Introduction to UnixDos

The "UNIX on DOS Toolkit(UnixDos)" brings the power of UNIX to any computer running the
ANY Windows or MS-DOS operating system (Windows 95/NT/3.X/MS-DOS)!
The UnixDos Toolkit consists of 64 UNIX utilities emulated for the DOS/Windows
systems plus 28 additional utilities which can be very useful.
Please send any questions and suggestion around UnixDos to:
UnixDos@pobox.com

UnixDos License Agreement
Make sure you understand and accept the UnixDos License Agreement.

Overviews
All the 92 UnixDos utilities can be grouped in the following ways:
• Overview by functional Groups
• Overview by Origin (Basic UNIX/Berkeley UNIX and new Utilities)
• Overview by Alphabet

Overview of UnixDos Online Help
This Online Help has the following three major sections:
• General Information (applies to ALL UnixDos utilities)
• Practical Examples (for the impatient Users who want to see some results quick)
• Reference Manual (Each single UnixDos program is explained in detail)

UNIX Emulation under DOS/Windows:
All the 64 UNIX utilities have been as much emulated as possible to the standard UNIX
commands in the UnixDos Toolkit, but some DOS specific adjustments needed to be
made (see Differences between DOS and UNIX).
We also show you all the UnixDos Improvements and Changes to "Classic UNIX".
Some UNIX utilities do not apply to the Windows/DOS environment and have not been
ported (see here)

Argument Expansion:
In these sections we introduce you to:
the standard UNIX/UnixDos Argument Expansion
and the Regular Expression for File Matches
and the Regular Expression for Text Searches
and the added power of the UnixDos Argument Expansion Extensions
and the helpful Argument Expansion Text Module SHOW_EXP

16/32Bit Versions
UnixDos is available for all the current 32bit Windows operation systems
(Windows95/NT).
To make the UnixDos Toolkit also available for as many users as possible we also offer
the 16-bit version which runs on a simple MS-DOS environment (requires no Windows

3.1!) or in a MS-DOS window under Windows 3.11.
UnixDos runs on all the following Windows and MSDOS (or emulated) operating systems:
- MS-DOS 5.00 or Higher until 6.22 (16Bit)
- Windows 3.,1 and 3.11 from MS-DOS Prompt (16Bit)
- Windows 95 from MS-DOS Prompt (32Bit)
- Windows NT 3.5X from CMD Prompt (32Bit)
- Windows NT 4.0 from CMD Prompt (32Bit)

In this manual the "DOS Environment" refers to a command window under Windows 95/NT/3.1 or the actual plain MS-DOS prompt when you exit Windows.

Installation

No special installation process is needed to run any UnixDos utility in your DOS environment; Simply copying all UnixDos utilities into a new directory and updating the PATH is sufficient
(see the 'Installation' section below for details).

We also describe How to get Command Line Help and the UnixDos Environment Variables

About this Online Help

This manual is written for both the novice user and the professional expert.
If you find text is underlined and green you can click on it to get more information.
Suggestions of programs and commands you are typing on the command line are displayed in Bold and in fixed pitch font:

uecho this is a command you type from the keyboard

The output or input files are shown in italic and fixed pitch font:

this is a command you type from the keyboard

Error Message (STDERR) Capture

In UNIX and DOS/Windows there are three basic channels of communication from the command line:
- STDIN: Standard input   Example: wc < in.txt
- STDOUT: Standard output   Example: ls > files.lst
- STDERR: Standard error   Example: cp > nothing will still display the command line help

All error message from any UnixDos utility is written to the error message channel (STDERR).
Example:

cp -l notexist new

Will display the error message on your screen:
notexist   -> new
*   cp: Cannot find 'notexist'*

```
0 files written successfully
```

If you redirect the output still you will see this message because <u>CP</u> writes the error and progress message to the error channel (stderr):

```
cp -l notexist new > cp.txt
```
```
notexist              -> new
    cp: Cannot find 'notexist'
0 files written successfully
```

To redirect the error and diagnostic output use the <u>STDERR</u> utility:

```
stderr cp -l notexist new > cp.txt
```

Now the error messages are written to 'cp.txt'

To see the error output AND write the message to a file use <u>TEE</u>:

```
stderr cp -l notexist new > tee cp.txt
```
```
notexist              -> new
    cp: Cannot find 'notexist'
0 files written successfully
```

History of UnixDos

We worked for many years in the UNIX environment and quickly grew very fond of UNIX and its immense versatility and power. The excellent design idea to allow "linking" of several individual utilities together using the pipe mechanism and processing text streams in a step by step fashion created its immense power. This allows you to solve almost any data manipulation or administrative task and create very powerful new tools if your data is stored as plain text.

Then we entered the MS-DOS world in 1987 and felt suddenly like a fish stranded on the beach: The set of DOS utilities compared to UNIX looked very primitive and limited. (Even now after 10 years this didn't change even when the Windows95 and WindowsNT hit the market. DIR got a few sorting options and a file compare FC was added)

So we started to embark on a lengthy journey and developed a package which makes all the relevant UNIX utilities available under the MS-DOS environment. We also had a list were we felt the UNIX utilities could use some improvement, so while we were at it we implemented those extras and also added some new features
(see "Differences between UnixDos and classic UNIX").

We could have saved ourselves a lot of work by using other already existing UNIX packages on the market (MKS, PICNIX, DOSNIX etc.) but found them very limited in functionality or not so user friendly (see "The Competition" for details). Also since there is no source code they can obviously not be customized!

DOS/UNIX Name Clash

Unfortunately some utilities have the an identical name under DOS and UNIX, but have a completely different functionality.

To have a clear distinction we inserted the 'u' for UnixDos utility name.

Below is a list of those 8 clashes:

UNIX	UnixDos	NAME WITHIN HELP
cd	ucd	(u)cd
date	udate	(u)date
echo	uecho	(u)echo
exit	uexit	(u)exit
find	ufind	(u)find
join	ujoin	(u)join
more	umore	(u)more
sort	usort	(u)sort
time	utime	(u)time

To avoid confusion these utilities will still appear in the documentation at the places without the inserted 'u' (ucd is found under 'c' not 'u').

Installation of UnixDos for Windows 95/NT

You will need 7 Megabytes on your hard disk to install the full UnixDos package. Below you will find the simple steps to install the package:

- Download UnixDos from http://www.ProfSoftware.com
  unixdo32.ZIP for UnixDos for Windows 95/NT

- Unzip the contents to a temporary location:
  ```
  c:
  cd \temp
  pkunzip -o \download\unixdo32.zip
  ```

- Start the SETUP program and follow the directions
  ```
  c:
  cd \temp
  SETUP
  ```

- The SETUP program will prompt you for the destination drive (by default C:) and the destination directory (by default C:\UnixDos).

- Then SETUP reminds you to add the new UnixDos directory to the PATH:
  ```
  edit C:\AUTOEXEC.BAT
  set PATH=C:\DOS;C:\MSVC\BIN;C:\WINDOWS;C:\UNIXDOS
  ```

- To install your License run UD_MGMT (UnixDos Management);
  choose I (Install) and follow the directions:
  ```
  C:
  cd \unixdos
  ud_mgmt
  ```

To move the License to another machine run UD_MGMT (UnixDos Management); choose U (UnInstall) and follow the directions.

Installation for Windows 3.11/MS-DOS

You will need 11 Megabytes on your hard disk to install the full UnixDos package.
Below you will find the simple steps to install the package:
(If you 3.5 disk drive is on B: replace A: with B:)

- Download UnixDos from http://www.ProfSoftware.com
  unixdo16.ZIP for Windows 3.X, MS-DOS

- Create the UnixDos directory:
  ```
  c:
  md \unixdos
  pkunzip \download\unixdo16.zip
  ```

- Add the new UnixDos directory to the PATH in AUTOEXEC.BAT:
  ```
  edit C:\AUTOEXEC.BAT
  set PATH=C:\DOS;C:\MSVC\BIN;C:\WINDOWS;C:\UNIXDOS
  ```

- To install your License run UD_MGMT (UnixDos Management);
  choose I (Install) and follow the directions:
  ```
  C:
  cd \unixdos
  ud_mgmt
  ```

To move the License to another machine run UD_MGMT (UnixDos Management);
choose U (UnInstall) and follow the directions.

Overview by functional Groups

Below is an overview by the following functional groups:
- Move/delete/change files/directories
- Display/search contents of a file
- Manipulate file contents (analyze, convert, count, edit, extract, sort, split)
- List/find files/directories
- Compare/merge files
- User Interaction
- Program Development
- Miscellaneous

Move/delete/change files/directories

(u)cd	Change to specified directory
chmod	Change file mode/attribute
cp	Copy file(s) to new directory/file
cpio	Copy/restore files to/from archive or move to new directory
mkdir	Create new directory(ies)
mv	Move file(s) to new directory/file
mvdir	Rename directory
rm	Remove file(s)/directory(ies)
touch	Change file time

Display/search contents of a file

bgrep	Search in binary file for pattern or value
bvi	Binary Viewer
cat	Display text file contents
egrep	Search text file(s) for one or several pattern (search expression)
fgrep	Search text file(s) for one or several pattern (straight text)
file	Display file type
grep	Search text file for single pattern expression
head	Display beginning of text file
nl	Display line number and text file
(u)more	Display text file one screen at a time and allow browsing
od	Display any file contents as octal, decimal, hex or char values
pr	Print a text file
tail	Display the end of a text file
tee	Display contents of a pipe and store also into file(s)
strings	Display the text sections of any file (usually binary)

Manipulate file contents (analyze, convert, count, edit, extract, sort, split)

af	Analyze file
b64	Decode Base64 Internet encoded files
bed	Binary stream editor
bsplit	Split a binary file into separate file chunks containing specified number of bytes
cut	Extract columns/fields from a file
dd	Convert file (ASCII/EBCDIC/upper/lower-case/swap bytes)
dio	Direct disc input/output
dos2unix	Convert DOS text files to UNIX
getlines	Get range of lines from a file
io	Extract file area and write to another file (at specified offset)
sed	Extract specified lines and/or insert/substitute text
(u)sort	Sort file based on fields, columns or entire line
split	Split a file into separate file chunks containing specified number of lines
sum	Display the checksum of file
tr	Translate a set of characters
unix2dos	Convert UNIX text files to DOS
uuencode	Convert binary file to ASCII codes for transmission
uudecode	Convert ASCII file (from uuencode) back to binary
wc	Count lines/words/characters in file

List/find files/directories

basename	Extract the filename+extension
df	Display free space on disk(s)
dirname	Extract the directory name
drvname	Extract the drive letter
extname	Extract the extension
filename	Extract the filename without extension
du	Display directories with occupied space
ls	List files (and sort by name/size/time/type)
(u)find	Find files based on age/name/size/time/type/filemode
pwd	Display current directory
slash	Convert DOS filename style to/from UNIX
which	Find programs by name or pattern

Compare/merge files

cf	High speed file compare byte by byte
cmp	Compare two text files byte by byte
cmpdir	Compare all files in two directories by time/size/contents
comm	Display common/uncommon lines of two files
diff	Compare two text files line by line
(u)join	Join two sorted text files based on common field
paste	Join several text files
uniq	Find unique lines/fields in sorted text file
test	Test any condition (numeric/text/file related)

User Interaction

line	Read one line from the keyboard
inpbox	Display message and prompt for input
msgbox	Display message box
txtbox	Advanced interactive text viewer
usrchar	Prompt user for single character
usrprmpt	Prompt user for text

Program Development

<u>cc</u>	UnixDos C Compiler front end
<u>cc1</u>	UnixDos C compiler pre processor
<u>cc3</u>	UnixDos C compiler post processor
<u>m4</u>	Advanced macro processor for text files with own language
<u>mak2mk</u>	Convert Visual C Makefile to UnixDos makefile

Miscellaneous Functions

add	Add Numbers
banner	Print text in big letters horizontally
bbanner	Print text in very big letters horizontally vertically
bell	Ring bell
cal	Display calendar (month or entire year)
clear	Clear the screen
conv	Convert numbers to/from any base
(u)date	Display/set the current date and time
(u)echo	Display text, arguments or environment variables
env	Display/set environment (and run program with new environment)
expr	Calculate the specified expression
kill	Terminate/Kill processes
mf	Show free memory
ps	Show current processes
sleep	Wait for specified number of seconds
(u)time	Run a program and display elasped time
uptime	Display elapsed time since last reboot
saveexit	Save last errorlevel value
setenv	Set environment variable(s)
show_exp	Show UnixDos argument expansion
stderr	Redirect output from STDERR to STDOUT
wait_end	Wait for process completion

Overview by Origin

Below is an overview by the following functional groups:
- <u>Basic or Core UNIX utilities</u>
- <u>Berkeley UNIX</u>
- <u>New and additional UnixDos utilities</u>

Basic or Core UNIX utilities

The UnixDos utility package contains the following 55 essential UNIX commands:
(see Berkeley UNIX for more UNIX utilities)

banner	Print text in big letters horizontally
basename	Extract the filename+extension
cal	Display calendar (month or entire year)
cat	Concatinate several files
cc	UnixDos C Compiler front end
(u)cd	Change to specified directory and drive!
chmod	Change file mode/attribute
cmp	Compare two files byte by byte
comm	Display common/uncommon lines of two files
cp	Copy file(s) to new directory/file
cpio	Copy/restore files to/from archive or move to new directory
cut	Extract columns/fields from a text file
(u)date	Display/set the current date and time
dirname	Extract the directory name
dd	Convert file (ASCII/EBCDIC/upper/lower-case/swap bytes)
df	Display free space on disk(s)
diff	Compare two text files line by line
du	Display directories with occupied space
(u)echo	Display text, arguments or environment variables
egrep	Search text file(s) for one or several pattern (search expression)
env	Display/set environment (and run program with new environment)
expr	Calculate the specified expression
fgrep	Search text file(s) for one or several pattern (straight text)
file	Display file type
(u)find	Find files based on name/size/time/type/filemode
grep	Search text file for single pattern expression
head	Display beginning of text file
(u)join	Join two sorted text files based on common field
kill	Terminate/Kill processes
line	Read one line from the keyboard
ls	List files (and sort by name/size/time/type)
m4	Advanced macro processor for text files with own language
mkdir	Create new directory(ies)
mv	Move file(s) to new directory/file
mvdir	Rename directory
nl	Display line number and text file
od	Display any file contents as octal, decimal, hex or char values
paste	Join several text files
pr	Print a text file
ps	Show current processes
pwd	Display current drive and directory

rm	Remove file(s)/directory(ies)
sed	Extract specified lines and/or insert/substitute text
sleep	Wait for specified number of seconds
(u)sort	Sort file based on fields, columns or entire line
split	Split a file into separate file chunks containing specified number of lines
sum	Display the checksum of file
tail	Display the end of a text file
tee	Display contents of a pipe and store also into file(s)
test	Test any condition (numeric/text/file related)
(u)time	Run a program and display elasped time
touch	Change file time
tr	Translate a set of characters
uniq	Find unique lines/fields in sorted text file
wc	Count lines/words/characters in file

Berkeley UNIX utilities

UnixDos also contains the 9 following Berkeley UNIX extensions which have been originally developed in the University of Berkeley, California:

<u>bbanner</u>	Print text in very big letters horizontally vertically
<u>clear</u>	Clear the screen
<u>(u)more</u>	Display text file one screen at a time and allow browsing
<u>setenv</u>	Set environment variable(s) (in C-Shell: csh)
<u>strings</u>	Display the text sections of any file (usually binary)
<u>uptime</u>	Display elapsed time since last reboot
<u>uuencode</u>	Convert binary file to ASCII codes for transmission
<u>uudecode</u>	Convert ASCII file (from uuencode) back to binary
<u>which</u>	Find programs by name

Additional new UnixDos utilities

UnixDos contains also the following 20 new additional utilities:

add	Add Numbers
af	Analyze file
b64	Decode Base64 Internet encoded files
bed	Binary stream editor
bell	Ring bell
bgrep	Search pattern in binary file
bvi	Binary file Viewer
cc1	UnixDos C compiler pre processor
cc3	UnixDos C compiler post processor
conv	Convert numbers to/from any base
cf	High speed file compare of two binary files byte by byte
cmpdir	Compare all files in two directories by time/size/contents
dio	Direct disc input/output
dos2unix	Convert DOS text files to UNIX
drvname	Extract the only drive letter in a path name
extname	Extract the file extension
filename	Extract the filename without extension
getlines	Get range of lines from a file
inpbox	Display message and prompt for input
io	Extract specified area within a file and write to another file (at specified offset)
mak2mk	Convert Visual C Makefile to UnixDos makefile
mf	Show free memory
msgbox	Display message box
saveexit	Save last errorlevel value
show_exp	Show UnixDos argument expansion
slash	Convert filenames to/from UNIX/DOS style
stderr	Redirect output from STDERR to STDOUT
txtbox	Advanced interactive text viewer
unix2dos	Convert UNIX text files to DOS
wait_end	Wait for process completion

Overview by Alphabet

add	Add Numbers
af	Analyze file
b64	Decode Base64 Internet encoded files
banner	Print text in big letters horizontally
basename	Extract the filename+extension
bbanner	Print text in very big letters horizontally vertically
bed	Binary stream editor
bell	Ring bell
bgrep	Search pattern in binary file
bsplit	Split a binary file into separate file chunks containing specified number of bytes
bvi	Binary file Viewer
cal	Display calendar (month or entire year)
cat	Concatinate several files
cc	UnixDos C Compiler front end
cc1	UnixDos C compiler pre processor
cc3	UnixDos C compiler post processor
(u)cd	Change to specified directory and drive!
cf	High speed file compare of two binary files byte by byte
chmod	Change file mode/attribute
clear	Clear the screen
cmp	Compare two files byte by byte
cmpdir	Compare all files in two directories by time/size/contents
comm	Display common/uncommon lines of two files
conv	Convert numbers to/from any base
cp	Copy file(s) to new directory/file
cpio	Copy/restore files to/from archive or move to new directory
cut	Extract columns/fields from a text file
(u)date	Display/set the current date and time
dd	Convert file (ASCII/EBCDIC/upper/lower-case/swap bytes)
df	Display free space on disk(s)
diff	Compare two text files line by line
dio	Direct disc input/output
dirname	Extract the directory name
dos2unix	Convert DOS text files to UNIX
drvname	Extract the only drive letter in a path name
du	Display directories with occupied space
(u)echo	Display text, arguments or environment variables
egrep	Search text file(s) for one or several pattern (search expression)
env	Display/set environment (and run program with new environment)
expr	Calculate the specified expression
extname	Extract the file extension
fgrep	Search text file(s) for one or several pattern (straight text)

file	Display file type
filename	Extract the filename without extension
(u)find	Find files based on name/size/time/type/filemode
getlines	Get range of lines from a file
grep	Search text file for single pattern expression
head	Display beginning of text file
inpbox	Display message and prompt for input
io	Extract specified area within a file and write to another file (at specified offset)
(u)join	Join two sorted text files based on common field
kill	Terminate/Kill processes
line	Read one line from the keyboard
ls	List files (and sort by name/size/time/type)
m4	Advanced macro processor for text files with own language
mak2mk	Convert Visual C Makefile to UnixDos makefile
mkdir	Create new directory(ies)
(u)more	Display text file one screen at a time and allow browsing
msgbox	Display message box
mv	Move file(s) to new directory/file
mvdir	Rename directory
nl	Display line number and text file
od	Display any file contents as octal, decimal, hex or char values
paste	Join several text files
pr	Print a text file
ps	Show current processes
pwd	Display current drive and directory
rm	Remove file(s)/directory(ies)
saveexit	Save last errorlevel value
sed	Extract specified lines and/or insert/substitute text
setenv	Set environment variable(s) (in C-Shell: csh)
show_exp	Show UnixDos argument expansion
slash	Convert filenames to/from UNIX/DOS style
sleep	Wait for specified number of seconds
(u)sort	Sort file based on fields, columns or entire line
split	Split a file into separate file chunks containing specified number of lines
stderr	Redirect output from STDERR to STDOUT
strings	Display the text sections of any file (usually binary)
sum	Display the checksum of file
tail	Display the end of a text file
tee	Display contents of a pipe and store also into file(s)
test	Test any condition (numeric/text/file related)
(u)time	Run a program and display elasped time
touch	Change file time
tr	Translate a set of characters
txtbox	Advanced interactive text viewer
uniq	Find unique lines/fields in sorted text file

unix2dos	Convert UNIX text files to DOS
uptime	Display elapsed time since last reboot
usrchar	Prompt user for single character
usrprmpt	Prompt user for text
uudecode	Convert ASCII file (from uuencode) back to binary
uuencode	Convert binary file to ASCII codes for transmission
wait_end	Wait for process completion
wc	Count lines/words/characters in file
which	Find programs by name

Introduction to the Standard Argument Expansion in UNIX/UnixDos

In many cases when you want to reference many files on the command line you will have to list all the filenames individually
Example:
You have the following three files in the current directory 'do_arg.c', 'more.c' and 'wc.c'.
To copy all these to the '/tmp' directory you have to type:
```
cp do_arg.c more.c wc.c \tmp
```

You can imagine this can be very tedious and error prone to list many filenames on the command line. So more than 25 years ago when the UNIX environment was created the "argument expansion" was introduced.
Example:
So now you can just type:
```
cp *.c /tmp
```

The "argument expansion" allows you to specify an entire group of files instead of having to type each individual filename on the command line. Certain special characters (or <u>meta characters</u>) have been introduced to allow you to specify exactly which file you want to select (see below for more details). In our example the '*' will be replaced by any filename and '*.c' will be "expanded" to the list of all files ending with '.c'.

In UNIX this "expansion" job is centrally done automatically for you no matter which utility you use at the command line from the "Shell" which surrounds the operating system like the shell in the sea surrounds a pearl (sh,csh,ksh).

In Winodws3.1/95/NT/MSDOS the command line is completely "dead" because the "shell" (command.com) does not do any expansion, only some internal commands have a very limited "expansion" inbuilt (DIR, COPY, etc...).

To fill this painful gap we included in all individual UnixDos utilities the full "expansion" logic, so that you don't have to remember which utility has expansion and which not and you don't need any special installation like in other products (MKS /etc/glob.exe).

The most powerful argument expansion can (still) be found in the UNIX environment, while DOS is still very primitive.
Example:
To copy your C sources and include files in DOS you have to type two separate commands:
```
DOS: copy *.c  A:
     copy *.h  A:
```

While UNIX and UnixDos allows you to do the job in one go:

```
cp *.c *.h  a:
```

In UNIX and UnixDos you could even use a more sophisticated expression (see below under <u>Regular Expression for File Matches</u>):
```
cp *.[ch]  a:
```

The UnixDos utilities provide even more power than the UNIX argument expansion thru the <u>UnixDos Argument Expansion Extensions</u>.

Regular Expression for File Matches

The UNIX and UnixDos "Regular Expression" allows you to find specific filenames. To communicate your search criteria the following <u>meta characters</u> are available in UNIX and UnixDos:

<u>Wildcard (*)</u>
<u>Wildspace (?)</u>
<u>Character List ([ ])</u>
<u>Inverted Character List ([^   ])</u>
<u>Escape Meta Character (\)</u>
<u>Environment Variable UNIX Style ($Env)</u>
<u>Run program(s) as Argument (`prog`)</u>
<u>No Expansion</u>
<u>UnixDos Configuration File (unixdos.cfg)</u>
<u>Combining Meta Characters</u>

Wildcard (*):

Matches any number of characters.

Example:

`a*`   matches all files starting with 'a' of any length.

`*.c`   matches all files with the extension '.c'

*a*b*   matches all files which contain an 'a' followed by a 'b'

Wildspace (?):

Matches any single character.

`??.c` matches all files starting with exactly 2 characters and extension '.c'

`*.??` matches all files which have a 2 letter file extension

?a*.c matches all files which have an 'a' as the second letter and extension '.c'

Character List   ([   ]):

Matches any single character which is part of the specified list/range.
<u>Example:</u>

`[xyz+]*`   matches all files starting with x,y or z or +

`[a-f]*`   matches all files starting with a,b,c,d,e or f (range)

`[a-fxyz]*` matches all files starting with a,b,c,d,e, f (range) or x,y,z(list)

Inverted Character List   ([^   ]):

Matches any single character which is <u>NOT</u> part of the specified list/range.
<u>Example:</u>
`[^xyz]*`   matches all files not starting with x,y or z
`[^a-f]*`   matches all files not starting with a,b,c,d,e or f (range)
`[^a-fxyz]*` matches all files not starting with a,b,c,d,e, f or x,y,z

\c   Escape Meta Character:

Disable the special meaning of a meta character 'c'

<u>Example:</u>

`ab\?` matches all files starting with 'ab' followed by   '?'

Environment   Variable UNIX Style ($Env):

UNIX style access to an environment variable '$env'.

<u>Example:</u>
```
set CLST=*.c
ls $CLST
```

will be initially expanded to:
```
ls *.c
```

which is then translated finally to:
```
ls {all C Sources}
```
(instead of 'ls' any UnixDos utility could be used!)

Run Program(s) as Argument (`prog`):

Will execute all command(s) within backward single quotes 'cmd' and places each line in the output as a new argument on the command line.
<u>Example</u>:
```
banner `udate`
```

Will show run 'udate' (which gives the "Day-of-the-Week Month Day") and then place the output as one argument for the 'banner' program to show the date in big letters:
```
#       #                                    #                            #####      #
#   #   #   ######  #####          #  #   #     #    ####        #       #   ##
#   #   #   #           #      #         # #   #    #   #   #     #                #   # #
#   #   #   #####   #       #        #     #  #    #   #                 #####       #
#   #   #   #           #       #     #######  #     #   #  ###        #              #
#   #   #   #           #       #     #        #  #   #   #     #       #              #
 ## ##    ######  #####          #        #    ####     ####         #######
#####
```

This is a very powerful feature which allows you to run one or <u>several</u> commands and convert each line in the output to a new argument.
You can even use redirection and piping within your command sequence! using the replacements for the redirection symbols ({}) defined in the <u>UnixDos Configuration File</u>

<u>Example</u>:
```
banner `udate } wrk.tmp;cut -f3,2,5 { wrk.tmp`
```

This will first run 'udate' and place the output into file 'wrk.tmp'.
Then it will redirect file 'wrk.tmp' as input into 'cut' and select fields 3,2 and 5 in that order.

Might show.
```
 #####      #              #                            #     #####
#####
#       #   ##              # #   #   #   ####         ##   #     # #
#
        #   # #             #   #   #   #   # #   #       # #   #     # #
#
 #####      #              #     #   #   #   #                 #     ######
######
#           #              ####### #   #   #  ###        #              #
#
#           #              #     # #   #   #     #       #    #     # #
#
#######  #####              #     #   ####     ####         #####   #####
#####
```

To eliminate the blanks use:
```
banner `udate}x1;cut -f3,2,5{x1}x2;sed \"s: ::g \" x2`
```

This will first run UDATE and place the output into file 'x1'.
Then it will redirect file 'x1' as input into CUT and select fields 3,2 and 5 in that order
and write the output to file 'x2'. Then it will run SED using 'x2' as input and replace any
blank with nothing.

