragon
```

```
manticore
```

and file &doggie reads

```
terrier
```

```
hound
```

```
\$\$paniel
```

and file dragon reads

```
wyvern
```

```
dragon
```

```
lizard
```

then the above will be expanded to

```
filter armorial +unicorn o+wyvern o+dragon o+lizard o+manticore  
+terrier +hound +$$paniel -wyvern -dragon -lizard
```

which will print lines from file "armorial" that contain: any of: "unicorn", "wyvern", "dragon", "lizard", "manticore"; and ALL of: "terrier", "hound", "\$paniel"; and none of: "wyvern", "dragon", "lizard".

## 1.3 Revision History

### 1.3.1 Code

Version 1.0 September 1992.

1.1 Sep '92 fixed minor bugs

1.2 Sep '92 added 'or'-linking to keywords

1.4 Oct '92 fixed a minor error in string lengths, added size DEFINES

1.5 Jan '94 increased string lengths, fixed a Stupid Newbie Error re: assumption that `*argv[]` was writable

2.0 Feb '94 added search-term file expansion and multiple text file capability, including wildcards when system permits

### 1.3.2 Documentation

*Log : filter.w, v* Revision 1.4 2000/04/18 04:40:21 Mark Basic documentation complete.

Revision 1.3 2000/04/18 04:21:48 Mark Document compiles.

# Chapter 2

# Implementation

## 2.1 The Program

"filter.c" 6 ≡

```
#define LENGTH 1201 /* 1 more than max \# characters */
#define ARGS 2000

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

<Print the syntax message 8a>
<Lowercase a string 7>

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

<Variables and types of the main function 8b>

<Parse the input line 9>

    k=0;
    do
    {
        k++;
        if(firststring==1) infile=stdin;
        else
            if(!(infile=fopen(argv[k],"r")))
            {
                fprintf(stderr,"Can't open file %s for reading.\n",argv[k]);
```

```

        syntax();
    }
    l=0;

    for (;;)
    {
        if(NULL==fgets(line,LENGTH,infile)) break;
        if(LENGTH==strlen(line)+1) fprintf(stderr,"* Warning: truncated line \n%s\n",line);
        if(lowcase) strlow(strcpy(lowline,line));
        for(i=1; i<=nostring; i++)
        {
            test=0;
            switch(*myargv[i])
            {
                case '=':
                case '_':
                    if(NULL!=strstr(line,(myargv[i]+1))) test=1;
                    break;
                default:
                    if(NULL!=strstr(lowline,(myargv[i]+1))) test=1;
                    break;
            }
            if(test!=flag[i] && orflag[i]==0) break;
            if(test==flag[i])
                while(orflag[i]==1) i++;
        }
        if(i>nostring)
        {
            if(firststring>2 && l==0)
            {
                printf("File %s:\n",argv[k]);
                l=1;
            }
            printf("%s",line);
        }
        if(infile!=stdin) fclose(infile);
    }
    while(k<firststring-1);
    exit(0);
}

```

⟨Lowercase a string 7⟩ ≡

```

void strlow(string)
char *string;
{
    while (*string!='\0')
    {
        *string=tolower(*string);
    }
}

```

```

        string++;
    }
}

```

Macro referenced in scrap 6.

⟨Print the syntax message 8a⟩ ≡