Might show.

```
 #####      #        #                       #      #####   #####   #####
#     #    ##       # #      #       #    ####     ##    #       # #      # #        #
      #   # #       #   #    #       #   #        # #    #       # #      # #      # #
 #####    #   #    #    #    #       #  #         #       ######  ######  ######
#         #######  #    #    #   ###     #              #       # #      #        #
#         #      # #    # #    #    #       #    #    #       # #      #       # #       #        #
#######  #####  #      #    ####    ####   #####    #####   #####    #####
```

No Expansion ('arg'):

If arguments are enclosed in forward single quotes any expansion is suppressed.

<u>Example:</u>

```
ufind / -name *.c' -ls
```

Will NOT expand '*.c' into the list of C source but passes this "regular expression" on to the "ufind" UnixDos utility which does the expansion internally.

UnixDos Configuration File (unixdos.cfg):

Since the default characters for piping(|), input(<) and output(>) will be immediately executed from the "Shell" we replaced them with the following defaults:

Comment	UNIX/DOS	UnixDos	Config Command	Example
pipe	|	#	PIPE=#	ls # wc -lt
input	<	{	IN={	ls } x
output	>	}	OUT=}	wc { x
separator		;	SEP=;	ls } x;wc { x
file exchange		~	FEX=~	cut -c2-10
~mytext				
workfile			WRK=~	(internal)

You can redefine our own values by placing "Config Commands" in the UnixDos configuration file which resides in the same directory where the UnixDos utilities are located.

Example UnixDos Configuration file (c:\unixdods\unixdos.cfg):
PIPE=#
IN={
OUT=}
SEP=;
FEX=~
WRK=wrk.tmp

Combining Meta Characters:

You can use any combination of the above meta characters.
<u>Example:</u>
[a-c]m*.[^ch]*

matches all files starting with a,b or c,
always followed by the letter m,
followed by any number of characters (or none),
always followed by the '.' character,
followed by any character except c and h (in the file extension)
followed by any number of characters (or none)

UnixDos Argument Expansion Extensions

In addition the already powerful file search mechanism in UNIX and UnixDos (see Regular Expression for File MatchesREG_EXP_FILE) UnixDos provides the following added features:

Full Inversion (!{re})
Selective Inversion (!{re1}!{re2})
File List Argument (@lst)
Environment Variable DOS Style (%Env_ or %Env%)
Automatic Environment Variable ($UDPROG)
Text Exchange (~[fn~]text)
Environment Variable Exchange (~[fn~]$ENV)

{re} stands for any Regular Expression for File MatchesREG_EXP_FILE

Full Inversion (!{re})

Matches any file NOT fulfilling the regular expression '{re}'.

Example:

`!*.c` matches all files NOT having the file extension '.c'

!*.zip   matches all non-zip files

`!*.*` matches all files which have NO file extension

Selective Inversion (!{re1}!{re2}):

Matches any file NOT fulfilling the regular expression '{re1}' within all files matching '{re2}'.

<u>Example:</u>

`![d-m]*!*.c`   matches all files with file extension '.c' which
do NOT start with d,e,f,g,h,i,j,k,l,m

File List Argument (@lst):

Reads text file 'lst' and places each line as a separate argument
<u>Example:</u>
Text file 'new.lst' contains the following 5 lines:
```
*.c
*.cpp
*.h
*.hpp
*.ipp
```
The following command line:
```
ls @lst
```

will be initially expanded to:
```
ls *.c  *.cpp *.h *.hpp *.ipp
```

which is then translated finally to:
```
ls {all C Sources and include files}
```
(instead of 'ls' any UnixDos utility could be used!)

Environment Variable DOS Style (%env or %env%):

DOS style access to an environment variable '%env' or '%env%'.

<u>Example:</u>

```
set CLST=*.c
ls %CLST
```

will be initially expanded to:

```
ls *.c
```

which is then translated finally to:

```
ls {all C Sources}
```

(instead of 'ls' any UnixDos utility could be used!)

Automatic Environment Variable   ($UD{prog}):

Each UnixDos Utility automatically includes the contents of its automatic program Environment Variable. The name is composed of "UD" in the beginning directly followed by the capitalized letters of the program name.
Example:
UnixDos utility "df" would use env Variable "UDDF".
This means you can easily implement the desired default behavior of any UnixDos utility, by setting the corresponding environment variable! - for example to NOT get error message for
files from RM which have already been erased:
```
set UDRM=-s
```

Examples:
```
set UDLS=-lt
ls *.c
```

which is then translated to:
```
ls -lt *.c
```

To alert you about the automatic inclusion of the default arguments it will display the following message:
```
Adding default environment: $UDLS=-lt
```

If this message bothers you insert the '!' character:
```
set UDLS=!-lt
```
This will suppress the message but still use the values after '!'

If you want to specify several arguments use the UnixDos argument separator ';':
```
set UDLS=-lt;*.c;*.h;*.cpp;*.hpp
```

To change the separator specify your separator character in the UnixDos configuration file
(instead of 'ls' any UnixDos utility contains this feature, i.e.
```
set UDGREP=...
```
is the "Automatic Variable" for UnixDos GREP)

Text Exchange (~[fn~]text):

In some cases you want to pass text into a UnixDos program where a filename is required. Usually you would create a new file, enter the text and use that filename as an argument on the command line.
UnixDos allows you to simplify these steps by directly specifying the text on the command line.
<u>Example:</u>
```
cut -c1-10 "~This is straight text"
```

In this example UnixDos will write the text to a temporary work file (fileex01.tmp, fileex02.tmp...) and then run the command as if you had typed (cut the first 10 charcater out of the text):
```
cut -c1-10 fileex01.tmp
```

and display the result:
```
This is st
```

The temporary work file 'fileex01.tmp' will contain the full text:
```
This is straight text
```

If you want to specify your own work filename specify the name in front of the text:
```
cut -c1-10 "~mytext.tmp~This is straight text"
```

This will write the full text to 'mytext.tmp' and then show:
```
This is st
```

This feature allows you to pass text "text" into any UnixDos program as a filename.

Environment Variable Exchange (~[fn~]$Env):

In some cases you want to pass the contents of an environment variable into a UnixDos program where a filename is required. Usually you would create a new file, write the contents of $Env to that file and use that filename as an argument on the command line. UnixDos allows you to simplify these steps by directly specifying the environment variable on the command line.

Example:
```
cut -f2 "-d;" ~$PATH
```

In this example UnixDos will write the contents of the PATH environment variable which contains all the program directories to a temporary work file (fileex01.tmp, fileex02.tmp...) and then run the command as if you had typed (cut the first 10 character out of the text):
```
cut -f2 "-d;" fileex01.tmp
```

and display the result for example:
```
C:\WINDOWS\SYSTEM32
```

The temporary work file 'fileex01.tmp' will contain the full text for example:
```
C:\WINDOWS;C:\WINDOWS\SYSTEM32;F:\MSDEV\BIN;C:\UNIXDOS
```

If you want to specify your own work filename specify the name in front of the environment variable:
```
cut -f2 "-d;" ~mypath.tmp~$PATH
```

This will write the full text to 'mypath.tmp' and then show:
```
C:\WINDOWS\SYSTEM32
```

This feature allows you to pass the contents of any environment variable into any UnixDos program as a filename.

Regular Expression for Text Searches

Some UnixDos utilities use "regular expression" to select specific text pattern from text files.
The following UnixDos utilities use text searches:

GREP, EGREP, FGREP   search for pattern in text file(s)
SED   Find and replace pattern in text file(s)
(U)FIND   Find files using regular expressions
(U)MORE   View text file(s) and search for text while viewing it
WHICH   Search for programs along the PATH environment

The regular expression for text searches contain some of the meta character available in the Regular Expression for File Matches:

Character List ([   ])
Inverted Character List ([^   ])
Escape Meta Character (\)

In addition the following text search meta characters are also available:

Line Start (^)
Line End ($)
Wildcard for Text Searches (.*)
Wildspace for Text Searches (.)
Group Start \(
Group End \)
Group Reference \n

Line Start (^):

Refers to the beginning of each line.

<u>Example:</u>

`^a*`   matches all lines starting with letter 'a'.

`^[^m-w]*`   matches all lines NOT starting with m,n,o,p,q,r,s,t,u,v,w

Line End ($):

Refers to the end of each line.

<u>Example:</u>

`a$`   matches all lines having the letter 'a' in the last column

`[^a-f]$`   matches all lines NOT having the 'a' thru 'f' in the last column

Wildcard for Text Search (.*)

Matches any single character repeated any number of times (including zero times).
(? is only used for File matching!)

Example:

`^g.*[a-f]`   matches all lines starting with 'g' and the always followed by a,b,c,d,e or f

`[xX].*[0-9A-F]$`   matches all lines containing 'x' or 'X' and then being followed by a hexadecimal digit (0-9, A-F)

Wildspace for Text Search (.)

Matches any single character. (? is only used for File matching!)

<u>Example:</u>

`^..[a-f]`   matches all lines starting with exactly 2 characters always
followed by a,b,c,d,e or f

`[D0-9].$`   matches all lines ending with D or a single digit and always
followed by any single character

Group Start \(

The group start meta character only applies to the <u>SED</u> UnixDos command.
This meta Indicates the beginning of a substitution group.
It is used in conjunction with the group end and group reference meta character.

<u>Example:</u>
This example will get the list of environment variables and switch the value   with the name and replace the '=' sign with a text 'is the value of ':

```
env | sed "s:\(.*\)=\(.*\):\2 is the value of \1:"
```

Group End \(:

The group end meta character only applies to the <u>SED</u> UnixDos command.
Indicates the ending of a substitution group.
It is used in conjunction with the group start and group reference meta character.

<u>Example:</u>
This example will get the list of environment variables and switch the value   with the name and replace the '=' sign with a text 'is the value of ':

```
env | sed "s:\(.*\)=\(.*\):\2 is the value of \1:"
```

Group Reference \n:

The group reference meta character only applies to the <u>SED</u> UnixDos command.
Indicates the ending of a substitution group.
It is used in conjunction with the group start and group end meta character.
'n' refers to group number   in the range from 1 to 9

<u>Example:</u>
`\3`   refers to group number 3

<u>Example:</u>
In this example we use the list of all environment variables:
`env`

```
HELPFILES=c:\windev\help
HOMEDRIVE=C:
HOMEPATH=\users\default
INCLUDE=c:\madhur\src\dx\def;\msvc\include
INIT=C:\MSVC
ISLVINI=t:\pvcs
LDFLAGS=/B/NOI/NOE/SE:1000/ST:4000
LDLIB=llib
LIB=c:\madhur\lib;c:\msvc\lib
LIBPATH=t:\pvcs\dos
```

In this example we switch column 2 with column 1 using '=' as delimiter
and insert the text 'is the value of':
`env | sed "s:\(.*\)=\(.*\):\2 is the value of \1:"`

```
C: is the value of HOMEDRIVE
\users\default is the value of HOMEPATH
c:\madhur\src\dx\def;\msvc\include is the value of INCLUDE
C:\MSVC is the value of INIT
t:\pvcs is the value of ISLVINI
/B/NOI/NOE/SE:1000/ST:4000 is the value of LDFLAGS
llib is the value of LDLIB
c:\madhur\lib;c:\msvc\lib is the value of LIB
t:\pvcs\dos is the value of LIBPATH
```

How to get Command Line Help

Each UnixDos utility has an inbuilt short help screen explaining all the options and switches available for that utility.

Most of the UnixDos utility require arguments (i.e. CP or MV) and you will automatically get a short help display when you just type the utility name.

Example:
cp requires at least two arguments (the "source" and "destination"):
```
cp
```

Will display (after the initial 2 intro lines):
```
cp (Copy files)
usage: cp [-{l|f|v}] {src} {dest}
-a     = attr  : don't keep old file attributes
-f     = force : always remove old files (even if read-only)
-l     = list  : show which file is currently copied
-t     = times : don't keep old modification times
-v     = verify: verify that input is identical to output
src    = source file(s)
dest   = destination (has to be directory if several source files)
```

Those UnixDos utilities which do NOT require any arguments will display the help screen when you enter the argument '-?'.

Example:
LS requires no arguments (in which case it display all files in the current directory); to get help type:
```
ls -?
```

Will display (after the initial 2 intro lines):
```
ls (List files and directories)
usage: ls [-ACDFRcdelprstuxy] [-w {wid}] [files]
-A = ARCHIVE  : show also DOS Archive flag (-rw-rw-rw-A)
-C = COLUMN   : suppress multi column output
-D = DosAttrib: suppress display of DOS Attributes (A=Arch, S=Sys, H=Hidden)
-F = FileAttr : suppress '/' for directories, '*' for executables
-R = RECURSIVE: list all subdirectories
-c = CREATE   : show time of file creation (default is modification)
-d = DIRECTORY: show only directory detail NOT all files in it
-e = EXTEND   : don't show filesize in extended format (30,764,678)
-l = LONG     : long listing (default sort by name)
-p = PERSIST  : show '/' and '*' (-F) even when output is redirected
-r = REVERSE  : reverse the sort order
-s = SIZE     : sort by file size (show 1k increments)
-t = TIME     : sort by time
-u = ACCESS   : show time of last file access (default is modification)
-x = ACROSS   : sort across lines instead of down the column
-w = WIDTH    : set max filename width '{wid}' in column output (default is
```

18)

if filename need to be truncated last char is '>': Long Nam>
-y = TYPE   : sort by file type/extension
(If the output is redirected to a file LS automatically prints each file
on a separate line without '/' and '' attributes: ls > lst.tmp (-p will!)*

Differences between DOS and UNIX:

The emulation of the UNIX utilities under DOS and Windows in UnixDos is limited by the parameter of the surrounding DOS and Windows operating systems.
Below is a list of all the basic differences in terms of the command line interface.
(We are not getting into any kind of internal differences of the operating systems!)

Filename Separator:

UNIX separates the filename and directory with   the forward
slash '/' (e.g. "/usr/src/file.c").
Unfortunately DOS changes this and is using the backward slash '\' (e.g.
"\USR\SRC\FILE.C").
UnixDos allows you to enter files in both forms and will always
convert to the UNIX notation unless you enclose in single quotes (e.g. '\usr\src\file.c')

Filename Length and Style:

Windows 95/NT have finally caught up to UNIX with the filelength up to 255 characters
and any form and extension is permitted.

MSDOS/Windows 3.11 has the following fixed structure for the filenames:
up to 8 places for the filename + up to 3 for the extension (e.g. FILE.TXT).

UNIX allows a longer filename (14 or in "Berkeley UNIX" up to 255   places) and uses
the separating dot just as any other character - so you can have several "extensions" of
various lengths ('file.longext' or 'programname' or 'lex.a.c' are valid filenames)
UnixDos can handle up to 15 directories levels deep.

User groups:

UNIX provides three user groups and connected layers of restrictions:
the owner, the group and others
DOS/Windows 3.11 and Windows 95 has only one: the owner
Windows NT has security provisions similar to UNIX - that's why you have to log-on.

File links:

UNIX allows access to the same file using different name(s) - so called "links" (not "hot
dogs").
This great feature is completely missing in DOS/Windows!.

File times (create/modify/access):

UNIX/Windows95/NT maintain 3 different file times:
creation time, modification time and last access time

DOS/Windows3.11 maintains only the modification time.

Pipes:prog1 | prog2

UNIX allows you to run several programs simultaneously and
connect the input of one program to the output of another program
via a "pipe". This communication is done entirely in memory and is therefore
very fast and does not require any additional temporary disk space.

DOS/Windows does not allow you to run two programs simultaneously from the
command prompt so DOS/Windows will run the first program until it completes, store
the output in a temporary disk file and then start the second program and feed it the
temporary disk file (output from prog1) and remove it after it has been read (unless you
interrupt with Ctrl-C) in which case you get lots of "left-overs" sitting around.

Argument expansion:

UNIX will expand the arguments <u>before</u> any program starts using the "Shell" (Bourne
Shell, C Shell or Korn Shell). This means that the program does not have to worry about
translating any expressions into the proper arguments   (i.e. translate '*.c' into the list of
all C source files).

DOS/Windows 3.11/Windows 95(COMMAND.COM) and Windows NT(CMD.EXE) will
not expand any arguments for you. So each UnixDos utility has to do this argument
expansion job individually, because we wanted to avoid the need for any kind of shell
(as MKS or 4DOS needs).

UnixDos Improvements and Changes to "Classic UNIX"

Although UNIX is a great operating system in some specific cases it could use some improvements. So we implemented these overdue changes and make them available in the UnixDos Toolkit.

Filename Sort Order

UNIX utilities just scan the directory sequentially and so you get an unsorted list:
DU, (U)FIND etc.
UnixDos will always sort the top level and each sub level in descending operations so you always will get a sorted list
(unless you want to save a little time and switch it off with "set UDNOSORT=ON")

Argument Expansion

UnixDos provides the full UNIX argument expansion functionality but adds even more power thru the following features (see UnixDos Argument Expansion Extensions):

- full file match inversion: `ls !*.zip`
- selective file match inversion: `ls ![a-m]*!*.zip`
- file contents retrieve: `ls @file.lst`
- file text exchange: `cut -c5-10 ~ThisIsText`   displays:
  `IsText`
- env variable exchange: `cut -f2 "-d;" ~$PATH`
- multiple commands: `uecho `ls } x1.tmp;wc -lt`
  `{ x1.tmp` files`
- automatic environment: `set UDLS=-le; ls x*`
- two level expansion: `uecho '*.c` > c.lst; ls @c.lst`

Current directory (ucd)

If you call (U)CD without any argument under UNIX it will go automatically to the "homedirectory".
In DOS/Windows there is no reliable equivalent so instead the UnixDos (U)CD will show you the current drive and directory.

Check Destination (cp/mv)

Under the UNIX system CP and MV will not check if the last argument on the command line is actually a directory and it will silently create a new file and leave you under the wrong impression that all the specified files have been copied/moved.
Example:
You want to move all C source to the "\tmp" directory, but you make a typo and forget the slash in front of tmp:
`mv *.c tmp`   (instead of: `mv *.c /tmp`)
Instead of finding all your C sources in the "/tmp" directory, you will only find the last C

source in the file "tmp" in the current directory and **all the other C sources are lost**!
CPand MV in UnixDos will first check if the last argument is a directory (when more than 2 files are specified) and give you an error message if this is not the case.

Show Progress (cp/mv/rm)

Under the UNIX system CP, MV and RM have no option which will show you the currently processed files, so that you can see the progress (as CPIO is doing it so nicely even the new destination).
UnixDos CP, MV and RM allows you to see each file as it is processed using the "-l option".

Copy and Move Verify (cp/mv)

Under the UNIX system CP and MV have no verify option which actually re-reads the new copy and compares it against the original.
UnixDos CP and MVprovides you with the verify feature (-v option) to compare the copy against the original using an actual re-read. This applies only in those cases to the MV utility when you move files between drives(file systems).

Honor Field Sequence (cut)

Under the UNIX system   CUT ignores the sequence in which you specify the selected fields.
Example:
`cut -f3,1` is identical under UNIX to:
`cut -f1,3`
UnixDos CUTwill **not** ignore the specified field sequence and will allow you to rearrange the field sequence.

Allow reverse character extraction (cut)

Under the UNIX system   CUT does not allow character extraction counting from the end of the line(s)
UnixDos CUT provides you with a new option (-C) which can extract characters counting from the end of the line!
Example:
`cut -C1-6 ~ThisIsText`
Will produce:
IsText

Show Filename and Hex Values (cmp)

Under the UNIX system CMP does not show you the filenames or any hexadecimal/text values (only octal).
UnixDos CMP lists the filenames first, so you can easily see which values go with which file, and also shows the result in hex and text form:

Offset	x1	x2
3	0x32(2)	0x33(3)
6	0x33(3)	0x32(2)

Show Bytes not Blocks (df)

Under the UNIX system <u>DF</u> displays the free space by default in number of blocks (which varies in size depending on the file system) instead of actual bytes like UnixDos. The MKS and other <u>DF</u> ports even show the wrong result for big SCSI drives or Network drives.
<u>Example:</u>
```
df c: d: e:


C:(C_900MB              )      76,840,960 free (   901,857,280 total)   8.5%
free
D:(D_512MB              )     150,716,416 free (   536,428,544 total)  28.1%
free
E:(E_400MB              )      29,106,176 free (   433,258,496 total)   6.7%
free
=============================================================================
==
   (*** GRAND TOTAL *** )     256,663,552 free ( 1,871,544,320 total)  13.7%
free
```

User-friendly Display(diff)

Under the UNIX system <u>DIFF</u> the comparison result as a command sequence for the ED editor - it is NOT meant for regular humans to read this kind of cryptic output. Also you cannot see the filename only the ">" or "<" character to easily see which text goes with which input file.

UnixDos <u>DIFF</u> will show the differences in an easily readable form with the filenames and also allows you to view the context surrounding the differences.
<u>Example:</u>
```
diff x1 x2

 x1 (Mon Sep 01 16:29:22 1995, 12 bytes)
       x2 (Mon Sep 01 16:29:28 1995, 12 bytes)
 -----------------------------------------------------------------------
       1          1    |A
+      2               |B
-                 2    |C
       3          3    |C
       4          4    |D
 -----------------------------------------------------------------------
       1          1    |A
       2          2    |B
+      3               |C
-                 3    |B
       4          4    |D
 -----------------------------------------------------------------------
```

Faster and focus can be chosen (dircmp/cmpdir)

Under the UNIX system you can compare directories with the DIRCMP shell script. This is a very slow shell script and is not flexible to compare by time/size/contents.

UnixDos CMPDIRprovides a high-speed tool to compare entire directories using the time, size or contents

Show (Mega)Bytes in sorted sequence (du)

Under the UNIX system DU displays the space used by files and directories in blocks (which varies in size depending on the file system) which is very user-unfriendly and does not allow you to see the actual bytes without the empty padding space. Also it displays the result in an arbitrary unsorted sequence, while UnixDos DU always displays the result in sorted sequence.

UnixDos DU will show the result in alphabetical order in actual megabytes (or bytes -b) and allows you to discard the padded empty space (so that you can correctly compare when you copy directory trees to a new media with a different block size!).
Example:
```
du /windows

   Megabytes      (Cluster = 16384 bytes)
      1.4090     /win95/COMMAND
      0.0328     /win95/CONFIG
      0.6226     /win95/Cookies
      0.0492     /win95/CURSORS
      1.4090     /win95/Desktop
      0.0819     /win95/Favorites
      3.5389     /win95/FONTS
      2.8672     /win95/HELP
      0.8192     /win95/History
      4.9971     /win95/INF
      9.6829     /win95/JAVA
      0.1638     /win95/MEDIA
      0.2458     /win95/MSApps
       ---       /win95/NetHood
       ---       /win95/OCCACHE
      0.0328     /win95/PIF
      0.2621     /win95/Recent
      0.0655     /win95/SendTo
       ---       /win95/ShellNew
      6.2751     /win95/Start Menu
      2.7853     /win95/SYSBCKUP
     63.0620     /win95/SYSTEM
      0.0492     /win95/spool
       ---       /win95/TEMP
      5.4395     /win95/Temporary Internet Files
     16.1219     /win95/{Files}
    120.0947     /win95
```

Enable/Disable escape logic and space between arguments(uecho)

Under the UNIX/MKS system ECHO does not allow you to enable or disable the automatic escape character logic and automatic space insertion between arguments.

This might be essential if you work with the problematic backslash in DOS path names - or if you have to join arguments without a separation space in between.
Example:
```
uecho -e $PATH
```

Show results in sorted sequence(ufind)

Under the UNIX/MKS system FIND displays the searched files in an arbitrary unsorted sequence, while UnixDos (U)FIND always displays the result in sorted sequence.

Add -older and -NEWER/-OLDER option (ufind)

Under the UNIX/MKS system FIND does not have the capability to also search for files which are older, while UnixDos UFIND makes this option available and therefore allowing you to search for an actual time window. It also allows to search for after/before or equal with the -NEWER and -OLDER option.

Allow long line length and descend into subdirectories (grep/egrep/fgrep)

Under the UNIX/MKS system GREP, EGREP and FGREPcannot handle lines longer than 1024 bytes which can easily be too short when you work with larger database dumps. They also do not provide a descending option so that you can also easily scan all subdirectories for one or several pattern.
UnixDos GREP, EGREP and FGREPcan handle lines up to 10,000 bytes and allows you to descend into all the subdirectories with the "-d" option:
```
grep -d pattern *.c* *.h
```

Show file details for read-only files (rm

Under the UNIX/MKS system RM does not show all the file details when you use the interactive '-i' option.
UnixDos will show you all details (ls -l) equivalent:
Example:
```
rm -i \tmp\*

-rw-rw-rw-        731 Sep 01 13:17 /tmp/1: ?
-rw-rw-rw-         26 Sep 01 15:49 /tmp/a.b.c: ?
-rw-rw-rw-         26 Sep 01 15:50 /tmp/a.b.commm: ?
```

Allow non text for substitution (sed)

Under the UNIX/MKS system SED cannot substitute text for LineFeeds (\n) etc.
in case you want to break up fields into lines.
UnixDos SED allows you to substitute also with control codes (\t\n\r or \hex code).
Example:
```
sed "s/;/\n/g" ~$PATH

F:\MSDEV\BIN
C:\U
```

```
C:\WIN95
C:\WIN95\COMMAND
```

Allow long line length of 10,00 bytes (sed)

Under the UNIX/MKS system <u>SED</u> cannot handle lines longer than 1024 bytes which can easily be too short when you work with larger database dumps
UnixDos <u>SED</u> can handle lines up to 10,000 bytes.

Bigger roll-back (tail)

Under the UNIX/MKS system   <u>TAIL</u> cannot go back more than 4096 bytes.
UnixDos <u>TAIL</u> can go back up the 20,000 bytes.

UnixDos Competition

MKS

The biggest competitor to UnixDos is the MKS Toolkit from Canada.
It is a great package but has the following limitations and drawbacks:

* All utilities need the '/etc/glob.exe' file installed on each drive you will use for the argument expansion.
  (UnixDos does NOT need any special installation or any '/etc' directory)

* Argument expansion fails in directories with more than approximately 800 files
  (UnixDos handles up to 3000 arguments on the command line)

* The text line length is usually limited to 1024 character
  (UnixDos work up to 10,000 characters)

* Argument expansion has no inversion(!*.c) or file list input (@lst)

* Does not have most of the <u>UnixDos improvements</u>

* higher price, no demo

WinXs

This package is shareware for $15 but strictly runs all the UNIX utilities only as Windows applications, so you will have to point and click a lot and it is missing lots of utilities.

<u>DOSNIX</u>

This package is shareware for $35 but has no "regular expression" or any of the other regular UNIX argument expansions (try: `ls a*` or `ls *exe`). It also misses lots of UNIX utilities. Some utilities are still in a very early stage (like df which shows you the free bytes but no total bytes or percent used).

UnixDos Environment Variables

To tailor the behaviors of several UnixDos utilities the following environment variables are available to change certain default parameter.

```
set EDITOR=b              Editor to be used in "more" utility (default is vi)
set PATH=....             Main program directory list used from "which" utility
```

since UnixDos is such a long code for all the environment variables UD is used instead:
```
set UDDIFF=vdiff -b       Defines the 'diff' program for the CMPDIR utility
set UDEXIT=c:\myexit.txt  Defines the filename to store the exit value (test.exe,
saveexit.bat)
set UDNETLST=G-Z              Defines the list of network drives (used by df) (default
is F-Z)
set UDNOSORT=ON           suppress automatic file/directory sort during expansion
set UDPATHEXT=.exe;.bat;.pif    define custom executable file extension list   (ls,
which)
set UDRETRY=5             number of retries when verify fails (cp, mv) (default is 10)
set UDVER=ON              Show the UnixDos utility version
set TEMP=c:\temp          point to temporary work directory (if not set root "/" is used)

set UDAF                  Custom environment for AF utility
set UDGREP                Custom environment for GREP utility (and so forth for all
utilities)
```

Argument Expansion Test Module: SHOW_EXP

The powerful UnixDos argument expansion can get a bit confusing, so we created a small test utility, which just echoes the arguments you supplied and shows you the final arguments which would be passed into your UnixDos program.

<u>Example:</u>
In this example we first set the UnixDos automatic environment variable for SHOW_EXP (UDSHOW_EXP) containing a regular expression for file matches. On the SHOW_EXP command line we supply the LIB environment variable and call <u>UDATE</u> to get the current date and time as an argument:

```
set UDSHOW_EXP=au*.bat;*.sys
c:
cd \
show_exp $LIB `udate`
```

Adding default environment: $UDSHOW_EXP=au.bat;*.sys*
Arguments before expansion:
```
-------------------------------------------------------
IN:  argv[ 0] = c:\unixdos\SHOW_EXP.EXE
IN:  argv[ 1] = $LIB
IN:  argv[ 2] = `udate
```

Arguments after expansion:
```
-------------------------------------------------------
OUT: argv[ 0] = c:/unixdos/show_exp.exe
OUT: argv[ 1] = autoexec.bat
OUT: argv[ 2] = config.sys
OUT: argv[ 3] = io.sys
OUT: argv[ 4] = msdos.sys
OUT: argv[ 5] = pagefile.sys
OUT: argv[ 6] = c:\lib;c:\msvc\lib
OUT: argv[ 7] = Thu Aug 29 16:00:06 1996
```

Notice how the following expansions take place:
First the automatic environment for SHOW_EXP is pulled in and expanded,
We used the separator ';' to specify two regular expressions "au*.bat" and "*.sys":
```
UDSHOW_EXP  ->    OUT: argv[ 1] = autoexec.bat
                  OUT: argv[ 2] = config.sys
                  OUT: argv[ 3] = io.sys
                  OUT: argv[ 4] = msdos.sys
                  OUT: argv[ 5] = pagefile.sys
```

Then the general environment variable "LIB" is expanded:
```
$LIB        ->    OUT: argv[ 6] = c:\lib;c:\msvc\lib
```

And finally the UDATE utility is called and the result becomes the last argument:
```
`udate`     ->    OUT: argv[7] = Thu Aug 29 16:00:06 1996
```

Future Plans

We are currently working on a user-friendly WINDOWS interface for all/most of the UnixDos utilities, because it can be quite intimidating to remember all those cryptic UNIX command line options.

The following UNIX utilities are currently NOT part of the current UnixDos package: (maybe in the future they will!)

Programmer Development utilities (awk,   lex,   yacc)
Shells (sh, ksh, csh)
UNIX editor (vi)
Speller (spell, edspell, eddict)

List of UNIX programs not part of UnixDos

Below is a list of all the UNIX utilities which have not been ported and included in the UnixDos toolkit usually because they apply to administrative task which do not exist under Windows/MSDOS or are done very differntly by other utilities.

Program	Comment
ar	done my programing environment
awk	maybe as a separate product in the future
col	filter reverse line feeds
chgrp	no groups (done by Microsoft NT operating system)
chown	no owners (done by Microsoft NT operating system)
cron/crontab	there are many scheduler programs already
csh	C Shell interactive command line environment (maybe in future release)
ed	ancient single line editor
mail	part of Microsoft operating system
kill	maybe will be ported in a future release
id	done by Microsoft operating system
ld	done my programing environment
lex	maybe as a separate product in the future
ln	link symbolic names to files (that is a big whole in Windows/MSDOS!)
lp/lpstat	done by Microsoft operating system
make	done my programing environment
nm	done my programing environment
nohup	run command immune to interrupts
pack	there is already pkzip and WinZip available
passwd	done by Microsoft NT operating system
pcat	there is already pkzip and WinZip available
pg	too similar to more
rmail	part of Microsoft operating system
rsh/sh	shell already part of Microsoft operating system
spell	part of most wordprocessor systems
strip	done my programing environment
stty	done by Microsoft operating system
su	(done by Microsoft NT operating system)
tar	CPIO is much more powerfull so we dropped TAR
umask	shell specific
uname	maybe in future release
uucp/uu*	done by Microsoft operating system
wait	shell specific
vi	already many VI product out there
wall	already done by network utilities
who	already done by network utilities
yacc	maybe as a separate product in the future

UNIXDOS SOFTWARE LICENSE AGREEMENT

LICENSE:
You may:
1. Use the UNIXDOS programs on a single computer for up to 30 days without a license.
2. Use the UNIXDOS programs on a single computer beyond the 30 day trial period if you have purchased a UnixDos license.
3. Install the UnixDos License on ONE machine per UnixDos License using the UD_MGMT utility.
4. Copy the program into any machine readable or printable form for backup in support of the program on the single machine.
5. Modify the program and/or merge it into another program for your use on the single machine. Any part of the licensed program merged into another program will continue to be subject to the terms of this agreement.
6. Freely distribute the UNIXDOS programs to any other machine but not run the installation utility (UD_MGMT) which will allow the UNIXDOS software to run beyond the trial period of 30 days.
7. Move the software and license to another machine using the UD_MGMT utility with the Uninstall option.
8. request for a new registration key in the rare event of a hard disk failure before you could run the UD_MGMT uninstall procedure.

You may NOT:
1. Run the installation process on any other machine than the single computer after you installed the UNIXDOS software unless you purchased another License for that machine.
2. Run the UNIXDOS software on more than your single computer beyond the 30 day trial period.
3. use, copy, or modify the program or any copy, modification or merged portion, in whole or in part, except as expressly provided for this license.

If you transfer possession of any copy, modification, or merged portion of the program not permitted by this license to another party, your license is automatically terminated.

TERM:
This license is effective until terminated. You may terminate it at any time by destroying the program together with all the copies, modifications, and merged portions in any form. It will also terminate if you fail to comply with any term or condition of this agreement. You agree upon such termination to destroy the program together with all copies, modifications and merged portions in any form.

LIMITED WARRANTY:
This program is provided "AS IS" without warranty of any kind, either expressed or implied, including, but not limited to the implied warranties or merchantability and fitness

for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the entire cost of all necessary servicing, repair and correction.

UnixDos is delivered via download from it's   Internet site http://www.ProfSoftware.com.

Professional Software Solutions does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.
Professional Software Solutions will provide support via email and release updates of the software at its own discretion for free or an upgrade fee, but is under no obligation to respond to support requests or provide updates.

However, Professional Software Solutions warrants the diskettes on which the program is furnished to be free from defects in materials and workmanship under normal use for a period of thirty(30) days from the date of the purchase. If during this 30 days period the diskettes should become defective and unusable, return them to Professional Software Solutions for a no-charge replacement.

ACKNOWLEDGMENT:
You acknowledge that you have read this agreement, understand it and agree to be bound by its terms and conditions. You further agree that it is the complete and exclusive   statement of the agreement between us.

REFUNDS:
UnixDos can be tested and used without any limitation or reduction in functionality for a period of 30 days. We therefore are NOT issuing any refunds for the purchase of UnixDos.

NOTE:
The original diskettes in recognizable form may be required in order to receive repair or update releases of this software product.

DISCLAIMER:
Burkhard Eichberger and Professional Software Solutions have taken due care in preparing this manual and the programs and data on the electronic disk media (if any) accompanying this package including research, development, and testing to ascertain their effectiveness.

Professional Software Solutions makes no warranties as to the contents of this package and specifically disclaims and implied warranties of merchantability or fitness for any particular purpose. Professional Software Solutions further reserves the right to make changes to the specifications of the program and the contents of the package without obligation to notify any person or organization of such changes.

TRADEMARKS:

UNIXDOS is a trademark of Burkhard Eichberger/Professional Software Solutions.
UNIX is a trademark of X/Open Company Limited.
IBM and OS/2 are trademarks of International Business Machines Corporation.
MS-DOS, NT, Windows   and Windows NT are trademarks of Microsoft Corporation.
INTEL is a trademark if Intel Corporation.
NFS and PC-NFS are trademarks of Sun Microsystems, Inc.
MKS is a trademark of Mortice Kern Systems, Inc.

Professional Software Solutions
1626 North Wilcox Suite #252
Los Angeles, California 90028
unixdos@pobox.com
http://www.ProfSoftware.com

Practical Applications of the UnixDos Toolkit

In this section we present a few practical examples on how to actually use the main UnixDos utilities. All the names of the UnixDos utilities are underlined and when you click on the name you automatically jump to the detailed description of this utility in the UnixDos Reference ManualExamples1,2 and 7 are implemented also as icons, so that you can run them directly.

If you have superpad as your notepad replacement make sure you view the output in System Fixed Font(Format/Select System Fixed Font) so that the display lines up.

If you want that the programs automatically close after completion, click with the right mouse button to open the Properties window, go to the "Program" tab and click on "Close on exit".

List of Examples:

Example1: Get quick overview of your free space on ALL your drives
Example2: Find space hogs (Hit list of large files sorted by size)
Example3: Custom process on a list of files (i.e. replace something for many files)
Example4: Find a text document anywhere on your machine using a text search
Example5: Get used Disk Space and Directory Overview
Example6: Cleanup temporary files
Example7: Find all files you worked on today
Example8: Alphabetical list of all files on your machine
Example9: Measure elapsed Time for (batch) process
Example10: Find directories in the root directory
Example11: String numbers on one single line
Example12: User Prompt Example
Example13: Loop Example
Example14: View many small files together (i.e join all the batch files on your computer)
Example15: Return to your home directory in a batch file
Example16: Find the location of a program
Example17: Break a line with fields into several lines

1: Get a quick overview of free space on ALL your drives

Often you need to know where you have the biggest free space or how much the capacity of all your drives are. Use the <u>DF</u> command:
```
df -a
```

<u>The output might look like:</u>
```
C:(C_900MB              )     73,302,016 free (   901,857,280 total)   8.1%
free
D:(D_512MB              )     79,224,832 free (   536,428,544 total)  14.8%
free
E:(E_400MB              )     46,317,568 free (   433,258,496 total)  10.7%
free
F:(F_512MB              )     25,722,880 free (   536,428,544 total)   4.8%
free
G:(F_1GB_MUSIC          )    161,873,920 free ( 1,053,999,104 total)  15.4%
free
L:(DRG1                 )              0 free (   573,407,232 total)   0.0%
free
==============================================================================
==
  (*** GRAND TOTAL *** )    386,441,216 free ( 4,035,379,200 total)   9.6%
free
```

To exclude the CD-ROM on drive (L:) use:
```
df C: d: e: f: g:
```

<u>New Result:</u>
```
C:(C_900MB              )     73,302,016 free (   901,857,280 total)   8.1%
free
D:(D_512MB              )     79,224,832 free (   536,428,544 total)  14.8%
free
E:(E_400MB              )     46,317,568 free (   433,258,496 total)  10.7%
free
F:(F_512MB              )     25,722,880 free (   536,428,544 total)   4.8%
free
G:(F_1GB_MUSIC          )    161,873,920 free ( 1,053,999,104 total)  15.4%
free
==============================================================================
==
  (*** GRAND TOTAL *** )    386,441,216 free ( 3,461,971,968 total)  11.2%
free
```

You can store your current list of drives in a file and then just refer <u>DF</u> to it:
In this example we use the file 'c:\my.drv' for the fist of drives:
<u>c:\my.drv:</u>
```
c:
d:
e:
```

```
f:
g:
```

Now we can call <u>DF</u> without having to type all the drive letters again and again:

```
df @c:\my.drv
```

2: Find space hogs (Hit list of large files sorted by size)

Many times you might get tight on disk space and want to know where all the space goes.
The following command will search for the 20 biggest files on drive C D and E:
du -h10 C: D: E:

The output might look like:
List of the 10 largest files sorted by size in descending order:
```
--------------------------------------------------------------
 20,322,482 e:/MSVC/HELP/VCBKS15.HLP
 16,610,031 e:/MSVC/HELP/VCSDK15.HLP
 15,581,184 e:/MSVC/HELP/VCBKS15.HLP
 11,090,254 e:/WINDEV/WINDEV.ZIP
 10,919,006 e:/PMW/PMW.WRK
 7,245,478 e:/DRG/MSDNCD15.IDX
 4,325,728 e:/MSOFFICE/ACCESS/MSACC20.HLP
 4,246,784 e:/MSOFFICE/EXCEL/EXCEL.EXE
 3,490,816 e:/MSOFFICE/WINWORD/WINWORD.EXE
 3,390,373 e:/VB30/WINAPI/WIN31WH.HLP
```

As an alternative you can search for all the files which are larger than a certain specified size. The following command will search the entire current hard disk C: D: and directory E:\TMP and show you very quickly all the files which are larger than one megabyte (1,000,000 bytes):
ufind C: D: E:/tmp -size +1m -ls   (or)
ufind C: D: E:/tmp -size +1000000c -ls

The output might look like:
```
 -rw-rw-rw- 41943040 Aug 22 00:12 c:/PAGEFILE.SYS
 -rw-rw-rw- 10485760 Aug 22 16:27 c:/WIN95/WIN386.SWP
 -rw-rw-rw- 99999744 Aug 06 12:29 d:/BOOKS/DATABASE/ODD.NFO
 -rw-rw-rw- 94371840 Aug 22 00:12 d:/PAGEFILE.SYS
 -rw-rw-rw- 11249352 Aug 10 11:20 d:/PARK/ACC_NIR.ZIP
 -rw-rw-rw- 23705165 Aug 22 14:34 d:/PARK/ud_all.zip
 -rw-rw-rw- 35443990 Aug 22 00:59 d:/PARK/win95.zip
 -rw-rw-rw- 11386604 Dec 04 1996 d:/PARK/WW32_302.ZIP
 -rw-rw-rw- 32099034 Nov 29 1996 d:/PARK_FIX/MSOFFICE.ZIP
 -r--r--r-- 55871814 Jun 06 1996 d:/PARK_FIX/NT351_0.Z
 -rw-rw-rw- 12244344 May 27 1996 d:/PARK_FIX/VB40.ZIP
```

You can customize the threshold size we used here (-1m): 1,000,000 bytes to any size you like (i.e. -3m   used 3 megabytes).

3: Custom process each files in a list   (i.e. replace something in each file)

In this example we demonstrate how to process each line in a file list.
This example will first make a list of all your files in the current directory and then show you each name.
You can insert your custom process to apply to each file in the list
(i.e to replace use: sed "s:old:new:g" %NAME% > x1.tmp AND mv -lvf x1.tmp %NAME%)

The following script will get one line at a time until the file is exhausted (see test_lst.bat):

```
@echo off
set LINE=1
set LST=files.lst
rem ****** FILL THE FILES LIST ******
ls > files.lst
:LOOP
    rem ****** EXTRACT THE NEXT LINE ******
    getlines %LINE% %LINE% %LST% > name.tmp
    if errorlevel 1 goto END

    rem ****** SET AN ENV VARIABLE WITH THE CURRENT FILE NAME
****
    call setenv "NAME=@name.tmp"

    rem ****** HERE YOU CAN PLACE YOUR CUSTOM PROCESS ******
    echo %LST% LINE %LINE% = %NAME%

    rem ****** EXAMPLE FOR REPLACING old WITH new ******
    rem sed "s:old:new:g" %NAME% > new.tmp
    rem mv -lvf new.tmp %NAME%

    rem ****** INCREMENT THE LINE NUMBER ENV VARIABLE ******
    call setenv "LINE=`expr $LINE + 1`"
    goto LOOP
:END
del name.tmp
```

Might show something like:
```
files.lst  LINE 1 = autoexec.bat
files.lst  LINE 2 = command.com
files.lst  LINE 3 = config.sys
files.lst  LINE 4 = pagefiles.sys
files.lst  LINE 5 = windows ...
```

4: Find a text document anywhere on your machine using a   text search

In some cases when you have many documents(*.doc, *.txt etc) across many different directories you might forget where a particular document resides. With the following command you can easily find the document if you remember at least one fairly unique text string within that document (the case doesn't matter -i option)

```
c:
cd \
grep -d -l -i mystring *.doc *.txt *.hlp *.wri *.rtf
```

This will find "MyString" or "myString" or "MYSTRING" since the "-i" option will ignore the case.

Example:
```
c:
cd \
grep -d -l -I "unixdos license" *.doc *.txt *.hlp *.wri
```

Might show:
```
Cannot open '*.doc'
Cannot open '*.rtf'
Cannot open '*.wri'
ud/help/ud_gen.rtf
ud/help/ud_lic.rtf
ud/help/ud_tips.rtf
ud/help/unixdos.hlp
Unixdos/unixdos.hlp
```

The "cannot open" message simply indicate that GREP did not find any of these files in the root directory -nothing to worry about!

5: Get used Disk Space and Directory Overview

Sometimes you need to "step back" and get an overview of all the directories on your hard disk. The following command will produce a list of all the directories in alphabetical order and store it in a text file "all.du" in the root directory:

```
c:
cd \
du -o / > all.du
```

The output in 'all.du' might look like:
```
Megabytes (Cluster = 8192 bytes)
    5.6033 f:/Acrobat
    0.0082 f:/AOL30
    5.9228 f:/aolpress
    9.6911 f:/BITWARE
    0.0492 f:/DIGIDEMO
    0.0164 f:/ETC
    0.3441 f:/FTP
   31.3262 f:/instshld
    3.9895 f:/JAMWIN
  217.3419 f:/MSDEV
   18.4566 f:/MSVC32S
    6.5536 f:/PBWIN
    2.6214 f:/PMAIL
    3.9485 f:/PMAIL250
    2.6214 f:/RCDPC30
    0.0737 f:/RECYCLED
    ----   f:/TMP
    0.0328 f:/TYPESET
    4.5957 f:/VENTURA
   30.6954 f:/VisualCafe
  126.5009 f:/WIN311
    0.7373 f:/WS_FTP
    2.0480 f:/{Files}
  511.2955 f:/
```

Note that empty directories are show as '---' and in the overview mode <u>DU</u> will only show the directory totals, but not all the sub directories.

To see the sub directories use <u>DU</u> without the Overview option (-o):
```
du f:\aolpress
```

The output might look like:
```
Megabytes (Cluster = 8192 bytes)
 0.0082 f:/aolpress/AutoSave
 1.3517 f:/aolpress/HELP/refer
 0.9011 f:/aolpress/HELP/TUTORIAL
 2.4330 f:/aolpress/HELP
 0.0082 f:/aolpress/npcache
 5.9228 f:/aolpress
```

6: Cleanup temporary files

As a daily routine you might want to remove all the temporary files on your machine (over night for example).
Temporary files usually have the `*.tmp' file extension and reside in certain directories.
We will use the <u>UFIND</u> utility and execute <u>RM</u> for each temporary file found:
```
ufind / -name '*.tmp' -exec rm -lf {} ;            (or)
ufind / -name '*.tmp' -exec rm -lf {} ^
```

If you also have the habit of naming all my temporary work files starting with "xx", to also clean them up I use:
```
ufind / -name '*/xx*' -exec rm -lf {} ^
```

Here is a batch file which also cleans up the temporary directories::
```
@echo off
c:
cd \
rm -rfl "\win95\temp\*"
rm -rfl "\win95\Temporary Internet Files\*"
rm -fl "\tmp\*"
rm -fl "\temp\*"
rm -rfl "\recycled\*"
ufind / -name '*.tmp' -exec rm -lf {} ;
```

7: Find all files you worked on today

Sometimes will want to get a list of files you worked on today.
First we have to create a reference file of midnight of the current day and the do the search.
In this example we are searching the drives C:/D:/E:/F: - so you just replace this list with your list of drives.

The following batch file will do the job:
```
@echo off
rem ****** DEFINE THE REFERENCE FILENAME ******
set REF_FILE=c:\today.ref
rem ****** CREATE THE REFERENCE FILE ******
udate +%%m%%d%%n | sed "s:$:0000:" > today.tmp
touch @today.tmp %REF_FILE%
rm today.tmp
ls -l %REF_FILE%

rem ****** DO THE SEARCH ******
uecho Searching files worked on today ...
ufind c: d: e: f: -newer %REF_FILE% -ls > today.lst

rem ****** SHOW THE RESULT ******
umore today.lst
```

If you just want to see all the files you worked on within the last 24 hours use directly
<u>UFIND</u>:
```
ufind c: d: e: f: -mtime 24h -ls
```

8: Finding duplicate files on your machine

Often you will find many duplicate or identical files on your machine stored in different directories. This takes up extra space and can be problematic if you have the same two or more versions of the same program name stored in different directories since it will be accidental which program is called depending on the PATH and the current directory.

UnixDos offers a simple way to get a list of all the duplicate files on your machine.

The following batch file will do the job for drives c: and d:

```
@echo off
rem **** GET ALL FILES ****
set LINE=1
set LST=work5.tmp
set OUT=files.dup
uecho Collecting all names ...
ufind c:/ d:/ -keyname: -ls > work1.tmp
uecho Found "`wc -lt work1.tmp # cut -f1`" Files ...

rem **** SORT ALL FILES ****
uecho Sorting all names ...
usort work1.tmp -o work2.tmp
usort -u +0 -1 -t: work2.tmp -o work3.tmp

rem **** FIND DUPLICATES ****
uecho Finding Duplicates ...
diff -c1 work2.tmp work3.tmp > work4.tmp
grep "^+" work4.tmp | cut "-d|" -f2 | cut -f1 -d: > %LST%
wc -lt %LST% | cut -f1 > work7.tmp
uecho Found "@work7.tmp" duplicates  ...

rem **** GENERATE OUTPUT ****
rm -rf %OUT%
uecho Generating output in %OUT% ...
:LOOP
    getlines %LINE% %LINE% %LST% > name.tmp
    if errorlevel 1 goto END

    rem **** SHOW PROGRESS ****
    call setenv "NAME=@name.tmp"
    uecho -e %LINE% of "@work7.tmp" : %NAME%
    uecho "====================================" >> %OUT%

    rem **** EXTRACT DUPLICATES ****
    sed "s:\.:\\.:g" name.tmp  > work6.tmp
```

```
        sed "s:\$:\\$:g" work6.tmp > name.tmp
        sed "s:^:^:"      name.tmp  > work6.tmp
        sed "s/$/:/"      work6.tmp > name.tmp
        grep "@name.tmp" work2.tmp >> %OUT%

        call setenv "LINE=`expr $LINE + 1`"
        goto LOOP
:END
uecho "===================================" >> %OUT%

rem **** CLEANUP AND DISPLAY RESULT ****
rm work*.tmp
umore %OUT%
```

If you want to narrow the search only to executable (*.exe) files use:
```
ufind c:/ d:/ -name '*.exe' -keyname: -ls > work1.tmp
```

To get simply a sorted list by the filename enter:
```
ufind c:/ -keyname: -ls > all.lst
usort all.lst -o all.lst
```

9: Measure elapsed Time for (batch) process

Sometimes you might want to know how long a process or a batch of processes take.
The following command will reset the starting time
```
uptime -i
```

Then you can do your processing:
```
call proc.bat
```

To get the elapsed time just enter:
```
uptime
```

And it will display something like:
00:00:20 elapsed since reboot at:
 Fri Oct 25 11:02:56 1996

This means 20 seconds since the last "uptime -I"

10: Find the main directories from the root directory

To find all the main directories in your root directory enter:
c:
**cd **
ls -p | grep "/"

Might display:
/dos622
/msoffice
/Program Files
/SC
/windows

To find all the main directories with one level deep subdirectories in your root directory enter:
ufind c: -type d -level 2 -print

Might display:
c:/DOS622
c:/MSOFFICE/ACCESS
c:/MSOFFICE/CLIPART
c:/MSOFFICE/EXCEL
c:/MSOFFICE/POWERPNT
c:/MSOFFICE/SETUP
c:/MSOFFICE/WINWORD
c:/MSOFFICE
c:/Program Files/Accessories
c:/Program Files/Borland
c:/Program Files/Common Files
c:/Program Files/ICW-Internet Connection Wizard
c:/Program Files/Internet Explorer
c:/Program Files/Microsoft Exchange
c:/Program Files/Microsoft Internet
c:/Program Files/Symantec
c:/Program Files/The Microsoft Network
c:/Program Files
c:/SC/MSDOS6.22
c:/SC/WIN95
c:/SC/WIN_NT
c:/SC
c:/WINDOWS/Java
c:/WINDOWS/MSAPPS
c:/WINDOWS/MSSETUP
c:/WINDOWS/OLD
c:/WINDOWS/PFM
c:/WINDOWS/REPAIR
c:/WINDOWS/SYSTEM
c:/WINDOWS/SYSTEM32
c:/WINDOWS/UNINSTAL
c:/WINDOWS/VGAUTIL
c:/WINDOWS

c:/

11: List several numbers (or names) on one single line

In some cases you might have a text file containing a number or short text on each line. Now when you print this file you would waste a lot of space since you only place one item on each line while you could have much more (like 10) per line.