```

void syntax ()
{
    fprintf(stderr,"Syntax: filter [filename] [filename ...] string [string ...]\n where each");
    fprintf(stderr," string (max. of %d) is a term to be searched for",ARGS);
    fprintf(stderr," in lines (max.\n %d chars) in file",LENGTH-1);
    fprintf(stderr," 'filename', prefixed by one of the following characters:\n");
    fprintf(stderr," + to show lines which contain string\n");
    fprintf(stderr," - to show lines which do not contain string\n");
    fprintf(stderr," = to show lines which contain string, case sensitive\n");
    fprintf(stderr," _ (underscore) to show lines which do not contain string,\n");
    fprintf(stderr,"         case sensitive\n\n");
    fprintf(stderr,"A string as above may be further prefixed with the letter 'o' to\n");
    fprintf(stderr," print the line if the current OR the preceding condition is true.");
    fprintf(stderr,"\nA string including blanks and the prefix may be enclosed in");
    fprintf(stderr," double quotes.\nStrings beginning with '$' or '&' designate");
    fprintf(stderr," file expansion; see filter20.doc.\n\nExamples:\n ");
    fprintf(stderr," filter armorial =Vert +argent ");
    fprintf(stderr,"-gules _Or -azure -purpur +foil > tempfile.txt\n");
    fprintf(stderr," type temp1.txt | filter +aardvark \042o+winged pig\042 ");
    fprintf(stderr,"o+wombat \042_|B|\042\n\nFilter utility v.2.0 ");
    fprintf(stderr,"(C) 1994 by Joel Polowin, Chem. ");
    fprintf(stderr,"Dept., Queen's University,\nKingston. ");
    fprintf(stderr,"Permission granted for free ");
    fprintf(stderr,"use; I want credit/blame for writing it.\n");
    fprintf(stderr,"polowin@silicon.chem.queensu.ca, polowinj@qucdn.queensu.ca\n");
    exit(1);
}

```

Macro referenced in scrap 6.

We should accumulate these...

⟨Variables and types of the main function 8b⟩ ≡

```

char line[LENGTH],lowline[LENGTH];
FILE *infile;
int i,j,k,l,test,firststring,nostring,lowcase;
void syntax();
void strlow();
char flag[ARGS+1],orflag[ARGS+1];
char *myargv[ARGS+1];
int blocksize;
char *prefix,*filename;

```

Macro referenced in scrap 6.

## 2.2 Parsing the input line

At least I *think* I got the input parser ... beware of improper boudaries.

⟨Parse the input line 9) ≡

```
firststring=0;
for(i=1; i<argc; i++)
{
    if(*argv[i]=='=' || *argv[i]=='+' || *argv[i]=='-' || *argv[i]=='_')
    {
        test=*argv[i];
        for(j=1; test==*(argv[i]+j); j++)
            ;
        if(j%2 && firststring==0)
            firststring=i;
        argv[i]+=j/2;
    }
}
```

⟨Check for no search string 12a)⟩

⟨Check for too many search strings 12b)⟩

```
lowcase=0;

for (i=0; i<ARGS+1; i++) orflag[i]=0;

blocksize=0;
for (i=firststring; i<argc; i++) blocksize+=strlen(argv[i])+1;
```

⟨Allocate memory for the internal arguement list 13a)⟩

⟨Transfer the external arguements to the internal list 13b)⟩

```
for (i=1; i<=nostring; i++)
{
    j=0;
    switch(*myargv[i])
    {
        case '+':
            strlow(myargv[i]);
            lowcase=1;
        case '=':
            flag[i]=1;
            break;
        case '-':
```

```

        strlow(myargv[i]);
        lowercase=1;
    case '_':
        flag[i]=0;
        break;
    case '0':
    case 'o':
        orflag[i-1]=1;
        myargv[i]++;
        i--;
        j=1;
        break;
    default:
        fprintf(stderr,"Error in string no. %d: %s\n",i-1,myargv[i]);
        syntax();
}

if((!j) && (((test==(myargv[i]+1))=='$') || (test=='&')))
{
    for(j=1; test==(myargv[i]+1+j); j++)
        ; /* count identical flag chars */

    l=j/2;

    for (k=0; *(myargv[i]+k)!='\0'; k++) /* shift string to delete */
        *(myargv[i]+k+1)=(myargv[i]+k+1+1); /* half of flag chars */

    if(j%2) /* an odd number of flag chars: expand file */
    {
        switch(*myargv[i]) /* determine prefix for expanded terms */
        {
            /* from current prefix and expansion type */
            case '+':
                if(test=='$') prefix="o+";
                else prefix="+";
                break;
            case '=':
                if(test=='$') prefix="o=";
                else prefix="=";
                break;
            case '-':
                if(test=='$') prefix="-";
                else prefix="o-";
                break;
            case '_':
                if(test=='$') prefix="_";
                else prefix="o_";
                break;
            default:
                fprintf(stderr,"Bugger-up in program!\n");
                exit(1);
        }
    }
}

```