The following <u>PASTE</u> command will list five numbers (or names) per single line:
```
paste - - - - - < list.txt
```

<u>Might show:</u>

```
181    300    97 2142   221
267    234   288  190   117
33     195   263  295  1617
1547   627   284  605   426
1013    94    74  273   569
129    610   569  783     1
918    832   454  811   130
210    145    86 1194
```

<u>If "list.txt" contained:</u>
181
300
97
2142
221
267
234
288
190
117
33
195
263
295
1617
1547
627
284
605
426
1013
94
74
273
569

129
610
569
783
1
918
832
454
811
130
210
145
86
1194

12: User Prompt Example

The following batch file shows several examples how to prompt the user for input:
- <u>USRCHAR</u> get a single character without return
- <u>USRPRMPT</u> get a text from the user and write to `usrprmpt.tmp' (i.e. filename)
- <u>LINE</u> read one line and send it to standard output

Prompt the user for the files to copy with checks (see test_cp.bat):

```
@echo off
rem ****** CHECK IF COPY WANTED *************************
usrchar "Do you want to copy a file?" (Y/N)\c yn
IF ERRORLEVEL 2 GOTO END

rem ****** GET THE INPUT FILENAME ***********************
usrprmpt Enter the source filename:\c -r
call setenv "IN=@usrprmpt.tmp"

rem ****** CHECK IF THIS FILE EXISTS ********************
test -r "$IN" runneg: echo FILE %IN% DOES NOT EXIST
IF ERRORLEVEL 1 GOTO END

rem ****** GET THE OUTPUT FILENAME **********************
echo Enter the destination filename:
call setenv "OUT=`line`"

rem ****** CHECK IF IT IS WRITABLE IF IT EXISTS **********
test -r $OUT
IF ERRORLEVEL 1 GOTO DO_COPY
test -w $OUT runneg: echo CANNOT WRITE TO FILE %OUT%
IF ERRORLEVEL 1 GOTO END

rem ****** EXECUTE THE ACTUAL COPY **********************
:DO_COPY
cp -l %IN% %OUT%
goto END

:END
```

<u>Output example1:</u>
Do you want to copy a file? (Y/N) Y
Enter the source filename: blabla
FILE blabla DOES NOT EXIST

<u>Output example2:</u>
Do you want to copy a file? (Y/N) Y

```
Enter the source filename: x1.bat
Enter the destination filename: xnew
CANNOT WRITE TO FILE xnew
```

Output example3:

```
Do you want to copy a file? (Y/N) Y
Enter the source filename: x1.bat
Enter the destination filename: newfile
x1.bat -> newfile 1 files written successfully
```

13: Loop Example

In this example we demonstrate how to write a simple loop in a batch file.
We will print a table of the powers of two (see test_exp.bat):

```
@echo off
set EXP=1
set BASE=2
:LOOP
    echo 2 power %EXP% = %BASE%
    call setenv "EXP=`expr $EXP + 1`"
    call setenv "BASE=`expr $BASE x 2`"
    test $EXP -le 30
    IF NOT ERRORLEVEL 1 GOTO LOOP
```

Will print:

```
2 power 1 = 2
2 power 2 = 4
2 power 3 = 8
2 power 4 = 16
2 power 5 = 32
2 power 6 = 64
2 power 7 = 128
2 power 8 = 256
2 power 9 = 512
2 power 10 = 1024
2 power 11 = 2048
2 power 12 = 4096
2 power 13 = 8192
2 power 14 = 16384
2 power 15 = 32768
2 power 16 = 65536
2 power 17 = 131072
2 power 18 = 262144
2 power 19 = 524288
2 power 20 = 1048576
2 power 21 = 2097152
2 power 22 = 4194304
2 power 23 = 8388608
2 power 24 = 16777216
2 power 25 = 33554432
2 power 26 = 67108864
2 power 27 = 134217728
2 power 28 = 268435456
2 power 29 = 536870912
2 power 30 = 1073741824
```

14: View many small files together (i.e. join all your batch files)

If you have many small files and want to print them you can either print one page per file or concatenate all files into one file, but loose the filename.
If you could insert in front of each line the filename you could easily find any information in one file.
In the following example we create a list of files of your zip files in files with extension *.zls.
This assumes that you already have created a ".ZLS" file for each ZIP file:
(i.e pkunzip -vn WIN.ZIP > WIN.ZLS).
Then you can generate one file containing a list of all the files within your ZIP files with:
```
grep -d -a "." *.zls > all_zip.txt
```


In the following example we will generate a single file with all the batch files on your C: drive:
```
c:
cd \
grep -d -a "." *.bat > all_bat.txt
umore all_bat.txt
```

Might show:

AUTOEXEC.BAT: rem C:\WIN95\net start
AUTOEXEC.BAT: @echo off
AUTOEXEC.BAT: verify on
AUTOEXEC.BAT: break on
AUTOEXEC.BAT: prompt $t pg
WIDGETS/samples/xttest/vc20/makeall.bat: call set_ec
WIDGETS/samples/xttest/vc20/makeall.bat: call clean
WIDGETS/samples/xttest/vc20/makeall.bat: cd debug
WIDGETS/samples/xttest/vc20/makeall.bat: call clean
WIDGETS/samples/xttest/vc20/makeall.bat: cd ..
WIDGETS/samples/xttest/vc20/makeall.bat: nmake /nologo DEBUG=1 -f xttest_w.mak
WIDGETS/samples/xttest/vc20/mkd.bat: nmake DEBUG=1 -f xttest_w.mak
WIN95/help/pingname.bat: ping www.microsoft.com
WIN95/help/pingname.bat: ping ftp.microsoft.com
WIN95/help/pingname.bat: @echo off
WIN95/help/pingname.bat: echo ...
WIN95/help/pingname.bat: echo Click the Help window to continue with the Internet troubleshooter.
WIN95/help/pingnum.bat: ping 198.105.232.1
WIN95/help/pingnum.bat: ping 198.105.232.6
WIN95/help/pingnum.bat: @echo off
WIN95/help/pingnum.bat: echo ...
WIN95/help/pingnum.bat: echo Click the Help window to continue with the Internet troubleshooter.
WIN95/save.bat: c:

```
WIN95/save.bat: cd \win95
WIN95/save.bat: rm -rfl d:\park\win95.zip
WIN95/save.bat: pkzip -pr d:\park\win95
WIN95/save.bat: ls -l d:\park\win*.zip
WINDOWS/d2n.bat: @echo off
WINDOWS/d2n.bat: switch N
WINDOWS/get_osnt.bat: cmd /c ver &gt; c:\ver.tmp
WINDOWS/n2d.bat: @echo off
WINDOWS/n2d.bat: switch D
WINDOWS/oled2n.bat: @echo off
WINDOWS/oled2n.bat: oleswtch N
WINDOWS/olen2d.bat: @echo off
WINDOWS/olen2d.bat: oleswtch D
WINDOWS/oleswtch.bat: @echo off
WINDOWS/tmpdelis.bat: @if exist C:\WINDOWS\tmpcpyis.bat del C:
\WINDOWS\tmpcpyis.bat
WINDOWS/tmpdelis.bat: @if exist C:\WINDOWS\winstart.bat C:
\WINDOWS\winstart.bat
```

If the batch files are all in one directory you an also use:
pr -X: -t *.bat > all_bat.txt

15: Return to your home directory from a batch file:

In this example we demonstrate how to save the initial home directory before a batch process starts and changes directories and the RETURN back to the original directory.

```
rem **** save initial home directory ****
pwd > %TEMP%/pwd.tmp
rem ****** change to other directories and make a backup ****
cd \src
pkzip -pr \park\src

rem ****** return to the initial home directory ******
ucd @%TEMP%/pwd.tmp
```

This example uses the UCD UnixDos utility to change directories which is able to convert the contents of a file into a regular argument - thereby allowing you to use the start directory as an argument to return back to:
(see FileList in UnixDos Argument Expansion Extensions):

Might show:
```
C:\SRC\WORK>rem **** save initial home directory ****
C:\SRC\WORK>pwd 1>c:\temp/pwd.tmp
C:\SRC\WORK>rem ****** change to other directories and make a backup ****
:\SRC\WORK>cd \src
C:\src>rem pkzip -pr \park\src
C:\src>cd \tmp
C:\tmp>rem ****** return to the initial home directory ******
C:\tmp>ucd @c:\temp/pwd.tmp
C:\SRC\WORK>
```

16: Find the location of a program

Sometimes you might need to know where a specific batch file or program is located. The PATH environment contains several directories of all your program on your machine.
It would be very tedious to search manually for a program in each of these directories. UNIXDOS provides this capability with the <u>WHICH</u> command:
which mybatch

Will search all directories in the PATH environment for any programs with the filename "mybatch".
The extension can be ".bat", ".com" or ".exe".

<u>Might show:</u>
c:/Unixdos/mybatch.bat

To alter the default list of "programs" set your own list of possible file extensions:
set UDPATHEXT=.exe;.com;.bat;.pif;.sh;

Now you can find also PIF files:
which dosprmpt

<u>Might show:</u>
c:/Windows/dosprmpt.pif

17: Break a line with fields into several lines

In some cases you might want to break a line which is separated by a field delimiter into several separate lines.
For example the PATH environment contains a list of all directories with programs and batch files etc

```
PATH=F:\MSDEV\BIN;C:\UTIL;C:\WIN95;C:\WIN95\COMMAND
```

To get a list of directories line by line enter:

```
sed "s/;/\n/g" ~$PATH
```

<u>Might show:</u>

```
F:\MSDEV\BIN
C:\UTIL
C:\WIN95
C:\WIN95\COMMAND
```

If the "fields.txt" contains:

```
field1-1,2,a,more text1
field1-2,3,c,more text2
field1-3,6,b,more text3
field1-4,10,d,more text4
```

To replace the commas with a new line enter:

```
sed "s:,:\n:g" bed1.txt
```

The output is:

```
field1-1
2
a
more text1
field1-2
3
c
more text2
field1-3
6
b
more text3
field1-4
10
d
more text4
```

Introduction to UnixDos Reference Guide

In this "UnixDos Reference Guide" you will find a detailed description of each UnixDos utility in alphabetical order. You will also find here many examples and notes.
Please send any questions and suggestion around UnixDos to:
UnixDos@pobox.com

To see an alphabetical list of all 92 UnixDos utilities click here:
Complete alphabetical UnixDos List

The documentation for the TXTBOX utility is here.

Below is a basic description of the format used foe each UnixDos utility:

Intro (Introduction)

Syntax:
```
command [optional options] {required arguments} {choice1|
choice2}
```

In the "Syntax" paragraph you will see short summary of all the options which are available in this utility. Below the summary (or "usage" information) you will see a short description of each of the presented choices.

`{...}`   *Required:*   Arguments which are required are enclosed in the curly brackets "{}"

`[...]`   *Optional:*   Arguments which are optional are enclosed in the square brackets "[]"

             Alternatives: List of several mutually exclusive alternative choices
```
choice1|choice2|more choices...
```

Option Summary:
```
[options] option description
{args}    argument description
```

Description:
In the "Description" paragraph you will find a detailed description about the functionality of this utility.

Example:
The "Example" paragraph you will find actual examples and applications of this utility

together with the output these example will generate. This is a good way to get "warmed up"
with the functionality of the utilities.
The command lines as you would type it is `printed in a fixed font`, while the
output which is generated by the utility is *`printed in a fixed italic font.`*

Notes:
Here you will find additional information alerting you of limits or problematic situations regarding the current utility or differences between the UnixDos implementation and "Classic UNIX".

Remarks:
Here you will find little anecdotes and interesting tidbits concerning this utility.

add (Add numbers)

<u>**Syntax**</u>
```
add [-d{del}] [-f{fld}] [-l{len}] [-m{mod}] [-p{p}] [file(s)...]
-d{del}    = del   : use delimiter {del} (default is blank)
-f{fld}    = field : use field number {fld} (default 1=first)
-l{len}    = length: use display length {l} for numbers (default is 8)
-p{p}      = prec  : use precision of {p} for numbers (default is 2)
-m{mod}    = modulo: display result every {mod} lines (default is 1,
0=none)
file(s)    = input file(s) (if no files use STDIN)
```

<u>**Description**</u>
ADD will add and analyze numbers. It will provide the following information:
- Sum Total
- Minimum
- Maximum
- Average
- Number of Values entered

You can specify which field and which delimiter is to be used (-f and -d option).
By default ADD will show the intermediate result after each input line - you can specify a different modulo factor or 0 to switch the intermediate display off (-m option).
By default ADD will display each value 8 digits wide and 2 numbers after the decimal point (precision), but you can change those default settings (-l, -p option).

<u>***Example:***</u>
If file "add1.txt" contains:
1,3
2,2.54
3,1.18
4,10
5,1.36

To add the numbers in the second column after the commas enter:
```
add -f2 -d, add1.txt
```

Will show:
```
1. Value     3.00 ,Sum=    3.00, Min=    3.00, Max=    3.00
2. Value     2.54 ,Sum=    5.54, Min=    2.54, Max=    3.00
3. Value     1.18 ,Sum=    6.72, Min=    1.18, Max=    3.00
4. Value    10.00 ,Sum=   16.72, Min=    1.18, Max=   10.00
5. Value     1.36 ,Sum=   18.08, Min=    1.18, Max=   10.00
5. Values    1.36 ,Sum=   18.08, Min=    1.18, Max=   10.00, Avg=    3.62
```

To switch the intermediate result display off enter:
```
add -f2 -d, -m0 add1.txt
```
And you will only see the summarization:
```
5. Values    1.36 ,Sum=   18.08, Min=    1.18, Max=   10.00, Avg=    3.62
```

af (Analyze file)

Syntax
```
af {input file}
```

Description
AF will analyze the specified file (text or binary) and print a statistical overview of the count of each possible byte value (0-255 or 0x00 - 0xFF).
This can help you to spot invalid characters in text files (see "Example" below).

Example
```
af af.exe
```

Will print the following result:

```
Analyzing file '/autoexec.bat' ....
1643 bytes read
```

CODE:	0 / 8	1 / 9	2 / A	3 / B	4 / C	5 / D	6 / E	7 / F
0x00:								
0x08:			65			65		
0x10:								
0x18:								
0x20:	103				7			
0x28:			51	1	5	3	12	19
0x30:	17	10	7	4	3	2	5	4
0x38:	9	2	65	19		27		
0x40:	1	20	9	42	28	43	7	7
0x48:	18	31		1	41	36	30	24
0x50:	21	1	18	33	32	15	15	8
0x58:	6	3	1		92			3
0x60:		24	16	58	32	63	11	6
0x68:	10	23	2	7	20	46	14	22
0x70:	18		32	51	37	22	24	9
0x78:	3	1						
0x80:								
0x88:								
0x90:								
0x98:								
0xA0:								
0xA8:								
0xB0:								
0xB8:								
0xC0:								
0xC8:								
0xD0:								
0xD8:								
0xE0:						*1*		
0xE8:								

0xF0:
0xF8:

This example shows for example that 'af.exe' contains 65 Linefeeds (0xA) and 65 carriage returns (0xD) in other words it has currently 65 lines.
Notice one "dirty" non-printable character at 0xE5 which should not be in a pure text file.

Remarks

I used this utility on many occasions to if a plain ASCII text files actually contain only printable characters (range 0x20 thru 0x7E) and if the counts for carriage return "CR" (0xD) is matching that of "line feed" LF (0xA).
You could also use it to detect is double quotes or commas actually appear in a database dump, posing probable problems when you use the same character as a field delimiter:

`First Name<tab>Middle Name<tab>O"Neil<tab>` Notice the problem in "FirstName","O Neil".

b64 (Decode Base64 Internet Text Files)

Syntax:
```
b64 (Decode Base64 Text)
usage: b64 {file}
```

Description:
In some cases when you receive e-mail messages over the INTERNET the mail program is not recognizing the attached binary file and will just place the data within your e-mail message.
B64 allows you to decode the textual \data back into the original binary form.

B64 will read the specified text file which contains b64 encoded data.
The following syntax is used to detect the data area(s):
```
---- next item ----
Content-Type: ....; Name="output1.dat"
Content-Transfer-Encoding: Base64

UEsDBBQAAAAIAAAAdxj3C183084AAErYAAALAAAAX01TVEVVTVC5FWF+8u/dfU9kb
LbxzUggpECKCIiUgoCOiEdCRamhBR8XQgkqRoIyIoggJ6CgQBEs4gBGd0WHECZZR
UZyA2AcJRbAMmmBXkYCowYiEIqFm3zPf+34+9z94f9nsdXLOyTbu8jxrrSdifVDQ
Id0cd38gi8MBuHr9WkABAEyxvusgNAIA55LE+a4DeByOgGPQERtjgg3dl76aHkdP
..... (more data)

---- next item ------
Content-Type: ....; Name="output2.dat"
Content-Transfer-Encoding: Base64

UEsDBBQAAAAIAAAAdxj3C183084AAErYAAALAAAAX01TVEVVTVC5FWF+8u/dfU9kb
LbxzUggpECKCIiUgoCOiEdCRamhBR8XQgkqRoIyIoggJ6CgQBEs4gBGd0WHECZZR
UZyA2AcJRbAMmmBXkYCowYiEIqFm3zPf+34+9z94f9nsdXLOyTbu8jxrrSdifVDQ
Id0cd38gi8MBuHr9WkABAEyxvusgNAIA55LE+a4DeByOgGPQERtjgg3dl76aHkdP
..... (more data)

---- next item ------
```

B64 will automatically recreate the original binary files (in this example output1.dat, output2.dat)

Example:
```
b64 message1.msg
```

Might display the following output:
```
b64: Writing data to 'output1.dat' (Line 44) ...
b64: Writing data to 'output2.dat' (Line 1048) ...
```

banner (Print text in big letters horizontally)

Syntax:
```
banner [-c{c}]{text1} [text2...]

-c{c}      Use character 'c' for printing (default is #)
text       text to be display as a banner (up to 10 per line)
```

Description:
BANNER will display the specified text element 'text1...' in big letters horizontally.
Each text element can be up to 10 characters long. Longer text will just be truncated -
not wrapped around to the next line. If you want to include blanks within the text use the
double quotes around your text so that the system passes the text with blanks as one
single argument: `banner "with blanks"`

Example:
```
banner `udate +%D` `udate +%T`
```

Will display the current date like:
```
  ###     #                 #    ###       #              #  #####      #
 #   #  #   #   #        #   #   #   #     ##            #  #         #   ##
#     # #   #   #         #  #     # # #            #  #       #   # #
#       # #   #          #   #     # #        #      ######     #
#       # #######   #     #     #    #        #            #    #
 #   #        #   #       #   #    #     #      #      #      #
  ###        # #          ###    #####  #        #####    #####

   #     #####      #      #    #####      #       ###    #####
  ##     #    #    ###    ##    #    #    ###     #   #   #     #
 # #     #    #    #      # #   #    #    #     #      # #   #     #
   #     #####         #       #####         #     # ######
   #     #    #    #    #    #    #    #    #     # #     #
   #     #    #   ###    #    #    #    ###    #    # #     #
 #####   #####      #    #####  #####     #       ###    #####
```

Remarks:
This is a great utility to insert big headers into printouts or lengthy batch processes to
indicate the start of a new step or the completion of a process.

bbanner (Print text in very big letters vertically)

<u>Syntax:</u>

```
bbanner [-c{c}] [-l[d]] [-w{n}] {text1} [text2...]

-c{c}      CHAR: use 'c' as the display character (default is #)
-l[d]      LOOP: after text is printed stop for 'd' seconds
           and repeat the same text again
-w{n}      WIDTH: set the horizontal width to 'n' (default is 80)
```

<u>Description:</u>
BBANNER will display the specified text element 'text1...' in very big letters vertically.

<u>*Example:*</u>

```
bbanner -c@ -l2 Very BIG
```

Will continually display the message 'SYSTEM IS IDLE' with 2 seconds
pause using the   character '@' instead of   the default '#':

```
                                                @@@
                                                @@@
                                                @@@
                                                @@@@
                                           @@@@@@@@@@
                                         @@@@@@@@@@@@@@
                                       @@@@@@@@@@@@@@@@@@
                                     @@@@@@@@@@@@@@@@@@@@@@
                                   @@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                 @@@@@@@@@@@@@@@@@@@@@@@@@@    @@@
                             @@@@@@@@@@@@@@@@@@@@@@@@@@@        @@@
                           @@@@@@@@@@@@@@@@@@@@@@@@@@
                         @@@@@@@@@@@@@@@@@@@@@@@@@@
                       @@@@@@@@@@@@@@@@@@@@@@@@@
                     @@@@@@@@@@@@@@@@@@@@@@@@
                   @@@@@@@@@@@@@@@@@@@@@@
                     @@@@@@@@@@@
                       @@@@@@@@@@
                         @@@@@@@@@
                           @@@@@@@@@@
                             @@@@@@@@@
                               @@@@@@@@@@
                                 @@@@@@@@@@      @@@
                                   @@@@@@@@@@   @@@
                                     @@@@@@@@@@
                                       @@@@
                                       @@@
                                       @@@

               @@@@@@@@@@
             @@@@@@@@@@@@@@@@@@
           @@@@@@@@@@@@@@@@@@@@@@@@
         @@@@@@@@@@@@@@@@@@@@@@@@@@@@
       @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
         @@@@@       @@          @@@@@
         @@@@        @@           @@@@
         @@@         @@            @@@
         @@          @@             @@
         @@          @@             @@
         @@          @@             @@
         @@          @@            @@@
         @@@         @@            @@@@
```

basename (Extract filename+extension)

Syntax:
```
basename [-d] [file(s)]

-d        dos: convert to DOS style (default is UNIX style)
file(s)   filenames to be converted.
```
If no filenames are given BASENAME will read STDIN and convert each input line.

Description:
BASENAME will extract the filename and extension from the filelist.

You can either specify one or more filenames on the command line or
provide a list of filenames in STDIN if you don't specify any filename.

Example:
```
basename \msvc\include\windows.h
```
Will show:
```
windows.h
```

```
basename -d \msvc\include\windows.h
```
Will show the basename is DOS style:
```
WINDOWS.H
```

To display all filenames in DOS style:
```
ufind / -exec basename -d {} ^
```

bed (Binary steam editor)

```
bed {s:old:new:} {file}
s:old:new:      substitute: any match of 'old' with 'new'
                The pattern have to have the same length in
bytes.
                Use \XX to specify hex values.
file            Binary file to operate on
```

Description:
BED will read the binary 'file' and search for the specified pattern in 'old'.
If BED finds a match it will replace it with new pattern specified in 'new'.
BED will write the modified file to STDOUT, use the redirection to catch the modified file.

Example:
The CD(Change Directory) command is hardcoded in the COMMAND.COM DOS shell.
In order to replace the internal CD command with the UnixDos (u)cd utility when you
type 'cd' on the command line we have to disable the internal CD within
COMMAND.COM.
BED can do the job:

change to the root directory   and do the replacement:
```
ucd c:/
bed s:CD:XD: command.com > xnew
```

save old command.com:
```
mv command.com cmd_old.com
```

plug in the new command.com:
```
mv xnew command.com
```

change to the directory where all the UnixDos utilities are located:
```
ucd /put
```

make the ucd.exe available also as 'cd.exe':
```
cp ucd.exe cd.exe
```

reboot...and try the new cd:
```
cd f:/login
```
Will go to:
F:\LOGIN

Remarks:
This utility is excellent to perform last minute patching off executables and other binary

files instead of having to rebuild because of on minor text change; but remember to change the source code as well!

bell (Ring bell)

Syntax:
```
bell [n]

n     number of rings or beeps (default is 1)
```

Description:
BELL will send ring(s) to STDOUT by using the ASCII control character 7.
You can specify more than one ring with the first argument on the command line.

Example:
Send 10 rings to the printer:

```
bell 10 >> prn
```

bgrep (Search pattern in binary file)

Syntax:
```
bgrep {pattern} {input} [start] [length]

pattern   = text or string to be found (\XX = hex code)
            (Example: \01\00 = short value of 1: 0x01,0x00)
input     = input file
start     = offset to start reading (default 0)
length    = number of bytes to read (default all until end of
file)
```

Description:
BGREP will read the binary 'file' and search for the specified pattern.
BGREP will list all matches with the corresponding file decimal offset.
By default the search starts at the beginning of the file (offset=0); but you can specify
any offset with the 'start' argument.
You can also limit the search by specifying how many bytes the search should do from
the starting offset (default is until the end of the file).

Example:
Search for 'CD' in COMMAND.COM:

```
bgrep CD /COMMAND.COM
```

Will print (under DOS 6.22):
```
Hit    1 at offset      39801
Hit    2 at offset      39820
Hit    3 at offset      39905
Hit    4 at offset      39978
Hit    5 at offset      49575
54645 bytes read from '/command.com' (starting at offset 0)
5 Hits found
```

Remarks:
This is a fast way to find text or any data within binary non-text files. I wrote this utility
because the main search engine "GREP" works only on text files. To view the actual
context of each occurrence use the UnixDos Binary Viewer (bvi) (see below).

bsplit (Split binary file into chunks)

Syntax:
```
bsplit -{n} file  [out]
-{n}  : minimum number of bytes for each chunk
file  : file to be split
out   : prefix - up to 5 characters (default is 'data')
```

Description:
BSPLIT will read the binary input file and writes this file in chunks of specified bytes into the a set of output files until the entire input file has been read. The first output file has the name 'data001', the second 'data002' and so on until 'data999'.
You can change the output prefix from 'data' to any letter sequence up to 5 character.

Example:
File `'bin.dat'` contains 275,000 bytes.
You want to split it in 1000,000 bytes chunks with prefix 'new':
```
bsplit -100000 bin.dat new
```

Will show:
```
bin.dat  -> new001 (100000 bytes)
bin.dat  -> new002 (100000 bytes)
bin.dat  -> new003 (75000 bytes)
```

Note that the last chunk 'new003' contains no exactly 100,000 bytes but just the remainder of the binary file.

bvi   (Binary interactive Viewer)

Syntax:
```
usage: bvi {file} [len] [off]
file    = file to be examined
len     = record length (default is 128 bytes)
off     = offset within file (default is 0)
```
If the record length of specified as 0, BVI will display the entire file in 100 byte chunks.

Description:
BVI allows you to view any file (binary or text).
If you don't specify the record length BVI will assume the default of 128 bytes.
If you don't specify any offset BVI will assume the beginning of the file (offset 0).
BVI will initially display a header with the filename, the total number of bytes in that file, the number of records based on the current record length and the record length:
```
File 'bvi.exe': 69558 bytes, 543 records with 128 bytes
```

BVI will display the following items for each record:
- record number (zero based in decimal unless switched to hex)
- record offset of the current line (zero based in 16 byte increments)
- 16 bytes shown as hexadecimal values
- 16 bytes shown a text characters (non-printable char shown as dots '.')

Example:
```
File 'bvi.exe': 69558 bytes, 543 records with 128 bytes
    0/  0: 4D 5A B6 01 88 00 CA 02 C0 00 B8 04 FF FF B3 13 MZ..............
    0/ 16: 00 14 00 00 12 00 51 06 1E 00 00 00 01 00 FE 02 ......Q.........
    0/ 32: 00 00 D6 02 00 00 B5 02 00 00 97 02 00 00 3F 02 ..............?.
    0/ 48: 00 00 17 02 00 00 F1 01 00 00 E1 01 00 00 D7 01 ................
    0/ 64: 00 00 A7 01 00 00 87 01 00 00 5C 01 00 00 48 01 ..........\...H.
    0/ 80: 00 00 3E 01 00 00 04 01 00 00 E5 00 00 00 D4 00 ..>.............
    0/ 96: 00 00 CD 00 00 00 B9 00 00 00 AC 00 00 00 A2 00 ................
    0/112: 00 00 93 00 00 00 89 00 00 00 79 00 00 00 6A 00 ..........y...j.
```

BVI will display the following prompt after each record display:
```
    ext,(P)rev,(G)oto,(F)ind,(E)xit,(M)ode,(Q)uit,(S)tring,(H)ex:
```

If the file is read-only the prompt will take out the update functions and show "READ-ONLY":
```
    ext,(P)rev,(G)oto,(F)ind,(E)xit,(M)ode,(Q)uit (READ-ONLY):
```

BVI Commands:
Commands can be entered in upper or lower case.
Next (Enter or N)   goto next record (default if ENTER or any other key is entered)

Previous (P or p)   goto previous record (if at the beginning wrap around to the last)

Goto (G[n])   jump to specified record 'n' (if 'n' is not specified you will be prompted)

Find (F)   search the file for pattern, you will be asked to answer the following
prompts:

```
Enter decimal Start Record (<0> =  first):
```
(default=0)
```
Enter decimal Stop  Record (Ret =     543):
```
(default = all remaining
records)
```
Enter the Search pattern (max 81):
```
Here you can enter straight text or binary values using \XX for hex values
Example: `END\0D\0A` will search for text "END" followed by CR(0x0D) +
LF(0x0A)

Exit (E or Q)   exit BVI

Mode(M{m})   switch to one of the following modes for displaying record
number+offset:

              md   decimal (default)
              mh   hexadecimal
              mo   octal

Update Text (S)   change text within the current record, you will be prompted for the
following:
```
Enter the start offset (0-127):
```
(specify the start point of your update
decimal)
```
Enter the end  offset (0-127):
```
(specify the end point of your update
decimal)
```
Offset  0, ('M') enter new character(s) (Return = no change):
```
Here you can enter the new text, either bytes by byte or you can enter it in
one   single chunk. If the chunk is longer than the specified number of
bytes BVI will   ignore the excessive bytes. Once you have completed
the new text entry BVI will
              show you the new record and ask for your confirmation before it actually
writes
              your update to the file and therefore commits the change:
```
Do you want to write to the disk? (Y/   ):
```
If you hit ENTER or not Y or y the last change will be thrown away.
After you confirmed the change by typing 'Y' BVI will replay with:
```
128 bytes written to disk
```
(In these examples the default record length of 128 bytes was used)

Update Data(H)   change data within the current record.
              You will be prompted for the same information as in the "Update
Text",
              except you enter hexadecimal values for each byte:
```
Offset 000, (0x41) enter new Hex value (Return = no change): AA
```

Example:
Examine BVI itself:

```
ucd \put
bvi bvi.exe


File 'bvi.exe': 69558 bytes, 543 records with 128 bytes
    0/  0: 4D 5A B6 01 88 00 CA 02 C0 00 B8 04 FF FF B3 13  MZ..............
    0/ 16: 00 14 00 00 12 00 51 06 1E 00 00 00 01 00 FE 02  ......Q.........
    0/ 32: 00 00 D6 02 00 00 B5 02 00 00 97 02 00 00 3F 02  ..............?.
    0/ 48: 00 00 17 02 00 00 F1 01 00 00 E1 01 00 00 D7 01  ................
    0/ 64: 00 00 A7 01 00 00 87 01 00 00 5C 01 00 00 48 01  ..........\...H.
    0/ 80: 00 00 3E 01 00 00 04 01 00 00 E5 00 00 00 D4 00  ..>.............
    0/ 96: 00 00 CD 00 00 00 B9 00 00 00 AC 00 00 00 A2 00  ................
    0/112: 00 00 93 00 00 00 89 00 00 00 79 00 00 00 6A 00  ..........y...j.
  ext,(P)rev,(G)oto,(F)ind,(E)xit,(M)ode,(Q)uit,(S)tring,(H)ex: g488 (go to
record 488)
  488/  0: 20 25 73 3C 20 72 65 74 20 25 6C 64 0A 00 25 73   %s< ret %ld..%s
  488/ 16: 20 28 42 69 6E 61 72 79 20 46 69 6C 65 20 56 69   (Binary File Vi
  488/ 32: 65 77 65 72 29 0A 00 75 73 61 67 65 3A 20 25 73  ewer)..usage: %s
  488/ 48: 20 7B 66 69 6C 65 7D 20 5B 6C 65 6E 5D 20 5B 6F   {file} [len] [o
  488/ 64: 66 66 5D 0A 00 66 69 6C 65 20 20 20 20 3D 20 66  ff]..file    = f
  488/ 80: 69 6C 65 20 74 6F 20 62 65 20 65 78 61 6D 69 6E  ile to be examin
  488/ 96: 65 64 0A 00 6C 65 6E 20 20 20 20 20 3D 20 72 65  ed..len     = re
  488/112: 63 6F 72 64 20 6C 65 6E 67 74 68 20 28 64 65 66  cord length (def
  ext,(P)rev,(G)oto,(F)ind,(E)xit,(M)ode,(Q)uit,(S)tring,(H)ex: s (Update Text)
Enter the start offset (0 – 127): 39  (start at decimal offset 39 'u')
Enter the end   offset (0 – 127): 43  (end at decimal offset 43 'e')
Offset  39, ('u') enter new character(s) (Return = no change): USAGE (new text)
  1e8/  0: 20 25 73 3C 20 72 65 74 20 25 6C 64 0A 00 25 73   %s< ret %ld..%s
  1e8/ 10: 20 28 42 69 6E 61 72 79 20 46 69 6C 65 20 56 69   (Binary File Vi
  1e8/ 20: 65 77 65 72 29 0A 00 55 53 41 47 45 3A 20 25 73  ewer)..USAGE: %s
  1e8/ 30: 20 7B 66 69 6C 65 7D 20 5B 6C 65 6E 5D 20 5B 6F   {file} [len] [o
  1e8/ 40: 66 66 5D 0A 00 66 69 6C 65 20 20 20 20 3D 20 66  ff]..file    = f
  1e8/ 50: 69 6C 65 20 74 6F 20 62 65 20 65 78 61 6D 69 6E  ile to be examin
  1e8/ 60: 65 64 0A 00 6C 65 6E 20 20 20 20 20 3D 20 72 65  ed..len     = re
  1e8/ 70: 63 6F 72 64 20 6C 65 6E 67 74 68 20 28 64 65 66  cord length (def
Do you want to write to the disk? (Y/   ): (confirmation)
  ext,(P)rev,(G)oto,(F)ind,(E)xit,(M)ode,(Q)uit,(S)tring,(H)ex: mh (switch to
hex mode)
  1e8/  0: 20 25 73 3C 20 72 65 74 20 25 6C 64 0A 00 25 73   %s< ret %ld..%s
  1e8/ 10: 20 28 42 69 6E 61 72 79 20 46 69 6C 65 20 56 69   (Binary File Vi
  1e8/ 20: 65 77 65 72 29 0A 00 75 73 61 67 65 3A 20 25 73  ewer)..usage: %s
  1e8/ 30: 20 7B 66 69 6C 65 7D 20 5B 6C 65 6E 5D 20 5B 6F   {file} [len] [o
  1e8/ 40: 66 66 5D 0A 00 66 69 6C 65 20 20 20 20 3D 20 66  ff]..file    = f
  1e8/ 50: 69 6C 65 20 74 6F 20 62 65 20 65 78 61 6D 69 6E  ile to be examin
  1e8/ 60: 65 64 0A 00 6C 65 6E 20 20 20 20 20 3D 20 72 65  ed..len     = re
  1e8/ 70: 63 6F 72 64 20 6C 65 6E 67 74 68 20 28 64 65 66  cord length (def
```

Remarks:
No wrap around search.
This is an excellent workhorse when you have to examine any binary file.
The search for text or data will miss those occasions when the pattern spawns across record boundaries. To double check use another record size and compare the search result or use BGREP to find all occurrences.

cal (Print calendar)

<u>**Syntax:**</u>
```
cal [ [month] year ]

month    = specify the month (defaults to current)
year     = specify the year  (defaults to current)
```
The Year can be entered as 1989 or 89

<u>**Description:**</u>
CAL will display the current month by default.
If you specify one single number CAL will use this number as the year and show the entire year.
You can specify the full year value or just the last two digits (96 is equivalent to 1996).
If you specify two numbers CAL will use the first for the month (1-12) and then the second for the year.

Example:
Display the current month:
```
cal
```

Might show:
```
   September 1970
 S  M Tu  W Th  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Display the October in 1963:
```
cal 10 1963
```

Will show:
```
   October 1963
 S  M Tu  W Th  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

Display the entire year 1969::
```
cal 69
```

Will show:
```
               1969
```

```
        Jan                      Feb                        Mar
 S   M Tu   W Th   F   S    S   M Tu   W Th   F   S    S   M Tu   W Th   F   S
         1   2   3   4                          1                              1
 5   6   7   8   9  10  11    2   3   4   5   6   7   8    2   3   4   5   6   7   8
12  13  14  15  16  17  18    9  10  11  12  13  14  15    9  10  11  12  13  14  15
19  20  21  22  23  24  25   16  17  18  19  20  21  22   16  17  18  19  20  21  22
26  27  28  29  30  31       23  24  25  26  27  28       23  24  25  26  27  28  29
                                                          30  31

        Apr                      May                        Jun
 S   M Tu   W Th   F   S    S   M Tu   W Th   F   S    S   M Tu   W Th   F   S
         1   2   3   4   5                  1   2   3    1   2   3   4   5   6   7
 6   7   8   9  10  11  12    4   5   6   7   8   9  10    8   9  10  11  12  13  14
13  14  15  16  17  18  19   11  12  13  14  15  16  17   15  16  17  18  19  20  21
20  21  22  23  24  25  26   18  19  20  21  22  23  24   22  23  24  25  26  27  28
27  28  29  30               25  26  27  28  29  30  31   29  30

        Jul                      Aug                        Sep
 S   M Tu   W Th   F   S    S   M Tu   W Th   F   S    S   M Tu   W Th   F   S
         1   2   3   4   5                      1   2        1   2   3   4   5   6
 6   7   8   9  10  11  12    3   4   5   6   7   8   9    7   8   9  10  11  12  13
13  14  15  16  17  18  19   10  11  12  13  14  15  16   14  15  16  17  18  19  20
20  21  22  23  24  25  26   17  18  19  20  21  22  23   21  22  23  24  25  26  27
27  28  29  30  31           24  25  26  27  28  29  30   28  29  30
                             31

        Oct                      Nov                        Dec
 S   M Tu   W Th   F   S    S   M Tu   W Th   F   S    S   M Tu   W Th   F   S
         1   2   3   4                          1            1   2   3   4   5   6
 5   6   7   8   9  10  11    2   3   4   5   6   7   8    7   8   9  10  11  12  13
12  13  14  15  16  17  18    9  10  11  12  13  14  15   14  15  16  17  18  19  20
19  20  21  22  23  24  25   16  17  18  19  20  21  22   21  22  23  24  25  26  27
26  27  28  29  30  31       23  24  25  26  27  28  29   28  29  30  31
                             30
```

cat   (Display and concatenate files)

```
cat [-estv] [-|file] ...

-e      = end    : show end of line as '$' (with -v option)
-s      = silent : don't print error messages
-t      = tab    : show TAB character as '^I' (with -v option)
-v      = verbose: show non-printable characters:
                   'M-' if bit7 is set (M = meta character)
                    ^X  Control-X
```

Description:
CAT   will read each specified file or files and send them all in one stream to the standard output STDOUT.
If no files are specified or '-' it will read the STDIN.
With CAT you can conCATinate separate files into one single file.

Example:
concatenate file1 and file2 (in that sequence) and write the output to 'file3'.
```
cat file1 file2 >> file3
```

Display 'cat.mak' in verbose mode and display TABs as '^I' and the end of line as '$'
```
cat -evt cat.mak
```

CAT_DEP = c:\madhur\src\UnixDos\def\def.h \^M$
^Ic:\madhur\src\UnixDos\def\l_decl.h \^M$
^Ic:\madhur\src\UnixDos\def\exparg.h \^M$
^Ic:\madhur\src\UnixDos\def\def_s.h^M$
^M$
^M$
all:^I$(PROJ).EXE^M$
^M$
CAT.OBJ:^ICAT

Notes:
Never specify any input file also as the output file, because before CAT will read any input   the OPERATING SYSTEM will create the output file and destroy any old contents:
```
cat file1 file2 >> file1
```

cc (C Compiler Front End)

<u>Syntax:</u>
```
cc [options] {file(s)}

options    compiler options specific to compiler
file(s)    file(s) to work on
```

<u>Description:</u>
CC is a front end to the following compilers:
* MicroSoft (CL)
* Zortech   (ZTC)
* TurboC/Borland (TCC)
* Intel 486 (ICC)

Most compiler will show the errors and warnings on the screen but will NOT create an error or warning file which contains only those messages for each compiled module. This means that you cannot compile automatically libraries and DLL's without loosing those important messages.
CC allows you to compile AND catch those messages in a separate files for each compiled module.

<u>*Example1:*</u>
```
cc -c getsize.c
```
Might produce the following diagnostics:
```
getsize.c(34) : warning C4101: 'NotUsed' : unreferenced local variable
getsize.c(29) : error C2065: 'Error' : undeclared identifier
getsize.c(29) : error C2146: syntax error : missing ';' before identifier
'here'
getsize.c(30) : error C2065: 'here' : undeclared identifier
getsize.c(30) : error C2143: syntax error : missing ';' before 'type'
getsize.c(33) : error C2065: 'statbuf' : undeclared identifier
getsize.c(33) : warning C4049: 'argument' : indirection to different types
getsize.c(33) : error C2224: left of '.st_size' must have class/struct/union
type
```
All these diagnostics would normally be lost but are now stored and available for later use
in the error C file (*.ERC) for this module 'getsize':
```
getsize.erc
```

<u>*Example2:*</u>
When you compile a lot of modules the screen in usually cluttered with all the compiler options and you can easily overlook any diagnostics.
Here is an example which compiles the 'somedll' MicroSoft makefile (somedll.mak):
```
nmake /nologo DEBUG=1 -f somedll.mak
```
The screen would show something like:
```
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
```

```
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPREV.CPP
rmprev.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRINT.CPP
rmprint.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRPTFT.CPP
rmprptft.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRPTHD.CPP
rmprpthd.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c GETSIZE.C
getsize.c
getsize.c(34) : warning C4101: 'NotUsed' : unreferenced local variable
getsize.c(29) : error C2065: 'Error' : undeclared identifier
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRTBMP.CPP
rmprtbmp.cpp
cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D "_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRTCAL.CPP
rmprtcal.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRTDB.CPP
rmprtdb.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRTDLG.CPP
rmprtdlg.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRTGRF.CPP
rmprtgrf.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRTHGB.CPP
rmprthgb.cpp
        cl /nologo /G2 /Zp1 /W3 /Zi /ALw /Gx- /Od /D "_AFXDLL" /D "_DEBUG" /D
"_
RMMGRDLL" /D "STRICT" /GD /GEf /Fd"RMMGR.PDB" /YuRMSTDAFX.H /c RMPRTITM.CPP
rmprtitm.cpp
```

In this example all the diagnostics are lost once they are gone from the screen:
the module `getsize` has a few problems.
To fix this situation check out the next example below...

Example3:
Now we compile a library using the CC front end so that the screen is not cluttered with

compiler options interspersed with the diagnostics.
Here is an example which compiles the 'llib' UnixDos makefile (llib.mk):
```
nmake /nologo DEBUG=1 -f llib.mk
```

The screen would now show something like:
```
        cc -c add_func.c                                # LLIB
        cc -c bgetl.c                                   # LLIB
        cc -c bname_t.c                                 # LLIB
        cc -c bread.c                                   # LLIB
        cc -c bwrite.c                                  # LLIB
        cc -c chrchk.c                                  # LLIB
        cc -c cmp_file.c                                # LLIB
        cc -c cntdel.c                                  # LLIB
        cc -c cntdeler.c                                # LLIB
        cc -c crbe.c                                    # LLIB
        cc -c ctyp.c                                    # LLIB
        cc -c debfun.c                                  # LLIB
        cc -c dir.c                                     # LLIB
        cc -c do_arg.c                                  # LLIB
        cc -c dx_chdir.c                                # LLIB
        cc -c dx_getdi.c                                # LLIB
        cc -c fclose_.c                                 # LLIB
        cc -c fgetl.c                                   # LLIB
        cc -c fldcmp.c                                  # LLIB
        cc -c fn_cp.c                                   # LLIB
        cc -c fopen_.c                                  # LLIB
        cc -c get_fram.c                                # LLIB
        cc -c get_free.c                                # LLIB
        cc -c getopt.c                                  # LLIB
        cc -c get_fram.c                                # LLIB
        cc -c get_free.c                                # LLIB
        cc -c getopt.c                                  # LLIB
        cc -c getsize.c                                 # LLIB
getsize.c(34) : warning C4101: 'NotUsed' : unreferenced local variable
getsize.c(29) : error C2065: 'Error' : undeclared identifier
getsize.c(29) : error C2146: syntax error : missing ';' before identifier
'here'
getsize.c(30) : error C2065: 'here' : undeclared identifier
getsize.c(30) : error C2143: syntax error : missing ';' before 'type'
getsize.c(33) : error C2065: 'statbuf' : undeclared identifier
getsize.c(33) : warning C4049: 'argument' : indirection to different types
getsize.c(33) : error C2224: left of '.st_size' must have class/struct/union
type
        cc -c htoi.c                                    # LLIB
        cc -c htol.c                                    # LLIB
        cc -c llib.c                                    # LLIB
        cc -c mem.c                                     # LLIB
        cc -c otol.c                                    # LLIB
        cc -c pk.c                                      # LLIB
        cc -c prin.c                                    # LLIB
prin.c(50) : warning C4013: 'system' undefined; assuming extern returning int
        cc -c qsort_.c                                  # LLIB
        cc -c read.c                                    # LLIB
        cc -c regexp.c                                  # LLIB....
```
Now you can easily spot diagnostics and see the module name.

All diagnostics are also captured in the "error C files" (in this case 'getsize.erc' and 'prin.erc')

Example4:
You can also easily check if there are any errors in any of the modules with the following command (which shows the number of lines '-l' in each error C file '*.erc'):
```
wc -l *.erc
```

The screen would now show something like:
```
0 add_func.erc
0 bgetl.erc
0 bname_t.erc
0 bread.erc
0 bwrite.erc
0 chrchk.erc
0 cmp_file.erc
0 cntdel.erc
0 cntdeler.erc
0 crbe.erc
0 ctyp.erc
0 debfun.erc
0 dir.erc
0 do_arg.erc
0 dx_chdir.erc
0 dx_getdi.erc
0 fclose_.erc
0 fgetl.erc
0 fldcmp.erc
0 fn_cp.erc
0 fopen_.erc
0 get_fram.erc
0 get_free.erc
0 getopt.erc
8 getsize.erc
0 htoi.erc
0 htol.erc
0 llib.erc
0 mem.erc
0 otol.erc
0 pk.erc
1 prin.erc
9 TOTAL
```
Or you can select only the files which don't have a zero lines (option '-n' added):
```
wc -nl *.erc
```

The screen would now show something like:
```
8 getsize.erc
1 prin.erc
9 TOTAL
```


All these error files will not clog up your hard disk because CC will produce a zero length error file when no error/warning has been produced. (Otherwise even though the error file might have only 40 bytes it would still use up a whole DOS cluster which can

be 32K on 1GB hard drives):

Example5:
To implement the CC front end you can convert the regular MicroSoft makefile generated from Visual C++ into the UnixDos makefile.
Here we convert the 'yourprog.mak' makefile into the UnixDos makefile 'yourprog.mk' use the UnixDos utility MAK2MK:
```
mak2mk  yourprog.mak > yourprog.mk
```

Now to run the new 'yourprog.mak' makefile type:
```
nmake /nologo DEBUG=0 -f yourprog.mk
```

Note
In some cases the PATH is not set correctly and CC or NMAKE will use the wrong version of command.com, and you might see:
```
    cc -c yourprog1.c
Incorrect DOS version
    cc -c yourprog2.c
Incorrect DOS version
...
```
To correct this problem:
1. place the correct directory where COMMAND.COM is located in the PATH setting in AUTOEXEC.BAT (AUTOEXEC.NT under Windows NT)
    or
2. edit the CC.BAT file and hard code the path for command.com:
    ```
    cc1 %1 %2 %3 %4 %5 %6 %7 %8 %9
    e:\winnt40\system32\command /c cc2
    del cc2.bat
    ```

Related utilities
To use minimal memory while compiling CC is actually a batch file(cc.bat) which is calling the following UnixDos utilities:
CC1   (C Compiler preparation)
CC3   (CC Post processing)
These utilities are described in detail below.

Here is the actual cc.bat:
```
cc1 %1 %2 %3 %4 %5 %6 %7 %8 %9
command /c cc2
del cc2.bat
```

CC1 generates the temporary compile batch file 'cc2.bat' which compiles and then calls the CC3 post processing utility. Then the temporary batch file 'cc2.bat' is removed.

If you don't use the MAK2MK conversion utility the following environment variables are needed
(Examples for MicroSoft environment):

CCDEF   main include directory:
             `set CCDEF=\src\local\def`

CCDEF_   secondary include directory:
             `set CCDEF_=\src\global\def`

CCFLAGS   release compile command line:
             `set CCFLAGS=cl  -nologo -AL -I%s -I%s -Gs -Os`
`-DINSTALL`

CCFLAGS_   debug compile command line:
             `set CCFLAGS_=cl -nologo -AL -I%s -I%s -Od -Zi`
`-DINSTALL`

cc1 (C Compiler preparation)

<u>Syntax:</u>
```
cc1 [C compiler options] {source file)
```

```
Environment Variables used:
%CFLAGS%      use C compiler options as is in CFLAGS
```

```
If %CFLAGS% is not defined the following variables are used instead:
%CCFLAGS%     defines the debug   compiler command line with options
              (%CCFLAGS% is used if %DEBUG%=ON is set, otherwise
%CCFLAGS_%)
%CCFLAGS_%    defines the release compiler command line with options
%CCDEF%       defines the first  include directory (-I%s)
%CCDEF_%      defines the second include directory (-I%s, optional)
'%s' wtithin 'CCFLAGS' or 'CCFLAGS_' is mapped to 'CCDEF and CCDEF_'
%CCDEBUG%     will show internal debug information
```

```
To specify '=' in FLAGS use '~' (i.e. set CCFLAGS=-DMYDEF~1 -AL)
```

<u>Description:</u>
CC1 is an internal UnixDos utility and most probably you will never call it yourself from the command line. I just document it here for a complete reference.
CC1 will read the requested C compiler options and generate a custom batch file 'CC2.BAT'.
This batch file will do the actual compilation. I had to go this kind of round about way because the compiler usually needs the full conventional memory available to himself and calling it from within a program uses up too much memory, but generating a batch file and then calling it doesn't use any memory (except a few K for another the command.com shell)!

Note that you have to include the actual compiler utility in the CCFLAGS environment:
```
set CCFLAGS=cl  -nologo -AL -I%s -I%s -Gs -Os -DINSTALL
```
- MicroSoft (CL)
- Zortech   (ZTC)
- TurboC/Borland (TCC)
- Intel 486 (ICC)

All the command line options are written to a temporary C response file (cc_crf.tmp) because the primitive DOS still cannot accept command lines longer than 128 characters and you can easily get long command line options with tons of compiler options.

CC1 will generate the CC2.BAT which looks like this:
```
@echo off
CL /nologo @cc_crf.tmp > ~
```

```
cc3 file1.erc file1.sp 0
```

The compiler original output is written to the work file '~'.
This work file is then processed from CC3 (see below).

cc3 (C Compiler post processing)

Syntax:
```
cc3 {fnerr} {fnfast} {sw}
fnerr   = Filename to write errors/warnings to
fnfast  = Filename to write the speed optimization time stamp
sw      = 0=no speed focus, 1=speed time optimization focus
```

Description:
CC3 is an internal UnixDos utility and most probably you will never call it yourself from the command line. I just document it here for a complete reference.
CC3 will read the current work file '~' and depending on the compiler environment extract only the error(s) and warning(s) into the 'fnerr' file as specified on the command line.

Note that you have to include the actual compiler utility in the CCFLAGS environment:
```
set CCFLAGS=cl  -nologo -AL -I%s -I%s -Gs -Os -DINSTALL
```
- MicroSoft (CL)
- Zortech   (ZTC)
- TurboC/Borland (TCC)
- Intel 486 (ICC)

After CC3 is finished it will automatically remove the work file '~' and the temporary C compiler response file 'cc_crf.tmp'.

(u)cd (Change directory and/or drive)

<u>Syntax:</u>
```
ucd [[drive:]new directory]

drive      the drive letter (if not specified use the current
drive)
directory the new directory to go to
```

<u>Description:</u>
UCD will change the current directory:
```
ucd \tmp
```

You can use forward or backward slashes:
```
ucd \tmp/save
ucd \tmp\save
ucd /tmp/save
```

UCD can also change directly to a new drive unlike the DOS CD command:
```
ucd f:/login
```

You can use regular expressions for the last directory name:
```
ucd \tm*
```
Will go to `\tmp`.
```
ucd /tmp/sa*
```
Will go to `\tmp\save`.

If several directories are matching UCD will just use the first one!
```
ucd /t*
```

You can use the UNIX style multi dot upward change directory:
```
cd ..              one directory up
cd ...             two directories up
cd ....            three directories up   ...
```

If only the drive is specified UCD will go automatically to the root directory on that drive:
```
ucd g:
```
Will go to `G:\`

If   the new directory does not start with the '/' or '\' character
UCD will position you into a subdirectory with the specified name.

<u>Notes:</u>
If you call "cd" under UNIX without specifying any directory it will place you into the
"home directory". Since there is no reliable equivalent under DOS the UnixDos "(u)cd"

will just display the current drive and directory, to keep it compatible with the DOS CD.

chmod (Change file mode)

Syntax:
```
chmod [-l] {[+|-][r|w|h|s|a]} {file(s)...}

-l      = list       : show each file being updated
+|-     = on/off     : + sets , - resets specified attribute
r       = read-only  : read-only attribute
w       = read/write : read/write attribute
h       = hidden     : hidden file attribute
s       = system     : system file attribute
a       = archive    : archive file attribute
{+|-{r|w|h|s|a}} {file(s)}
```

Description:
CHMOD will alter the file mode/attribute as specified (see above).

Example:
```
chmod -w *.c
```
Will make all C source files read-only (-w stands for taking away the write permission!)
Now they are protected against updates and deletion:
```
DEL *.c
```
Will show something like:
C:\tmp\CLEAR.C
Access is denied.

```
chmod -l +w *.c
```
Will make all C source files again available for read/write and show them as they are changed:
clear.c
...

Notes:
Several options found in under UNIX do not apply to the DOS environment:
group, other, execution permission, set-owner, sticky bit. Other options are DOS specify:
hidden, system, archive.
The direct octal assignment found under UNIX has been dropped since DOS does not
have any equivalent to the 3 layered security: owner/group/other.

cf (Fast Compare)

Syntax:

```
cf {file1} {file2} [bytes]
file1 = first file
file2 = second file
bytes = number of bytes to be compared (default is ALL)
```

Description:

CF compares the two files specified on the command line using a much faster method than the UNIX cmp command. By default CF will compare the entire file, but you can specify a smaller value if you want to compare only a sub segment. CF will show how many bytes it has actually compared. If you need to compare segments not starting at the file beginning just use first DD or IO utility to extract that segment into a new file.

If the files have a different content CF will show the first difference:
```
Comparing 21077 bytes
        0:  98  32
File 'file1' and 'file2' are NOT identical (CONTENTS)
```

If one file is shorter than the other CF will abort with a message:
```
File 'file1' and 'file2' are NOT identical (SIZE)
```

CF will exit with the following status:

0	file1 and file2 is identical
1	file1 and file2 have the same size but different contents
2	file1 and file2 have different sizes
3	internal error (out of memory etc.)
4	cannot open file1
5	cannot open file2

Example:
```
cf   cf.exe cmp.exe 10
```

Will show:
```
Comparing 10 bytes
        2: 163 238
File 'cf.exe' and 'cmp.exe' are NOT identical (CONTENTS)
```

Remarks:
The main reason why we created this new utility is speed:
it is approximately 20 times faster then the good old "cmp" which can make a big difference
if you have to compare 200 MB.

clear (Clear screen)

<u>Syntax:</u>
```
clear
```

<u>Description:</u>
CLEAR will erase the screen using the "cls" DOS command.

cmp (Compare two files)

Syntax:
```
cmp [-b{size}] [-l] [-s] file1 file2
-b{size}  = buffer: set the buffer size in memory (default =
20000)
-l        = list  : show each difference:
                    offset as decimal(0 based), bytes in hex and
char
-s = silent: don't display error messages
```

Description:
CMP will compare each byte of file1 and file2.
Depending on the result it will return the following exit codes:
0   files are identical
1   file are different
2   cannot access file
3   internal error (out of memory)

If you also want to see the differences between the files use the '-l' option.
It will show both the file names:
Offset   cmp.exe   cf.exe
Then it will list each byte mismatch:
2   0xEE(^O)   0xA4(^F)
You can see the decimal offset (2), the byte in file1 as hexadecimal value (0xEE) and as character (^O) and the corresponding byte in file2 as hexadecimal value 0xA4 and again as text character (^F).