```

    }

filename=myargv[i]+2;

if(!(infile=fopen(filename,"r")))
{
    fprintf(stderr,"Can't open search-term file %s\n",filename);
    exit(1);
}

test=*myargv[i]; /* prefix of current term */

blocksize=0; /* figure out how much memory to allocate */
for(j=0;;j++) /* for new terms */
{
    if(NULL==fgets(line,LENGTH,infile)) break;
    if(LENGTH==strlen(line)+1) fprintf(stderr,
        "* Warning: truncated search term file %s line\n%s\n",
        filename,line);
    if(line[strlen(line)-1]=='\n') line[strlen(line)-1]='\0';
    blocksize+=strlen(line)+1;
}

if (j==0)
{
    fprintf(stderr,"* Warning: empty search term file %s\n",
        filename);
    j=1;
}

blocksize+=j*strlen(prefix);
fclose(infile);

if(nostring+j-1>ARGS)
{
    fprintf(stderr,"File expansion gives too many terms (%d max)\n",
        ARGS);
    exit(1);
}

for(k=nostring; k>i; k--) /* shift old myargv to make room */
    myargv[k+j-1]=myargv[k];

if(NULL==(myargv[i]=malloc(blocksize)))
{
    fprintf(stderr,"Can't allocate memory for search term expansion.\n");
    exit(1);
}

if(!(infile=fopen(filename,"r")))

```

```

        {
        fprintf(stderr,"Can't open file %s for second read.\n",filename);
        exit(1);
        }

for(k=0;k<j;k++)
{
fgets(line,LENGTH,infile);
if(line[strlen(line)-1]!='\n') line[strlen(line)-1]='\0';
if(k==0)
{
*myargv[i]=test;
strcpy(myargv[i]+1,line);
}
else
{
myargv[i+k]=myargv[i+k-1]+strlen(myargv[i+k-1])+1;
strcpy(myargv[i+k],prefix);
strcat(myargv[i+k],line);
}
}

fclose(infile);
nostring+=j-1;
i--;
}
}
}

```

Macro referenced in scrap 6.

Need to have at least one search string. Print a warning if there isn't.

⟨Check for no search string 12a⟩ ≡

```

if(firststring==0)
{
fprintf(stderr,"Must specify a search string.\n");
syntax();
}

```

Macro referenced in scrap 9.

Similarly, need to not have too many.

⟨Check for too many search strings 12b⟩ ≡

```

nostring=argc-firststring;
if(nostring>ARGS)

```

```
{
  fprintf(stderr,"Too many search strings specified.\n");
  syntax();
}
```

Macro referenced in scrap 9.

⟨Allocate memory for the internal argument list 13a⟩ ≡

```
if (NULL==(myargv[1]=malloc(blocksize)))
{
  fprintf(stderr,"Can't allocate memory for string storage.\n");
  exit(1);
}
```

Macro referenced in scrap 9.

⟨Transfer the external arguments to the internal list 13b⟩ ≡

```
strcpy(myargv[1],argv[firststring]);

for(i=2; i<=nostring; i++)
{
  myargv[i]=myargv[i-1]+strlen(myargv[i-1])+1;
  strcpy(myargv[i],argv[i+firststring-1]);
}
```

Macro referenced in scrap 9.

# Appendix A

## Index

This index lists identifiers and other significant items by section of appearance (not by page).

"`filter.c`" Defined by scrap 6.

⟨Allocate memory for the internal argument list 13a⟩ Referenced in scrap 9.

⟨Check for no search string 12a⟩ Referenced in scrap 9.

⟨Check for too many search strings 12b⟩ Referenced in scrap 9.

⟨Lowercase a string 7⟩ Referenced in scrap 6.

⟨Parse the input line 9⟩ Referenced in scrap 6.

⟨Print the syntax message 8a⟩ Referenced in scrap 6.

⟨Transfer the external arguments to the internal list 13b⟩ Referenced in scrap 9.

⟨Variables and types of the main function 8b⟩ Referenced in scrap 6.

`blocksize`: [8b](#), 9, 13a.

`firststring`: 6, [8b](#), 9, 12ab, 13b.

`infile`: 6, [8b](#), 9.

`LENGTH`: 6, 8a, [8b](#), 9.

`line`: 6, 8a, [8b](#), 9.

`lowercase`: 6, [8b](#), 9.

`nostring`: 6, [8b](#), 9, 12b, 13b.

`strlow`: 6, [7](#), 8b, 9.

`syntax`: 6, [8a](#), 8b, 9, 12ab.

# Index

filter, 1-3