Example:
```
cmp -l cmp.exe cf.exe
```

Will show all differences:
Offset   cmp.exe   cf.exe
*   2   0xEE(^O)   0xA4(^O)*
*   4   0x81(^O)   0x4E(N)*
...

Notes:
The UNIX CMP will not give you any help which value belongs to which file and also show the bytes only as octal value:
*   23 334 101*
*   24 007 102*
*   25 036 103*

Remarks:
I never liked the octal only display from UNIX (or MKS) and found ASCII and

hexadecimal representation much more useful.
I also missed an indication which column relates to which file from the command line.

cmpdir (Compare two directories)

```
cmpdir {dir1} {dir2} [-{c|C|d|D|f|F|s|S|t|T}]
dir1 = dir1    : first directory
dir2 = dir2    : second directory
-c   = content : use the file contents for comparison
-d   = diff    : like -c' but will call the 'diff' program
                 the diff output is send to ./{file}.di{n}
                 {n} cycles thru '0-9' and then 'a-z' to get
unique names
-f   = func    : like '-d' but with function header information
-s   = size    : use the file size for comparison
-t   = time    : use the modification time (default)
(Capital options will show the filenames as they are processed)
The output is displayed on the screen and written to file
'cmpdir.out'

You can specify a custom diff program: set UDDIFF=VDIFF -B
(Uses '%TMP%/cmpdir1.tmp' and '%TMP%/cmpdir2.tmp' as workfiles)
(or '/cmpdir1.tmp' and '/cmpdir1.tmp' if %TMP% is not defined)
```

Description:
CMPDIR will compare all the files in the two specified directories.
By default it will focus on the modification time of the files as the criteria for comparison.
You can choose the following comparison criteria:
- file time (-t)
- file size (-s)
- file contents (-c)
- file contents with automatic creation of a difference file by running: `diff -b`
- file contents with automatic creation of a difference file by running: `diff -b -f`

CMPDIR will produce a two column output showing all files on each directory.
If the file is found in both directories CMPDIR will append the following comment depending on the comparison criteria:
- file time: (same),(older),(newer)
- file size   (same),(smaller),(bigger)
- file content   (same),(diff)

CMPDIR will display the result and also write the result to 'cmpdir.out' in the current directory so you can view and edit it.

If you want to use a different program to show the differences set the UDDIFF environment:

```
set UDDIFF=VDIFF -B
```

Example:
Compare '/tmp' and '/dos' based on time:
```
cmpdir /tmp /dos

/tmp                                /dos
----------------------------------------------------------------
4201.cpi      (same)                4201.cpi      (same)
4208.cpi      (same)                4208.cpi      (same)
5202.cpi      (same)                5202.cpi      (same)
ansi.sys      (same)                ansi.sys      (same)
append.exe    (newer)               append.exe    (older)
appnotes.txt  (same)                appnotes.txt  (same)
assign.com    (same)                assign.com    (same)
attrib.exe    (same)                attrib.exe    (same)
backup.exe    (same)                autoexec.bat
chkdsk.exe    (older)               backup.exe    (same)
cmd_orig.com  (same)                chkdsk.exe    (newer)
command.com   (same)                cmd_orig.com  (same)
comp.exe      (newer)               command.com   (same)
m4                                  comp.exe      (older)
                                    config.sys
                                    country.sys
                                    debug.exe
                                    diskcomp.com
                                    diskcopy.com
                                    display.sys
```

Note that the files 'm4' and 'autoexec.bat' do not have a trailing
comment which means that they don't have a corresponding file in the
other directory.

Notes:
UNIX provides only the DIRCMP tool without a choice of the focus and
a slower performance because DIRCMP is only a SHELL script instead
on an executable like CMPDIR.

comm (Select or reject common lines in two files)

```
comm [-l] [-[123]] {file1} {file2}

-l      = list : list filenames and results in 25 column format
-1      = col1 : suppress lines only in file1
-2      = col2 : suppress lines only in file2
-3      = col3 : suppress lines found in both files
```

Description:
COMM will read the two specified text files and determine which lines are common to both files and which one appear in only one of them.

COMM will then display the result according to the specified options:

nothing   show all three categories: only file1, only file2, both files (this is the default)

-1   show lines found only in file2 and in both files, not those only found in file1

-2   show lines found only in file1 and in both files, not those only found in file2

-3   show lines found only in file1 and file2, not those found in both files

-12   show only lines found in both files, not those found only in file1 and file2

-13   show lines found only in file 2, not those found only in file1 and in both files

-23   show lines found only in file 1, not those found only in file2 and in both files

-123   shows nothing because all three categories are suppressed.

Example:

If file 'comm1.txt' contains:

1
2
3
4
5
6

and file 'comm2.txt' contains:

1
3
5
7
9

To compare both these files enter:

```
comm comm1.txt comm2.txt
```

The result is:

```
                        1
2
                        3
4
                        5
6
        7
        9
```

Which means:
- 2,4 and 6 are found only in file1 (comm1.txt)
- 7 and 9 are found only in file2 (comm2.txt)
- 1,3 and 5 are found in both files

To find only the lines which are common to both files:
```
comm -12 comm1.txt comm2.txt
```

The result is:
```
1
3
5
```

To find the lines which are only found in file1:
```
comm -23 comm1.txt comm2.txt
```

The result is:
```
2
4
6
```

conv (Convert numbers to/from any base)

<u>Syntax:</u>
```
conv (Convert Integers)
usage: conv {value} [input-base (16)] [output-base (10)]
```

<u>Description:</u>
CONV allows you to convert integert values from any base (like 16 for hexadecimal) to any other base (like 10 for decimal) . The 32bit version can handle 64bit values, while the 16bit version can handle up to 32bit values.

<u>Example1:</u>
```
conv 123
```

Will show:
123 (hex) = 291 (decimal)

<u>Example2:</u>
```
conv 123 10 8
```

Will show:
123 (decimal) = 173 (octal)

<u>Example3:</u>
```
conv 123 16 2
```

Will show:
123 (hex) = 100100011 (binary)

<u>Example4 (Windows 95/NT only):</u>
```
conv 100100011010000010010001101000001 2 16
```

Will show:
100100011010000010010001101000001 (binary) = 123412341 (hex)
<u>Note:</u> CONV can go beyond the 32bit value range!

cp   (Copy files)

Syntax:
```
cp [-flv] {src} {dest}

-f     = force : always remove old files (even if read-only)
-l     = list  : show which file is currently copied
-v     = verify: verify that input is identical to output
src    = source: one or more files to be copied
dest   = destination:
           if several source files this must be a directory name
           if one source file can be directory or filename
```

Description:
CP will copy one or several files to the specified destination.
The modification time and file modes are preserved.

Example1:
To see the copy in progress specify the '-l' option:
```
cp *.exe /put
```

Might show:
```
add.exe               -> /put/add.exe
add_func.exe          -> /put/add_func.exe
af.exe                -> /put/af.exe
banner.exe            -> /put/banner.exe
basename.exe          -> /put/basename.exe
bbanner.exe           -> /put/bbanner.exe
bed.exe               -> /put/bed.exe
beep.exe              -> /put/beep.exe
bell.exe              -> /put/bell.exe
bgrep.exe             -> /put/bgrep.exe
bmv.exe               -> /put/bmv.exe
bvi.exe               -> /put/bvi.exe
...
```

Example2:
To ensure the copy is identical to the original specify the verify '-v' option.
This is very useful especially when you copy files to a floppy disk, because CP will
actually re-read the entire destination file and compare it byte by byte against the
original
(unlike the DOS COPY verify which just checks the verify of the controller):
```
cp -lv bvi.exe a:
```

Will show:
```
bvi.exe               -> a:/bvi.exe
1 files written successfully
```

Example3:

To copy file1 to file2:
```
cp file1 file2
```

If file2 exists already it will be overwritten.
If file2 is read-only protected CP will show the following error message:
cp: Cannot remove old 'file2'

Example4:
To force the copy even on top of a read-only or write protected file use:
```
cp -f file1 file2
```

Example5:
To copy several files use the regular expression or list the source files:
```
cp  mod1.mak mod1.def *.c \tmp
```

All modification times and file modes are preserved.

Example6:
In this example we are trying to copy several source files into a file or non existent directory:
```
cp *.c  /notadir
```

If the destination is not a directory and you specified several source files CP will show the following error message:
cp: Target '/notadir' is not a directory

Example7:
```
cp file1 file1
```

In this case you tried to copy a file onto itself and CP will show the following message:
cp: Cannot copy 'file1' onto itself

Notes:
The UNIX CP does not offer the '-l' (list) or '-v'(verify) or '-f'(force) option and will not warn
you when you copy several files into a file instead of a directory.

Remarks:
I always liked the DOS style progress report while the copy is crunching away but missed there the exact destination. I encountered also several cases where the /V (verify) turned out to be bogus because it was not really re-reading the floppy disk I was copying to but just checking the controller return value - so we created a true verify (-v) which takes double the time since it truly has to re-reads the entire file again but offers absolute guaranty of a successful copy.

cpio (Copy files from input to output)

Syntax:
```
Create backup: cpio -o[vacdmf] [[D:]backup]
Read  backup: cpio -i[tcdmnuvf] [[D:]backup] [pattern...]
Move  files    : cpio -p[dvmau] [[D:]directory]
-a = access    : reset access time of input files after copy
-c = compatible: Write header in ASCII for portability
-d = directory : create directories if needed
-f = file      : specify backup file (default is stdin/stdout)
-m = mod. time : retain modification time for newly created
files
-s = swap      : swap bytes
-t = table     : print table of contents (only with '-i' option)
-u = uncondit. : copy UNCONDITIONALLY
                 (normally older file doesn't replace)
-v = verbose   : print names of affected files after completion
-V = invert    : do NOT copy files with match [pattern]

Example CREATE : cpio -ovf a:\backup < files.lst
Example RESTORE: cpio -idvmf a:\backup "*.c" (only C sources)
Example MOVING : find . -print | cpio -pdvmf \newdir
(Multi Volumes are supported!)
```

Description:
CPIO is an excellent tool to store/restore/move files or entire directory branches.
CPIO functions in the three following modes:
- Output/Backup: creates a CPIO archive
- Input/Restore: reads a CPIO archive
- Pass Thru: copies files and subdirectories to new location

OUTPUT/BACKUP (-o option)
Read the incoming file list and creates one CPIO archive.
Example: `cpio -o < in.lst > out.cp`
will read 'in.lst' and write the CPIO archive to 'out.cp'

When you write the CPIO archive onto a floppy drive CPIO first checks the available space, then subtract 10K and write the new archive. If CPIO finds an old archive with the identical name it will first attempt to remove and then calculate the available space. If the space has run out it will beep and prompt you for the next volume/floppy:
```
cpio -of a:new.cp < in.lst
```

DISK FULL – Hit return to start Volume 2 (Abort CTRL C)

This allows you to backup large set of files which can spawn across several floppy disks

(-f option) and even compresses files as the backup is proceeding (-z option).

INPUT/RESTORE (-I option):
Read the specified CPIO archive and restores all the files from a that archive.
Example: `cpio -id < in.cp`
will read 'in.cp' and re-create all the file in the appropriate directories (-d)

PASS THRU (-p option):
Read the specified input file list and copy the files to the new directory.
Example: `cpio -pd /newdir < in.lst`
will read filelist 'in.lst' and re-create all files under the '/newdir' directory.

Restore directories (-d option):
This option will prompt CPIO to recreate the directories as it restores files - otherwise all files will end up in the current directory. This option is always recommended in conjunction with the '-I' option.

Verbose (-v option)
To see the files which are currently processed use the '-v' option.
Example: `cpio -iv < in.cp`
will read 'in.cp' and show all the files in that archive while it restores its contents.

Table of contents (-t option)
To see the files in a CPIO archive use the '-t' option in conjunction with the verbose option to get a long listing '-tv'.
Example: `cpio -itv < in.cp`
will read 'in.cp' and show all the files in that archive, but not restores any files.

File List
To generate a files list use the LS utility: `ls *.c *.mak *.h > src.lst` or
the UFIND utility: `ufind /src -print > src.lst`
Then you can edit the contents with any text editor: `vi src.lst`

Unconditional restore (-u option)
As a security device CPIO will by default NOT restore any file which exists already and have a newer or identical time stamp than the file in the archive. This is to prevent overwriting newer files in the hard disk with older versions from the archive.
CPIO will display the following messages:
`current <your file here> same`   (when the time stamps are identical)
`current <your file here> newer`   (when the stored file timestamp is older than the existing one)
To force CPIO to restore even older archive versions unconditional use the '-u' option.

Relative and absolute locations
When you store files in a CPIO archive you have the following two choices:
- absolute file locations (the filename starts with a slash and shows the absolute location):

```
/tmp/save/file1.c
/tmp/save/file1.mak
...
```
- relative file locations (the filename does not start with a slash):
```
save/file1.c
save/file1.mak
...
```
When you restore the absolute file locations CPIO will always restore these in the same place on the hard drive.

When Relative filenames are restored CPIO will place them relative to the current position:

Example:

Create an archive 'archive.cp' relative to the '/tmp' directory with the two files 'file1.c'(relative) and 'file1.mak'(absoulte):

```
cd /tmp
cpio -ovf archive.cp < src.lst
```
Will show:
```
file1.c
/tmp/file1.mak
8 blocks
```

Now we change to a new location and restore:
```
cd /new
cpio -idvm < /tmp/archive.cp
```
Will show:
```
file1.c
current </tmp/file1.mak> same
/tmp/file1.mak
9 blocks
```
The 'file1.c' will be newly restored in '/new/file1.c'(relative).

The 'file1.mak' will not be restored because CPIO detected that it existed already in '/tmp/file1.mak' and that the time stamps are identical displaying the message:
```
current </tmp/file1.mak> same
```

Specify CPIO archive file (-f {out} option)

Specifies the CPIO archive output file. This option is usually used to direct the output onto a floppy disk. If a file with the specified name {out} exists already it will be removed automatically. If the output file {out} fills all the available space on a floppy disk CPIO will pause and prompt you to insert a new free floppy disk. This continues until all the specified files in '{list}' are backed up. CPIO will always leave a small amount of space (10,240 bytes) on the floppy so that you can store the file list or other descriptive files.

Example:
```
ufind / -print | cpio -ovf  A:backup.cp
```

Will backup all the files on the entire hard disk onto several volumes/floppies

if needed into the file 'backup.cp' in drive 'A:'. Each floppy will
contain a file 'backup.cp' with the CPIO archive. Note that the filenames
generated from the UFIND command all start with the '/' which ensures
that all files will be in the exact same directory and subdirectory
when you restore.

To backup only '*.c' and '*.h' files under the '/src' directory:
```
ucd /src
ufind . (-name '*.c' -o -name '*.h' ) -print > src.lst
cpio -ovf B:src.cp < src.lst
```
Will first create the list file 'src.lst' using the UnixDos UFIND command.
CPIO will then   backup all C source (extension '.c') and C include
files (extension '.h')   within the \SRC directory or any subdirectory.
The output is written to the floppy in drive B:.
If the space in that floppy is not enough to hold all the files CPIO will prompt you
automatically for another floppy disk.
Note that the filename does not start with the slash '/' because the
UFIND command had   only '.' specified as the starting directory which
allows you to restore the files into any directory later on.

Character/Text header format (-c option)
By default CPIO will use a binary form to store header information.
This option allows you to store the header information in text form which is machine
independent.

Restore modification time stamp (-m option)
This option instructs CPIO to set the modification time and file attributes to the original
values as they were when the archive has been created.
By default the file modification time will be the current time and read-only files will be
writable!
This option is always recommended in conjunction with the '-I' option.

Restore selectively:
To restore files from the archive selectively specify your match criteria after the archive
filename:
Example:
```
ucd /tmp
cpio -itf B:src.cp 'a*.c' 'a*.h'
```
Will give you a table of contents only of those files in the archive 'B:src.cp' which match
either the expression 'a*.c' (any file starting with letter 'a' and extension '.c') or 'a*.h'.
Note that the single quotes (') around the search criteria are needed to avoid the
premature filename expansion!

Notes:
The UNIX CPIO does not support spawning across multi volumes (-f option).
The '-a' option does not apply since only the modification time stamp is maintained in
DOS.

<u>Remarks:</u>
This is a great work horse I use extensively to move files and file-trees around especially across DOS/UNIX platforms. It works great in combination with the "(u)find" utility.

cut (Cut/Select specified fields from lines)

```
usage: cut [-c{list}|-C{list}|-f{list}] [-d{del}] [-s] {file(s)}
(Maximum line length is 10000 bytes)
-c      = cut out columns according to {list} (1 based, from
beginning)
-C      = cut out columns according to {list} (1 based, from end)
-f      = cut out fields according to {list}
-d      = {del} is the field delimiter (only in conjunction with
-f)
          the default is white space (\t and blank)
          (Use hex for special characters like pipe character '|'
-d\7C)
-s      = suppress lines with no delimiter (only with -f)
{list} = decimal single numbers or ranges:
          1-10          = col/field 1,2,3,4,5,6,7,8,9,10
          10,1          = col/field 10,1
          10-           = col/field 10,11,...end
          1-4,7,10-12   = col/field 1,2,3,4,7,10,11,12
Example: cut -c1-5,7-9 text (reads col 1-5 and 7-9 from file
text)
```

Description:
CUT is an excellent tool to extract ("cut out") specific parts from text lines.

You can either specify fixed column areas (-c option) or "fields" (-f option) which are separated by the "field delimiter" (default is blank and TAB).
You cannot specify both extract types together (-c and -f).

CUT will extract according to your specification in each specified file.
If you did not specify any file CUT will use STDIN.
The output will always be written to STDOUT which you can redirect or pipe to another command.

CUT allows you to define your own field delimiter with the -d option.

Example1:
If the input (from UDATE) is: `Wed Aug 05 07:07:10 1991`

`udate | cut -f5,2,4`
will show:
1991 Aug 07:07:55

Example2:

If file   'cut.txt' contains:
1|2|3|4|5|6
from|to|min|max
A|B|CCC|D|E|F
no delimiter
|two|three|four|five

To select now field 3, 1 and 5 until the end enter:
`cut "-d|" -f3,1,5- -s cut.txt`

will show:
3|1|5|6
min|from
CCC|A|E|F
three||five

Example3:
To select column 2 thru 4 enter:
`cut -c2-4 cut.txt`

will show:
|2|
rom
|B|
n d
wo|

Remarks:
It always irritated me that the field sequence is ignored in the UNIX version of CUT which prevents you from re-arranging the field order.

(u)date (Display/change the date & time)

```
usage: udate [+format|MMDDhhmm[yy]]
MMDDhhmm[yy] = will set the date to month=MM(01-12), day=DD(01-
31)
             hh=hour(0-23), mm=minutes(0-59), yy=year(19xx)
+format      = will display date/time according to format
             %m  = month (01-12)            %d  = day   (01-
31)
             %y  = year  (00-99)            %H  = hour  (00-
23)
             %M  = minute(00-59)            %S  = second(00-
59)
             %D  = date  (MM/DD/YY)         %T  = time
(HH:MM:SS)
             %j  = day number within year (000-366, 001=Jan
1st)
             %w  = day number within week (0-6, 0=Sun)
             %a  = day within week (Sun,Mon...Sat)
             %h  = month in year (Jan,Feb...Dec)
             %n  = insert a new line
             %c  = no trailing new line
             %t  = insert TAB
             %r  = time    (HH:MM:SS [A|P]M)
             %%  = percent itself
Example: date "+DATE: %m/%d/%y%nTIME: %H:%M:%S" will print:
DATE: 04/01/91
TIME: 18:19:20
```

Description:
UDATE will show the current date/time if no arguments are specified in the following format:
Wed Aug 05 07:18:00 1991

You can specify your own custom format how UDATE should display the date and/or time.
Just start the format specification with a plus(+) followed by text and any combination of the print commands listed above:

Example1:
```
udate "+DATE: %%m/%%d/%%y%%nTIME: %%H:%%M:%%S"
```

Will show something like:
DATE: 04/01/91
TIME: 18:19:20

Note the double quotes around the custom format to pass it into UDATE as one single argument instead of splitting it up into three separate arguments:
Arg1: "+DATE:", Arg2: "%%m/%%d/%%y%%nTIME:", Arg3: "%%H:%%M:%%S"

Note also the double '%' so that Windows95 does not interpret (escpecially %d%) wrong!

Example2:
`udate 0805071891`
This will set the system time to:
Aug 08, 1991 07:18:00

(08 stands for the month (August), 05 is the day, 07 is the hour, 18 are the minutes and 91 stands for the year 1991)

dd (Disk to Disk copy/conversion/extraction)

```
dd [option=value] ...
```

The following options are available:
```
if={f}          = define input file (default is stdin)
of={f}          = define output file (default is stdout)
ibs={n}         = Input buffer size (default = 512 bytes)
obs={n}         = Output buffer size (default = 512 bytes)
cbs={n}         = Conversion buffer size
bs={n}          = buffer size (overrides both ibs and obs)
skip={n}        = skip {n} input blocks (will still read not jump)
count={n}       = copy only {n} blocks from input to output
iseek={n}       = start to read at offset {n} (jumps to {n})
oseek={n}       = start to write at offset {n}
keep            = keep output file if it exists (default creates
new)
conv={a,b..}  = define the conversion functions:
                ascii     = EBCDIC to ASCII
                ebcdic    = ASCII to EBCDIC
                ibm       = ASCII to EBCDIC-IBM style
                [l|u]case = to lower/upper case
                swab      = swap every pair of bytes
                noerror   = don't stop processing if error
occurs
                sync      = pad every input block to ibs
                inv       = invert/flip bits
{n}             = numeric value:
Numeric value defines by default the number of bytes.
The following multipliers are  available:
w    word (2 bytes)       100w are 200 bytes
b    block (512 bytes)    10b  are 5120 bytes
k    kilobyte (1024)      2k   are 2048 bytes
m    mega (1000000)       2m   are 2000000 bytes

You can multiply several values using '*' or 'x':
100w*10b are 1024000 bytes
200*80   are 16000 bytes
```

Description:
DD will read the specified input, do some processing (if requested) and write the output to the output file. DD will report after completion the number of input and output blocks with remaining bytes.
```
        25+       1 blocks in
        25+       1 blocks out
```

Example1:
To convert a the input file 'in.dat' to upper case and write the result to output file
'out.dat':
```
dd if=in.dat of=out.dat conv=ucase
```

Example2:
To extract bytes 1001 thru 1500 from 'in.dat' into 'wrk.dat':
```
dd if=in.dat of=wrk.dat bs=500 skip=2 count=1
```

Example3:
To extract bytes 2,000,000 thru 5,000,000 from 'in.dat' and write those 3,000,000 bytes
into output file 'out.dat' starting at offset 1,000,000 (not 0!).
```
dd if=in.dat of=out.dat keep bs=1m iseek=2 oseek=1 count=3
```

First we prevent erasing of the output file 'out.dat' when it is opened: `keep`
Then we define the blocksize as one megabyte (1m = 1,000,000 bytes): `bs=1m`
Then we skip the first 2,000,000 bytes in the input 'in.dat': `iseek=2`
Then we define out output offset in 'out.dat': `oseek=1`
Then we define the number of bytes to actually transfer: `count=3`
The 'iseek; option is a much faster way to start reading at a specific input offset than the
"skip" method because it will not scan thru the unused area but directly jump there!

Note:
The UnixDos DD provides the iseek, oseek, keep and inv options which are not
available under UNIX!

Remarks:
DD is a great tool to move data chunks around.
But I always missed the ability to also write into not always to the beginning of the
output file and so we added the new "`iseek/oseek/keep`" commands.
I experienced also several situations where I had to "flip" or invert bits from backups on
QIC tapes and so I also added the "`inv`" converting option.

df   (Show free and total disk space)

Syntax:
```
df [-a] [-c] [-t] [-n{lst}] [-[drv1] [drv2] [drv3...]
-a      = ALL    : display space on all disks starting from C:
-c      = CHKDSK : use CHKDSK to determine the disk space
          (instead of the '_dos_getdiskfree' function which
doesn't
           always report the correct value on large SCSI drives)
-n{lst} = NETWORK: define the list of network drives (default F-
Z)
          (In 'lst' you can define letters or ranges:
           Example: -nF-IOS-U  defines network drives: FGHIOSTU)
          (you can also use the UDNETLST environment to specify
'lst')
          (the default is equivalent to: set UDNETLST=F-Z)
           To obtain the drive info for Network drives "NDIR
/spa" is used
-t       = TOTAL  : Don't show grand totals at the end
           (default show if > 1 drives)
drv     = display space for disk 'drv' (lower or upper case)
(By default df will display the space on the current disk)
```

Description:
DF will show you the free and total space in bytes.
DD will also show the volume label of the disk and the percentage of free space in relation to
the total space available.

The "-a" option will show the space of all available drives starting with C:

Example1:
Show the drive information of the current drive:
```
df
```

Might display:
*C:(MADHUR_C_1G   )   220,954,624 free (1,023,967,232 total) 21.6%
free*

Below are all the displayed elements listed:
C:	the drive displayed
MADHUR_C_1G	the volume label or network directory
220,954,624 free	the number of free space on this disk in bytes
1,023,967,232 total	the total space on this disk in bytes
21.6% free	the percentage of free space in relation to the total space

Example2:
To show the drive information for drive D:
```
df d:
```

Might display:
```
D:(DISKDRIVE_D        )     6,623,232 free (   129,931,264 total)   5.1%
free
```

Example3:
When there are more than one drive displayed DF will automatically show also a GRAND
TOTAL.

df -a

Might display:
```
C:(C_900MB            )    34,816,000 free (   901,857,280 total)   3.9%
free
D:(D_512MB            )    55,508,992 free (   536,428,544 total)  10.3%
free
E:(E_400MB            )    37,593,088 free (   433,258,496 total)   8.7%
free
F:(F_512MB            )    25,018,368 free (   536,428,544 total)   4.7%
free
G:(F_1GB_MUSIC        )   270,696,448 free ( 1,053,999,104 total)  25.7%
free
L:(COREL_345          )             0 free (   526,319,616 total)   0.0%
free
================================================================================
==
  (*** GRAND TOTAL *** )   423,632,896 free ( 3,988,291,584 total)  10.6%
free
```

Note: To switch of this GRAND TOTAL use the -t option.

Example4:
To show the drive information of all drives(-a) using CHKDSK(-c) for the non network
drives and defining drive F thru Z as network drives(-nF-Z):
```
df -a -c -nF-Z
```

Might display:
```
C:(MADHUR_C_2G        )   148,537,344 free ( 2,146,467,840 total)   6.9%
free
E:(STASH              )             0 free (   284,114,944 total)   0.0%
free
F:(DEV\SYS            ) 1,055,326,208 free ( 3,198,156,800 total)  33.0%
free
G:(DEV\DEV            ) 1,684,013,056 free ( 6,442,450,944 total)  26.1%
free
J:(DEV\DEV            ) 1,683,881,984 free ( 6,442,450,944 total)  26.1%
free
L:(DEV\HOMER          ) 1,347,092,480 free ( 3,145,531,392 total)  42.8%
free
```

```
M:(TAMARAC\VOL1        ) 1,311,965,184 free ( 4,718,592,000 total)  27.8%
free
R:(DEV\DEV             ) 1,683,030,016 free ( 6,442,450,944 total)  26.1%
free
S:(DEV\DEV             ) 1,683,095,552 free ( 6,442,450,944 total)  26.1%
free
T:(DEV\DEV             ) 1,683,095,552 free ( 6,442,450,944 total)  26.1%
free
U:(TAMARAC\KRUSTY      ) 1,553,465,344 free ( 6,494,683,136 total)  23.9%
free
V:(DEV\DEV             ) 1,683,030,016 free ( 6,442,450,944 total)  26.1%
free
X:(TAMARAC\KRUSTY      ) 1,553,465,344 free ( 6,494,683,136 total)  23.9%
free
Y:(TAMARAC\VOL1        ) 1,311,965,184 free ( 4,718,592,000 total)  27.8%
free
Z:(DEV\SYS             ) 1,055,326,208 free ( 3,198,156,800 total)  33.0%
free
```

Notes:

The UnixDos DF shows the space in use friendly bytes while the UNIX shows the space in clumsy blocks which might change from drive to drive (or kilobyte if you specify the -k option).

The "-i" (i-nodes) option under UNIX does not apply to the DOS environment.

Remarks:
I always wondered why the UNIX df shows blocks instead of bytes (the Berkeley shows at least kilobytes) so I switched to straight bytes which are never ambiguous like blocks!

MKS/UNIX DF Failure:
Here is one example from the DF utility from MKS which uses the UNIX style:
```
\mks\bin\df -kP
Filesystem    1024-blocks       Used   Available  Capacity Mounted on
C:                 999968     854720      145248       86% C:/
df: couldn't stat file system for "": not a directory or path not found
df: couldn't stat file system for "": not a directory or path not found
df: couldn't stat file system for "": not a directory or path not found
L:                 999968          0      999968        0% L:/
M:                 999968          0      999968        0% M:/
R:                 999968          0      999968        0% R:/
S:                 999968          0      999968        0% S:/
T:                 999968          0      999968        0% T:/
U:                 999968          0      999968        0% U:/
V:                 999968          0      999968        0% V:/
X:                 999968          0      999968        0% X:/
Y:                 999968          0      999968        0% Y:/
Z:                 999968          0      999968        0% Z:/
```

MKS gives you wrong information for all the drives:
MKS claims the C: drive has only 999,968k total space but as you can see below it has 2GB(2,146,467,840 bytes) - the free space is correct (145,248k) or 148,537,344 bytes:
```
chkdsk c:
The type of the file system is FAT.
Volume MADHUR_C_2G created 5/21/96 2:15 PM
Volume Serial Number is 20A9-90EA
```
2146467840 bytes total disk space.
```
 32014336 bytes in 27 hidden files.
 19267584 bytes in 584 directories.
1946746880 bytes in 20485 user files.
```
148537344 bytes available on disk.
```
    32768 bytes in each allocation unit.
    65505 total allocation units on disk.
     4530 allocation units available on disk.
```

MKS claims the network drive X: has only 999,968k total space but as you can see below it has 6GB(6,342,464k) - the free space is also wrong (999,968k) instead of 1,517,056k:
```
chkvol x:

Statistics for fixed volume TAMARAC/KRUSTY:

Total volume space:                      6,342,464   K Bytes
Space used by files:                     4,825,408   K Bytes
Space available to MADHUR:               1,517,056   K Bytes
```

MKS also claims that drives F:,G: and G: are not available:
```
df: couldn't stat file system for "": not a directory or path not found
```

UnixDos DF Example:
Here is in contract the output of the UnixDos DF: `df -a -c -nF-Z`
which reports the correct values for all drives:
```
C:(MADHUR_C_2G        )   148,537,344 free ( 2,146,467,840 total)   6.9%
free
F:(DEV\SYS            ) 1,055,326,208 free ( 3,198,156,800 total)  33.0%
free
G:(DEV\DEV            ) 1,684,013,056 free ( 6,442,450,944 total)  26.1%
free
J:(DEV\DEV            ) 1,683,881,984 free ( 6,442,450,944 total)  26.1%
free
L:(DEV\HOMER          ) 1,347,092,480 free ( 3,145,531,392 total)  42.8%
free
M:(TAMARAC\VOL1       ) 1,311,965,184 free ( 4,718,592,000 total)  27.8%
free
R:(DEV\DEV            ) 1,683,030,016 free ( 6,442,450,944 total)  26.1%
free
S:(DEV\DEV            ) 1,683,095,552 free ( 6,442,450,944 total)  26.1%
free
T:(DEV\DEV            ) 1,683,095,552 free ( 6,442,450,944 total)  26.1%
free
U:(TAMARAC\KRUSTY     ) 1,553,465,344 free ( 6,494,683,136 total)  23.9%
free
V:(DEV\DEV            ) 1,683,030,016 free ( 6,442,450,944 total)  26.1%
free
X:(TAMARAC\KRUSTY     ) 1,553,465,344 free ( 6,494,683,136 total)  23.9%
free
Y:(TAMARAC\VOL1       ) 1,311,965,184 free ( 4,718,592,000 total)  27.8%
free
Z:(DEV\SYS            ) 1,055,326,208 free ( 3,198,156,800 total)  33.0%
free
```

diff (Compare two text files and show the differences)

Syntax:
```
diff [-b] [-c{n}] [-d{n}] [-i] [-z] {file1} {file2}
-b     = BLANK  : Ignore blanks and TAB's in the comparison
-c{n}  = CONTEXT: Change the number of context lines (default =
3)
-d{n}  = DEPTH  : Change the depth of scope
                 (default = 500 lines look ahead)
-i     = IGNORE : Ignore upper/lower case during comparison
-z     = SIZE   : Do not show the file size (default = show)
(If file1 or file2 is a directory the other name will be
appended)
```

Description:
DIFF will compare the two text files and show the differences.

If both files are identical DIFF will not show the following message:
file 'diff1a.txt' and 'diff1a.txt' are identical

Context display (-c option):
To make it easier for you to view the context where the change occurred DIFF will display the unchanged lines before and after. By default DIFF will display the 3 lines before and 3 lines after the change.
You can change the number of context lines: `-c5` will show 5 lines before/after the change.
You can also suppers the context display all together: `-c0`

White space discard (-b option):
If you want to discard changes which only change the number of blanks of TAB's use the '-b' option.

Example1:
If file 'diff1a.txt' contains   and file 'diff1b.txt' contains:

```
A                               A
NEW1-1                          B
B                               NEW2-1
NEW1-2                          NEW2-2
C                               C
```

Now to compare both these files enter:
```
diff diff1a.txt diff1b.txt
```

Will show:
```
diff1a.txt (Wed Sep 18 10:15:00 1996, 25 bytes)
       diff1b.txt (Wed Sep 18 10:15:00 1996, 25 bytes)
```

```
-----------------------------------------------------------------------
        1           1       |A
+       2                   |NEW1-1
        3           2       |B
        4           3       |NEW1-2
        5           4       |C
-----------------------------------------------------------------------
        1           0       |A
        2           1       |NEW1-1
        3           2       |B
+       4                   |NEW1-2
-                   3       |NEW2-1
-                   4       |NEW2-2
        5           5       |C
-----------------------------------------------------------------------
```

DIFF detected two changes:

- Insert line "NEW1-1" between line "A" and "B".
  You can see the '+' sign indicating and ADD, and the lines number '2' only in the column of file 'diff1a.txt' where the new line occurred.
- Replace line "NEW1-2" with lines "NEW2-1" and "NEW2-2".
  you can see the '+' and '-' signs indicating a replace and the lines numbers in the appropriate columns under 'diff1a.txt' and 'diff1b.txt'.

Example2 (Show also the function names):

If file 'diff3a.txt' contains and file 'diff3b.txt' contains:

```
#include <stdio.h>                        #include <stdio.h>

int func()                                int func()
{                                         {
    return(18);                                   printf("In function\n");
}                                                 return(18);
                                          }
main(int ac,char *av[])
{                                         main(int ac,char *av[]))
    printf("func=%d\n",func());           {
    return(0);                                    printf("Hello, func
ret=%d\n",func());
}                                                 return(0);
                                          }
```

Now to compare both these files enter:
```
diff -f diff3a.txt diff3b.txt
```

Will show:
```
diff3a.txt (Thu Oct 17 17:46:40 1996, 146 bytes)
        diff3b.txt (Thu Oct 17 17:46:48 1996, 181 bytes)
-----------------------------------------------------------------------
```
FUNCTION: func()
```
        2           2       |
        3           3       |int func()
        4           4       |{
+                   5       |    printf("In function\n");
        5           6       |    return(18);
```

```
      6        7   |}
      7        8   |
-----------------------------------------------------------------------
FUNCTION: main(int ac, char *av[])
      7        8   |
      8        9   |main(int ac, char *av[])
      9       10   |{
+    10            |    printf("Hello func=%d\n",func());
-             11   |    printf("Hello, func ret=%d\n",func());
     11       12   |    return(0);
     12       13   |}
-----------------------------------------------------------------------
```

DIFF showed you also for each change in which function the change or insertion occurred.

DIFF also shows the arguments in case you use overloaded function names with different arguments in C++.

Notes/Remarks:

The output of the UNIX DIFF has not been designed to be easily readable by the user but by the ED editor - so the output is very cryptic.

I never liked this type of output and so I redesigned the output to be user friendly:
- showing the filename
- showing the context
- showing the lines numbers with each line

dio (Direct Disk Input/Output)

Syntax
```
dio {R|W} {Drv} {Start} {cnt} {data}
R|W        = Read/Write
Drv        = Drive (A-Z)
Start      = Start Sector Number (0...)
Cnt        = Number Sectors to transfer (1...)
Data       = File for exchange
Exit Code: 0=Exchange successful, 1=Error, 10x=Unexpected error
```

Description
DIO will directly read or write from/to any disk.
This allows you to make a disk image directly to a file.
This can be very useful for example to save many different kind of boot discs to files instead of having to keep the actual original floppy discs, because then they are regular files from which you can recreate the boot discs!

Example
Read for disk image from a 1.44MB floppy to file 'disc1.dat':

dio A R 0 2880 disc1.dat

Note
Watch out!
This can be a potential dangerous tool because if you WRITE the wrong image to your hard disc boot sector or FAT table you may loose files or the entire disc!

dirname (Extract directory)

Syntax:
```
dirname [-d] [-n] [file(s)]
-d       = display in DOS style (default UNIX style)
-n       = don't show the drive letter
file(s)  = the filename(s) to be converted
If no 'file(s)' are specified dirname will read STDIN and
convert each line
```

Description:
DIRNAME will extract the directory name from the specified path(s).
You can use the UNIX display style(default) or specify DOS style display (-d option).
You can either specify one or more filenames on the command line or provide a list of filenames in STDIN if you don't specify any filename.

Example1:
```
dirname c:\tmp/save/src.zip dir1/file.c U:\PARK\file3
```

Will show:
```
c:/tmp/save
dir1
u:/park
```

Example2:
```
dirname -d c:\tmp/save/src.zip dir1/file.c U:\PARK\file3
```

Will show:
```
C:\TMP\SAVE
DIR1
U:\PARK
```

Example3:
Display directory without the drive letter:
```
dirname -n -d c:\tmp/save/src.zip dir1/file.c U:\PARK\file3
```

Will show:
```
\TMP\SAVE
DIR1
\PARK
```

Example4:
Display all directories in DOS style:
```
ufind / -type d -exec dirname -d {} ^
```

dos2unix (Convert DOS text files to UNIX)

Syntax:
```
usage: dos2unix {file(s)}
file(s) = file(s) to convert
```

Description:
DOS2UNIX will convert each file from the DOS text format (Carriage Return(CR) and Line Feed(LF) at the end of each line) into the UNIX text format (only Line Feed(LF) at the end of each line).

Example:
To convert "list1.txt" and "list2.txt" into UNIX format enter:
```
dos2unix list1.txt list2.txt
```

It will just show you the processed filenames and replace the original file contents:
```
list1.txt
list2.txt
```

extname (Extract file extension)

```
extname [-d] [file(s)]
-d      = display in DOS style (default UNIX style)
file(s) = the filename(s) to be converted
If no 'file(s)' are specified extname will read STDIN and
convert each
line
```

Description:
EXTNAME will extract the file extension from the specified path(s).
You can use the UNIX display style(default) or specify DOS style display (-d option).
You can either specify one or more filenames on the command line or provide a list of filenames in STDIN if you don't specify any filename.

Example1:
```
extname c:\tmp/save/src.zip dir1/file.c U:\PARK\file3.txt
```

Will show:
```
zip
c
txt
```

Example2:
```
extname -d c:\tmp/save/src.zip dir1/file.c U:\PARK\file3.txt
```

Will show:
```
ZIP
C
TXT
```

Example3:
To find out all extensions used on your machine enter:
```
ufind / -type d -exec extname -d {} "^" > ext.lst
usort -u ext.lst  > ext2.lst
```

du (Show disk usage)

Syntax:
```
du [-a] [-b] [-f] [-h{n}] [-o] [-p] [-r] [-s] [directory|file
list ...]
If no directory is specified the current directory one is used
(.)
-a       = All      : list each file
-b       = Bytes    : display in number of bytes
-f       = FilesCnt : list number of files
-h{n}    = SpaceHogs: list the 'n' biggest files in the specified
directory
-H{n}    = SpaceHogs: like -h but show also permissions and date
-o       = Overview : print only sum of main sub directories and
root                                files
-p       = Padding  : don't pad the size, show real size in bytes
-r       = Reach    : print warning of inaccessible directories
-s       = Sum      : print only sum total of each directory
-c{n[k]}= cluster : set padding cluster size to {n} bytes
(k=1024)
(default: display in megabytes (megabyte = 1,000,000 bytes!)
```

Description:
DU is an excellent tool to give you a quick overview of your space used on the disks across your directories.
If you don't specify any arguments DU will show the totals space used of your current directory and of any subdirectories in megabytes (1,000,000 bytes!).

You can specify individual directories or even entire drives:
`du c`   will give the overview of the entire C drive
`du /windows`   will only give the details of the "/windows" branch

By default DU will list all the subdirectories; to show only the main subdirectories you can specify the -o option (Overview) which will give you only the initial directories and the size of all the files in the starting(or root) directory:
`du -s d: e:`   will give you an overview of the entire D and E drive

Example1:
Show the 10 biggest files(hogs) on the E drive:
`du -h10 e:`

Might generate the following output:
List of the 10 largest files sorted by size in descending order:
```
----------------------------------------------------------------
```

```
20,322,482 e:/MSVC/HELP/VCBKS15.HLP
16,610,031 e:/MSVC/HELP/VCSDK15.HLP
15,581,184 e:/MSVC/HELP/VCBKS15.IND
11,090,254 e:/WINDEV/WINDEV.ZIP
10,919,006 e:/PMW/PMW.WRK
 7,245,478 e:/DRG/MSDNCD15.IDX
 4,325,728 e:/MSOFFICE/ACCESS/MSACC20.HLP
 4,246,784 e:/MSOFFICE/EXCEL/EXCEL.EXE
 3,490,816 e:/MSOFFICE/WINWORD/WINWORD.EXE
 3,390,373 e:/VB30/WINAPI/WIN31WH.HLP
```

Example2:

Give an overview of the main directories on the G drive:

`du -o g:`

Might generate the following output:

```
Megabytes      (Cluster = 16384 bytes)
   0.9339    g:/DM
   0.4915    g:/DOS622
   1.6384    g:/digidemo
      ---    g:/DRV_C
   0.5898    g:/PARK
   0.1475    g:/Q
   1.8842    g:/S8_OLD
      ---    g:/SCSI
   0.9339    g:/SES
      ---    g:/TMP
   7.8479    g:/U
 767.7706    g:/WAV
   0.5407    g:/{Files}
 782.7784    g:/
```

Example3:

Show the 15 directories with the most files on the C drive

`du -h15 -f c:`

Might generate the following output:

```
List of the  20 largest files sorted by size in descending order:
-----------------------------------------------------------------
        1268  /WINDOWS/SYSTEM32
         713  /WIN95/SYSTEM
         688  /WINDOWS/SYSTEM
         591  /U/SAVE
         471  /WINDOWS
         285  /U
         283  /SRC/BIN
         281  /SRC/DEF
         258  /WIN95
         227  /DOC
         190  /Unixdos
```

```
180   /MUDA/BIN/BAT
146   /WIN95/INF
145   /
137   /NO8
```

Example4:

Show all subdirectories under the current directory:
```
du
```

Might generate the following output:
```
Megabytes      (Cluster = 16384 bytes)
  0.9830    ./bigwin
  0.5243    ./bin
  0.3277    ./bwt
     ---    ./db_dos
  1.5401    ./db_dump
  0.2949    ./dbmadhur/dll
  0.8192    ./dbmadhur/save
  2.8180    ./dbmadhur
  3.3751    ./def
     ---    ./fp
 10.4858    .
```
Note: Empty directories is actually shown as '---' instead of the cumbersome UNIX and MKS style still showing one block even if the directory is empty.
```
  226 ./bigwin
   85 ./bin
   71 ./dbmadhur/dll
  221 ./dbmadhur/save
 1810 ./dbmadhur
  799 ./def
    1 ./fp
  127 ./bwt
  620 ./db_dump
    1 ./db_dos
 4402 .
```
Note: UNIX and MKS displays the directories in confusing random order instead of the alphabetically sorted DU output in UnixDos!

Show Bytes (-b option):

If you want to get a list only of specific directories and see the actual bytes enter:
```
du -b /dos /windows
```

Might show to following output:
```
Bytes
   10289152    /dos
     196608    /windows/dcsi
     131072    /windows/fonts
     294912    /windows/msapps/artgalry
     655360    /windows/msapps/equation
     393216    /windows/msapps/wordart
   15925248    /windows/msapps
     393216    /windows/msmail/msmail
```

```
    1212416    /windows/msmail
     720896    /windows/nls/english
     851968    /windows/nls
   69369856    /windows/system
  222625792    /windows
```

Show space WITHOUT padding (-p option):
By default DU will show the space a file occupies on the disk NOT the actual number of bytes the file itself contains: for example If you have a file with 20 bytes in size it will still occupy on the hard disk on cluster (which is usually 16,384 under 1GB or 32,768 above 1GB drives). This extra space is called "padding space" and can eat up enormous space if you have a lot of files on your disk. To see how much ACTUAL space is used by your files enter:

```
du -b -p                                    du -b
```

Might show to following output   Compare this to the padded statistics:

```
     Bytes
     Bytes                                    Bytes
    106622    ./bigwin                       983040    ./bigwin
     37536    ./bin                          524288    ./bin
     63321    ./bwt                          327680    ./bwt
       ---    ./db_dos                          ---    ./db_dos
    302475    ./db_dump                     1540096    ./db_dump
     33852    ./dbmadhur/dll                 294912    ./dbmadhur/dll
    105257    ./dbmadhur/save                819200    ./dbmadhur/save
    906448    ./dbmadhur                    2818048    ./dbmadhur
    375975    ./def                         3375104    ./def
   2158505    .                            10485760    .
```

Note the actual space used is 2.158MB but on the disk 10.486MB is occupied (79.4% is padding exists or in other words wasted space).
The '-b' in conjunction with the '-p' option can be very useful if you want to make sure that an entire directory branch has been transferred (for example to CD-ROM or a floppy) because if the cluster sizes between the two media or disks are different you will get different uses although the files themselves are identical. The UnixDos DU allows you to compare two directory branches on different media WITHOUT the padding!

Show all files in subdirectories (-a option):
To get a list of all the files on your entire disk into the file 'all.lst' enter:
```
cd /
du -a / > all.lst
```

Show number of files (-f option):
In some cases you might want to check just the number of files in the directories.
This option allows you to instruct DU to show the file counts.
Includes in the file counts are also all the hidden files in each directory.
In the following example you will see the number of files in each directory on your disk:
```
du -f /
```

Might display:

```
     Files
      169        /dos
        5        /windows/dcsi
        1        /windows/forms/_classad.000
       48        /windows/forms/configs
        2        /windows/forms/t_answer.000
       54        /windows/forms
        2        /windows/msapps/artgalry
        3        /windows/msapps/equation
       34        /windows/msapps/grphflt
        2        /windows/msapps/msgraph
        5        /windows/msapps/msgraph5
       24        /windows/msapps/msimager
        7        /windows/msapps/msinfo
       18        /windows/msapps/msquery
        3        /windows/msapps/orgchart
       10        /windows/msapps/proof
        1        /windows/msapps/sheetcnv
       14        /windows/msapps/textconv
        4        /windows/msapps/wordart
      127        /windows/msapps
        4        /windows/msmail/msmail
       13        /windows/msmail
        5        /windows/nls/english
        9        /windows/nls
        8        /windows/repair
      816        /windows/system
        5        /windows/temp
     1616        /windows
```

Notes/Remarks:
The UnixDos DU shows the size in megabytes and empty directories as '---' instead of
the number of block as under UNIX/MKS. The '-b' and '-p' options are not available
under UNIX to obtain the actual/unpadded filesizes.
We never liked the cumbersome definition of a megabyte as 1,048,576 and so we are
using the user-friendly 1,000,000 byte as the definition as one megabyte!

drvname (Extract drive from filename)

Syntax:
```
drvname [-d] [file(s)]
-d      = display in DOS style (default UNIX style)
file(s)  = the filename(s) to be converted
If no 'file(s)' are specified drvname will read STDIN and
convert each line
```

Description:
DRVNAME will extract the drive letter from the specified list of filenames.
You can use the UNIX display style(default) or specify DOS style display (-d option).

Example1:
```
drvname c:\tmp/save/src.zip dir1/file.c U:\PARK\file3
```

Will show:
```
c:
c:
u:
```

Example2:
```
drvname -d c:\tmp/save/src.zip dir1/file.c U:\PARK\file3
```

Will show:
```
C:
C:
U:
```

Example3:
You can either specify one or more filenames on the command line or provide a list of
filenames in STDIN if you don't specify any filename.
Display all drives appearing in the list of files stored in 'new.lst' in DOS style:
```
dirname -d @new.lst
```

(u)echo (Echo arguments)

Syntax:

```
uecho [argument(s)|-e|-E|-s|-S]
-e      = disable      the escape sequence translation
-E      = enable again the escape sequence translation (default)
-s      = disable      the space between arguments
-S      = enable again the space between arguments (default)
The following escape sequences can be used in the arguments:
\b      = backspace
\c      = exit immediately (default is LF at the end)
\f      = form-feed
\n      = newline
\r      = carriage return
\t      = tab
\v      = vertical tab
\\      = backslash
\0n     = octal code
\xn     = hex code
If the argument starts with '%' or '$' the contents of that
environment variable is echoed)

$env replace with the contents of the environment variable
(if existing) %env  same as above
```

Description:
UECHO is used to display text and environment variables to the screen (STDOUT).

Example1:
To display the current date with a lead text:
```
uecho THE DATE IS: `udate`
```

Might show:
THE DATE IS: Tue Sep 24 09:19:12 1995

Example2:
You can also display environment variables and disable/enable the escape codes:
```
uecho  "LIB: " -e %LIB -E "\nPATH:" -e %PATH
```

Might show:
LIB: c:\v2deb\lib;c:\msvc\lib;c:\msvc\mfc\lib
PATH: c:\unixdos;c:\msvc\bin;c:\windows\system32

In this UECHO command is composed out of the following elements:
*   Display a straight text: "LIB: "
    (enclosed in quotes to append the trailing blank)

- Disable the escape sequences because within the %LIB environment variable the backslash is used for the directories and would be misunderstood from UECHO as escape command: `-d`
- Display the %LIB environment variable: `%LIB`
- Enable the escape commands again - we need it for the new line: `-e`
- Display straight text starting on the next line: `"\nPATH:"`
- Disable the escape command again (backslash is used also in %PATH): `-d`
- Display the %PATH environment variable: `%PATH`

Example3:
You can also join arguments together which would otherwise be separated by a space:
```
set FILENAME=file1
uecho  -s $FILENAME ".ext"
```

Will show:
file1.ext

Without the space suppression you will always get a space between the filename and extension:
```
uecho  $FILENAME ".ext"
```

Will show:
file1 .ext

Notes/Remarks:
The UNIX/MKS UECHO does not have the capability to disable/enable the escape commands, which sometimes limits it - especially in the DOS environment where (unfortunately) the backslash is used for the directory names. This was in my mind a big mistake when DOS was cloned from UNIX!

egrep (Search files for one or more pattern/expression)

Syntax:
```
egrep [-abcdhilnpsvx] [-f {file}] [-e {p}] {p} {file(s)}
-a      = always  : always show filename (even in a single file
search)
-b      = bytes   : show the byte offset in file of match
-c      = count   : show only count of matches
-d      = descend : search also descending subdirectories for
'file(s)
-e {p}  = expr.   : search pattern {p} (can be repeated!)
                    (-e is only needed if the pattern starts
with '-')
-f {f}  = file    : file with more pattern (can be repeated!)
-h      = hide    : don't show the filename
-i      = ignore  : upper or lower case
-l      = list    : show only the filename on match
-n      = number  : show line number of match
-s      = suppress: don't show error messages
-v      = inVert  : find all NON MATCHES
-x      = eXact   : find only exact match on whole line
{p}     = regular expression to be matched (up to 1000 total
pattern)
file(s) = file(s) to be searched for match

NOTE: to disable the argument expansion always use single
quotes:
egrep '*text*' *.c
If no files are specified STDIN is used.
(Maximum line length is 10000 bytes)
```

Description:
This is a excellent tool for searching files for text matches.
EGREP belongs to the following family of "grep" utilities:
• EGREP: can search for several expressions in the specified files
• FGREP: can search for several plain text items in the specified files
• GREP: can search for one single expression in the specified files

On the following pages all the options which are identical in all three grep utilities (EGREP/FGREP/GREP) will be described with examples.

To learn how to create advanced search pattern see "Regular Expression for Text Searches" in chapter "ARGUMENT EXPANSION" in the "USER MANUAL".

If you don't specify any input file(s) to examine the "grep" utilities will read the standard input (STDIN) for any matches.

Lines of up to 10,000 characters in length can be processed and also non-text files (like WORD *.doc documents) can be searched!

On the reference pages for FGREP and GREP you will find a reference to the following pages instead of repeating all the shared options.

Exit Code:
EGREP/FGREP/GREP will exit with the following exit status:

```
0      Found match(es)
1      Found no match
2      Syntax error or inaccessible file
```

Descend into Subdirectories (-d option)
This option allows you to search not just for files in the current directory but also in ALL the subdirectories.
Example1:
Search for "PATH" in all batch files (*.bat) on the entire drive:
```
cd \
grep -d PATH *.bat
```
Might find:
autoexec.bat: PATH C:\U;C:\WINDOWS;C:\DOS
autoexec.bat: set LIBPATH=t:\pvcs\dos
autoexec.bat: set DBPATH=c:\v2deb\mhss
madhur/src/install.bat: echo INSERT \UnixDos INTO THE PATH IN AUTOEXEC.BAT
madhur/u/autoexec.bat: PATH c:\q\;C:\DOS;c:\u;c:\brief;
madhur/u/old/muda.bat: set DBPATH=c:\muda\bin

Example2:
Search for "PATH" and "INCLUDE" in all batch files (*.bat) on the entire drive:
```
cd \
egrep -d -e PATH -e INCLUDE *.bat
```
Might find:
autoexec.bat: PATH C:\U;C:\WINDOWS;C:\DOS
autoexec.bat: set LIBPATH=t:\pvcs\dos
autoexec.bat: set INCLUDE=c:\msvc\include;c:\msvc\mfc\include;c:
\windev\include
autoexec.bat: set DBPATH=c:\v2deb\mhss
madhur/src/install.bat: echo INSERT \UnixDos INTO THE PATH IN AUTOEXEC.BAT
madhur/src/set_ud.bat: set INCLUDE=\madhur\src\ud\def;c:\msvc\include
madhur/u/old/do_icm.bat: set INCLUDE=\msc\def
madhur/u/old/muda.bat: set DBPATH=c:\muda\bin
madhur/u/old/set_ec.bat: set INCLUDE=\msc\def
madhur/u/old/set_ec.bat: set CD2PATH=l:\cd2

Ignore Case (-i option)
This option allows you to search for text no matter if the text found is in upper or lower case:

```
egrep -i c: /*.bat
fgrep -i c: /*.bat
grep -i c: /*.bat
```
Might find:
autoexec.bat: set PVCSFILE=c:/put.txt
autoexec.bat: set PVCSLOG=c:/pvcs.log
autoexec.bat: rem C:\MADHUR\NO8\NDD C:/Q
tmp_bat.bat: sh show_arg.sh c:/roma

List Filenames only (-l option)
This option allows you to produce a list of files which contain the search text:
Example1:
```
cd \
egrep -d -e PATH -e INCLUDE *.bat
```
Might find:
autoexec.bat
madhur/src/install.bat
madhur/src/set_ud.bat
madhur/u/autoexec.bat
madhur/u/install.bat
madhur/u/old/do_icm.bat
madhur/u/old/muda.bat
madhur/u/old/set.bat
madhur/u/old/set_ec.bat

Example2:
To get a list of all your batch (*.bat) and command (*.com) files on your hard disk enter:
```
cd \
grep -d -l . *.bat *.com
```
Might find:
autoexec.bat
makeall.bat
command.com
ntdetect.com
cvision/cvcopy.bat
cvision/cvset.bat
dos/assign.com
dos/command.com
dos/diskcomp.com
dos/diskcopy.com
dos/doskey.com
dos/dosshell.com
...
In this example I used the descending option (-d) and the file list-only option (-l). The expression is "." which means search for any character in a line - so it basically will reliably always trigger a "file find" for the files specified (*.bat) and (*.com).

Hide Filenames (-h option)
In some cases you might NOT want to see any filename in front of the matches.
This option allows you to suppress the default filename prefix:
```
cd \
egrep -d -h -e PATH -e INCLUDE *.bat
```

Might find:
```
PATH C:\U;C:\WINDOWS;C:\DOS
set LIBPATH=t:\pvcs\dos
set INCLUDE=c:\msvc\include;c:\msvc\mfc\include;c:\windev\include
set DBPATH=c:\v2deb\mhss
echo INSERT \UnixDos INTO THE PATH IN AUTOEXEC.BAT
set INCLUDE=\madhur\src\ud\def;c:\msvc\include
set INCLUDE=\msc\def
set DBPATH=c:\muda\bin
set INCLUDE=\msc\def
set CD2PATH=l:\cd2
```

Show also the Line number (-n option)
In some cases you might want to know also the line number where the matches occurred.
This option allows you to add this information after the filename prefix:
```
cd \
egrep -d -n -e PATH -e INCLUDE *.bat
```
Might find:
```
autoexec.bat:    9:PATH C:\U;C:\WINDOWS;C:\DOS
autoexec.bat:   28:set LIBPATH=t:\pvcs\dos
autoexec.bat:   48:set INCLUDE=c:\msvc\include;c:\msvc\mfc\include;c:
\windev\include
autoexec.bat:   62:set DBPATH=c:\v2deb\mhss
madhur/src/install.bat:   17:echo INSERT \UnixDos INTO THE PATH IN
AUTOEXEC.BAT
madhur/src/set_ud.bat:   11:set INCLUDE=\madhur\src\ud\def;c:\msvc\include
madhur/u/install.bat:   77:echo 30 more bytes. One possible way to shorten
your PATH
madhur/u/old/do_icm.bat:    2:set INCLUDE=\msc\def
madhur/u/old/muda.bat:    2:set DBPATH=c:\muda\bin
madhur/u/old/set.bat:    8:set INCLUDE=\msc\def
madhur/u/old/set_ec.bat:    9:set INCLUDE=\msc\def
madhur/u/old/set_ec.bat:   14:set CD2PATH=l:\cd2
```

Always show the filename (-a option)
If grep is instructed to search only one single file it by default not displaying the filename (since it is obvious in which file the match occurred!).
This option allows you to force always to display the filename prefix:
```
cd \
grep -a PATH autoexec.bat
```
Might find:
```
autoexec.bat: PATH C:\U;C:\WINDOWS;C:\DOS
autoexec.bat: set LIBPATH=t:\pvcs\dos
autoexec.bat: set DBPATH=c:\v2deb\mhss
```

Show binary file offset (-b option)
If you want to know specifically where in a file a pattern occurred you will need the actual byte offset (the line number does not point you to the specific place within the file).
This option allows you to get also the byte offset after the filename prefix:
```
grep -b PATH /*.bat
```

Might find:

```
/autoexec.bat:       247:PATH C:\U;C:\WINDOWS;C:\DOS
/autoexec.bat:       768:set LIBPATH=t:\pvcs\dos
/autoexec.bat:      1564:set DBPATH=c:\v2deb\mhss
```

Show the matches per file (-c option)

If you only want to know specifically how many matches occurred within each file, but not actually see all the matches this option allows you instruct EGREP/FGREP/GREP to provide this information after the filename prefix:

```
cd \
grep -d -b PATH *.bat
```

Might find:

```
autoexec.bat:4
makeall.bat:0
cvision/cvcopy.bat:0
cwright/mak2pjt/mak2pjt.bat:0
acc/save.bat:0
```

Show all lines which do NOT match (-v option)

In some cases you might want to find all the lines in a file which are not matching the search criteria. This option allows you invert the search logic.

This example will search for all lines in "autoexec.bat" which are NOT starting with the "set" command:

```
grep -v ^set autoexec.bat
```

Might find:

```
@ECHO OFF
SET NWLANGUAGE=ENGLISH
LH /L:0 C:\DOS\SMARTDRV.EXE C+ /V
PROMPT $t $p$g
PATH C:\U;C:\WINDOWS;C:\DOS
SET TEMP=C:\TMP
Rem LH /L:1,27984 MSCDEX /D:MSCD001 /V /L:E
f:
login name.dev.mcgraw_co
CALL C:\MSVC\BIN\MSVCVARS
...
```

Show only lines with match completely (-x option)

In some cases you might want to find all the lines which match only the entire line instead only a pattern. This option allows you change the search logic in this way.

If "grep1.txt" contains:

```
just text
text with number 18
18
before 18 after
1818
```

This example will search for all lines which contain exactly number "18" in file "grep1.txt":

```
grep -x 18 /unixdos/grep1.txt
```

Will find only the line with just contains "18":
18

Searching binary files
You can also search binary files (like Microsoft Word documents *.doc).
To search for EGREP in all word documents enter:
```
cd \
grep -d EGREP *.doc
```
Might find:
/unixdos/ud_ref.doc: The UNIX EGREP cannot handle lines longer than 1024
bytes
/unixdos/ud_ref.doc: UnixDos EGREP limit is 10,000 bytes.
*/unixdos/ud_ref.doc: EGREP *.doc*
/unixdos/pub/ud_usr.doc: egrep   Search text file(s) for one or
several pattern (search expression)
/unixdos/ud_usr.doc: The following UnixDos utilities use text searches:
* grep, egrep, fgrep, sed, ufind, umore, which.*

Text Search Expression:
For advanced text searches some special commands are available so you can tailor the expression to your particular needs.

The following special character can be used in an expression:
^   beginning of the line
$ end of the line
.   any single character
*   any number of characters
[list]   any character in the class
                (e.g. [a-z0123456789] all lower case letters and numbers)
[^list]   any character NOT matching this class
\c   escape   the special meaning of character 'c' and use as is
                (e.g. \$ looks for '$')
To learn how to create advanced search pattern see "Regular Expression for Text Searches" in chapter "ARGUMENT EXPANSION" in the "USER MANUAL".

Notes:
The UNIX GREP utilities do not have the capability to descend into subdirectories, while the UnixDos GREP make this powerful feature available.

The UNIX GREP utilities cannot handle lines longer than 1024 bytes while the UnixDos GREP utilities can handle lines up to 10,000 bytes.

The UNIX/MKS GREP utilities don't work well with binary files (like Microsoft Word Documents):
ud_ref.doc: grep: input lines truncated – result questionable
ud_usr.doc: grep: input lines truncated – result questionable
The UnixDos GREP utilities can handle also those binary files (since they break up the

lines on CarriageReturn's(CR) and null characters (NULL).

The UNIX GREP utilities will show the filename only when you specify at least two filenames. If you want to see the filename with only one file you have to append '/dev/null' to fool them into thinking it is processing two files to display the filename. This seemed rather awkward to me an so I added the '-a' and '-h' option.

env (Show/change environment/run command)

```
env [-] [NAME=value...] [cmd [argument(s)]]
-       = start from scratch (default uses existing environment)
NAME=   = update/insert new variable to environment
cmd     = command/program to be executed under the new
environment
arg     = optional arguments for the command

To avoid expansion of arguments enclose in single quotes:
env LIB=newlib uecho '$LIB'
```

Description:
ENV works with the process environment in the following ways:
• display all environment variables
• run a new process with a few changes to the existing environment
• run a new process with an environment which starts from scratch
• run a non-UnixDos program using UnixDos argument expansion feature

Display current Environment
To see all the currently defined environment variables enter:
```
env
```

Might display:
```
COMSPEC=C:\WINDOWS\SYSTEM32\COMMAND.COM
COMPUTERNAME=MADHUR
DBPATH=c:\v2deb\mhss
DEBUG=ON
HOMEDRIVE=C:
HOMEPATH=\users\default
INCLUDE=\madhur\src\ud\def;c:\msvc\include
INIT=C:\MSVC
LIB=c:\madhur\lib;c:\msvc\lib
LIBPATH=t:\pvcs\dos
OS=Windows_NT
PATH=c:\u;c:\dos;c:\windows
PROMPT=$t $p$g
TEMP=c:\temp
TMP=c:\temp
```

Change the existing set of environment variables
ENV allows you to run a program(or process) with an altered set of environment variables.
You just redefine all the new or changed environment variable names on the command line and then ENV will call the program based on the altered environment.
In our example we will use the "$LIB" environment variable.
To see the contents of a single environment variable enter:

```
uecho -d $LIB
```
The result might be:
c:\madhur\lib;c:\msvc\lib

Now we will call the UECHO program again, but with an altered LIB variable:
```
env LIB=newlib uecho -d '$LIB'
```
The result will be:
newlib
(Note the single quotes around '$LIB' to avoid too early argument expansion)
If you would not use the single quotes the $LIB environment would already be expanded BEFORE env is started:
```
env LIB=newlib uecho -d $LIB
```
The result will be:
c:\madhur\lib;c:\msvc\lib

Starting a new set of environment variables
ENV allows you also to run a program(or process) with a completely new environment.
In the following example we instruct ENV to start a new environment from scratch with the "-" sign. The we define the two variables "LIB" and "PATH" and call env again to now "see" the new environment:
```
env - LIB=newlib PATH=c:\dos env
```
The result will be:
LIB=newlib
PATH=c:\dos

Using the UnixDos argument expansion feature for non-UnixDos programs
Since ENV will expand the arguments before your program is called. This allows you to use the powerful UnixDos argument expansion even for non-UnixDos programs (like pkzip).
In the following example we will create an archive containing text files(*.txt)   files starting with "a" thru "c" and all documents (*.doc) NOT starting will "a" thru "s":
```
env pkzip archive [a-c]*.txt ![a-s]*.doc!*.doc
```
Might show:
Creating ZIP: ARCHIVE.ZIP
  Adding: ABC.TXT   Deflating (5%), done.
  Adding: COMM1.TXT   Storing   (0%), done.
  Adding: COMM2.TXT   Storing   (0%), done.
  Adding: TMAP.DOC   Deflating (64%), done.
  Adding: UD_REF.DOC Deflating (70%), done.
  Adding: UD_USR.DOC Deflating (71%), done.

Note: Unfortunately the primitive DOS operating system still allows only 128 bytes for all the
command and the program name. If ENV is hitting this limit it will display the following message:
env: Cannot run 'your program' (Arg list too long)
In this case try to reduce the number of arguments: `[a-m]*.c instead of *.c`

(u)exit (Exit with specified return code)

<u>Syntax:</u>
```
usage: uexit {return value}
```

<u>Description:</u>
UEXIT allows you to generate a specific program exit code.

<u>*Example:*</u>
To exit with error level 18 enter:
```
uexit 18
```

To permanently save this exit value enter:
```
saveexit
```

To show (and test) the error level enter (see TEST):
```
test -e? 0
```

Will display:
LAST EXIT VALUE WAS: 18

expr (Compute an expression)

```
expr [-f|-b|-o|-h] {ex1 op ex1}
-f   floating point:
     EXPR will use floating point arithmetic instead of the default
     32 bit integer arithmetic  and display the result with 7 places
     after the decimal point
-b   binary:
     Display the integer result as 32 bits (0 or 1) with a ","  between
bytes
-o   octal:
     Display the integer result as octal value with leading '0'
-x   hexadecimal:
     Display the integer result as hexadecimal with leading '0x'
ex1,ex2    value or expression:
     A numeric value (48) or a text string ("abc").
     EXPR will use the following conversions:
     0xNN=hexadecimal number (e.g. 0xF1B5), 0NN=ocatl (e.g 0174),
     0bNN=binary number (0b1010011)  otherwise decimal number
op   operand:
     One of the following arithmetic operands:
     +         add
     -         subtract
     '*' or x         multiply (remember to enclose * in quotes)
     /         divide
     %         remainder
     or one of the following conditional operands:
     =    or '-eq'   equal
     >=   or '-ge'   greater than or equal
     >    or '-gt'   greater than
     =    or '-ne'   not equal
     test if regular expression matches or one of the following
     logical operands:
     ^     logical OR
     &     logical AND
EXPR returns: 0 if expression is true or the numeric result is
not 0
              1 if untrue or the numeric result is 0, 2 if error
Note:
To avoid file expansion for multiplication use '*' or the x
operand
```

Description:
EXPR allows you to enter an expression on the command line like a calculator:
```
expr 100 + 200 / 5
```
Will show:

140

Note the automatic prioritizing to first calculate 200/5 (40) and then add 100:
100+40=140

Parenthesis:
You can also use parenthesis to specify a specific evaluation order:
`expr ( 100 + 200 ) / 5`
Will show (300/5):
60
Note the blanks around the parenthesis "(" and ")" to make then a separate argument!

Floating point display:
In some cases you might want to get a non-integer result. The "-f" option will switch to the floating point logic:
`expr -f 100 / 7`
Will display:
14.285714
Otherwise if you would enter: `expr 100 / 7` you would get:
14

Hexadecimal display:
In some cases you might want to display the integer result as a hexadecimal value.
The "-x" option will switch to the hexadecimal display:
`expr -x 100`
Will display:
0x64
This way you can easily convert from decimal to hex.

Binary display:
In some cases you might want to actually see the individual bits of an integer result.
The "-b" option will switch to the binary display:
`expr -b 256  + 8`
Will display:
00000000,00000000,00000001,00001000
This way you can easily "see" the bits (in this case bit 3 and 8).
The result is separated by commas into 4 groups of 8 bits each, making up the full 32-bit value.

Octal display:
In some cases you might want to display the integer result as a octal value.
The "-o" option will switch to the octal display:
`expr -o 100`
Will display:
0144
This way you can easily convert from decimal to octal.

Non decimal input
In the previous example we always used decimal value input. EXPR allows you also to

enter each value in hex/decimal or octal:
```
expr 100 + 0x64 + 0144 + 0b1100100
```
Will display:
```
400
```
In this example we entered the same value (100) in the 4 possible forms:
- 100   decimal
- 0x64   hexadecimal
- 0144   octal
- 0b1100100   binary

Arithmetic:
So far we use always used just simple addition (+), but EXPR can also do the following arithmetic operations:
- addition   +
- subtraction   -
- multiplication   *   (or "x" to avoid expansion)
- division   /
- remainder%

Examples:
```
expr 1 + 2      = 3
expr 10 - 8     = 2
expr 3 '*' 5    = 15   (we need the single quotes to escape argument
expansion!)
expr 3 x 5      = 15   (alternative multiplication which does not need single
quotes)
expr 100 / 5    = 20   (no rounding is done: expr 4 / 5 = 0)
expr 12 '%' 5   = 2    (we need the single quotes to escape argument
expansion!)
```

Comparison:
EXPR allows you also to compare two values (is 100 less than 5?):
```
expr 100 -lt 5
```
Will show:
```
0
```
The result "0" means NOT true.

```
expr 100 -gt 5
```
Will show:
```
1
```
The result "1" means true (100 is greater than 5)

The following comparisons are available:
- less than   -lt   or "<"   (escape "<" from operation system)
- less or equal   -le   or "<="   (escape "<=" from operation system)
- equal   -eq   or =
- not equal   -ne   or '!=' (the '!' needs to be escaped from argument

expansion)
- greater or equal -ge   or ">="   (escape ">=" from operation system)
- greater than   -gt   or ">"   (escape ">" from operation system)

Example:
To check if you have more than 10 C source files in your current directory enter:
```
ls *.c* | wc | grep TOTAL | cut -f1 > x1.tmp
expr 10 -gt @x1.tmp
```
If you see "0" you have NOT, if you see "1" you have more than 10 C source files.
To execute a command based on this criteria enter:
```
ls *.c* | wc | grep TOTAL | cut -f1 > x1.tmp
test ` expr 10 -gt @x1.tmp` -eq 1 run: banner "10 or more"
```

Joining conditions:
EXPR allows you also to join several conditions:
- logical AND   "&"
- logical OR   "^"

Example:
To test if 100 is greater than 80 AND 20 is less than 40 enter:
```
expr 100 -gt 80 "&" 20 "<" 40
```
Will show:
1

Exit Code:
EXPR use the following exit code depending on the result:
0   TRUE (condition was true or arithmetic computation yielded not 0 result)
1   UNTRUE (condition failed or arithmetic computation result is 0)
2   ERROR

Notes/Remarks:
The following features of UnixDos EPXR are not available in the UNIX/MKS EXPR:
- floating point arithmetic (-f option)
- Numeric input conversion (hex 0xNN, octal 0NN, binary 0bNN)
- Numeric output conversion (-x hex, -o octal, -b binary)
- comparison mnemonics (-gt,-ge,-eq,-ne,-lt,le)
- using the 'x' as an alternative for the '*' which always has to be escaped with quotes!

The string functions (substr, index, index) found in the Berkeley UNIX EXPR are not supported under UnixDos.

fgrep (Search files for one or more plain pattern)

Syntax:
```
fgrep [-abcdhilnpsvx] [-f {file}] [-e {p}] {pat} {file(s)}
-a      = always  : always show filename (even in a single file
search)
-b      = bytes   : show the byte offset in file of match
-c      = count   : show only count of matches
-d      = descend : search also descending subdirectories for
'file(s)
-e {p}  = expr.   : search pattern {p} (can be repeated!)
-f {f}  = file    : file with more pattern (can be repeated!)
-h      = hide    : don't show the filename
-i      = ignore  : upper or lower case
-l      = list    : show only the filename on match
-n      = number  : show line number of match
-s      = suppress: don't show error messages
-v      = inVert  : find all NON MATCHES
-x      = eXact   : find only exact match on whole line
{p}     = pattern : text to be matched (up to 1000 total
pattern)
        (only needed if neither '-e' not -f' is specified
file(s) = file(s) to be searched for match

(Maximum line length is 10000 bytes)
```

Description:
This is a excellent tool for searching files for text matches.
EGREP belongs to the following family of "grep" utilities:
• EGREP: can search for several expressions in the specified files
• FGREP: can search for several plain text items in the specified files
• GREP: can search for one single expression in the specified files

See the EGREP pages for more details.
FGREP is identical to EGREP except it does NOT allow regular expression for your search criteria.

Example:
Search all C files (*.c) for the lines matching '1044','1039', '1080' and '1099':
The file 'num.lst.' contains:
1044
1039
1080

To find these lines enter:
```
fgrep -f num.lst *.c
```

Notes:
See Notes under <u>EGREP</u>.

file (Determine file type)

```
file [-f {fl}] {file(s)}
-f {fl}   = use filelist in file 'fl
```

Description:
FILE will determine the file type based on the file extension and display the result on the screen.
FILE "knows" more than 100 different file types.

Example:
List all the file types of the files in the current directory:
```
file *
```
The result can look like this:
```
          /windows/archdll.dll: WINDOWS Dynamic Link Library file
           /windows/arches.bmp: WINDOWS bitmap file
           /windows/argyle.bmp: WINDOWS bitmap file
         /windows/artgalry.ini: WINDOWS initialization file
          /windows/artview.ini: WINDOWS initialization file
          /windows/artview.scr: WINDOWS Screen Saver file
           /windows/atikey.dll: WINDOWS Dynamic Link Library file
           /windows/atikey.exe: Executable program file
         /windows/atomtime.ini: WINDOWS initialization file
           /windows/attend.ini: WINDOWS initialization file
         /windows/audiosta.ini: WINDOWS initialization file
         /windows/autotext.sig: General Data file
            /windows/awcas.dll: WINDOWS Dynamic Link Library file
         /windows/awclass1.dll: WINDOWS Dynamic Link Library file
         /windows/awclass2.dll: WINDOWS Dynamic Link Library file
          /windows/awfaxio.dll: WINDOWS Dynamic Link Library file
         /windows/awfxprot.dll: WINDOWS Dynamic Link Library file
           /windows/awt30.dll: WINDOWS Dynamic Link Library file
               /windows/b.pif: WINDOWS program information file
                   bosada.txt: General text file
                   bosbis.txt: General text file
                   UnixDos.bak: General Backup file
                   dosnix.eps: Encapsulated postscript file
                   dosnix.img: WINDOWS image file
                     grab.exe: Executable program file
                       hs.cfg: General configuration file
                       hs.dat: General data file
                     menu.eps: Encapsulated postscript file
                       vp.bat: DOS Batch file
```

Notes:
The UNIX FILE does not use the file extension - instead it actually scans the file contents to make a guess about the file type which will always remain a guess.
That's why the UnixDos   FILE uses the extension which is faster and more reliable.

filename (Extract file name without extension)

Syntax:
```
filename [-d] [file(s)]
-d      = display in DOS style (default UNIX style)
file(s) = the filename(s) to be converted
If no 'file(s)' are specified filename will read STDIN and
convert each line
```

Description:
FILENAME will extract the file name without the extension from the specified path(s). You can use the UNIX display style(default) or specify DOS style display (-d option). You can either specify one or more filenames on the command line or provide a list of filenames in STDIN if you don't specify any filename.

Example1:
```
filename c:\tmp/save/src.zip dir1/file.c U:\PARK\file3.txt
```

Will show:
```
src
file
file3
```

Example2:
```
filename -d c:\tmp/save/src.zip dir1/file.c U:\PARK\file3.txt
```

Will show:
```
SRC
FILE
FILE3
```

Example3:
To find out all filenames used on your machine enter:
```
ufind / -type d -exec filename -d {} "^" > file_.lst
usort -u file_.lst  > file.lst
```

(u)find (Find files)

Syntax:
```
usage: ufind [directory or file(s)] [options]
-name {regexp} = match files by expression: -name '*.c' (C source
files)
-newer/-older {ref} = match if newer/older than reference file {ref}
-perm {perm}   = match files by permission: 444 read-only, 666:
read/write)
-size {[+|-]n[c|m]}  = match by file size:
                +/-  = equal or larger/smaller than specified
                n    = size in kilobytes (1024 bytes)
                c/m  = size in bytes/megabytes
-[a|c|m]time {n[m|h|y]} = match if modified within time frame
                n    = number of 24 hour intervals (days)
                m/h/y= minutes/hour/year intervals
-type {d|f}     = match d=directory, f=plain file
-exec {cmd}     = run 'cmd' for matches '{}' is replaced with filename
                end with ';' or "^": -exec cp -l {} \dir ^ (copy
files)
-grep {regexp}  = match file contents by expression
-ls             = print full info of matching file or directory
-ok {cmd}       = confirm operation for each file (if 'y' then like
-exec)
-print          = print just name of matching file or directory
-level {l}      = files up to directory level {l} (root=0, main
dir=1...)
-o=OR,-a=AND,!=NOT: ( -name '*.c' -o -name '*.h' ) show '*.c OR *.h
find / -print (prints all files on your current drive)
find / -exec cp -lv {} \dir; (calls 'cp -lv' for all files)
```

Description:

UFIND is an excellent tool to produce lists of filenames based an a certain criteria or run a program for each file found.

Note that the filename search (-name) is case insensitive, but pattern searches (-grep) are case sensitive!

Name Search (-name), Show Filename (-print)

To find all Batch(*.bat) files on your hard disk enter:

```
ufind / -name '*.bat' -print
```

Might show:

```
/autoexec.bat
/cvision/cvcopy.bat
/cvision/cvset.bat
/makeall.bat
/msvc/bin/fsample.bat
/msvc/bin/ftime.bat
/msvc/bin/hc.bat
/msvc/bin/lcount.bat
```

```
/msvc/bin/ltime.bat
/msvc/bin/msvcvars.bat
/msvc/mfc/samples/hiersvr/makehelp.bat
/msvc/mfc/samples/mksample.bat
/msvc/mfc/samples/oclient/makehelp.bat
/msvc/mfc/samples/scribble/step6/makehelp.bat
/msvc/mfc/samples/scribble/step7/makehelp.bat
```
In this example we instructed UFIND to use only the root directory (/) and descend down from there, use the regular expression (*.bat) to select files based on the name (-name) and print the just filename (-print) - the (-ls) option allows to also show the file details (size, date, etc.)

Advanced Name Search (-name) with regular expression
To find all '*.cpp' files with have the letter 'm' in their filename:
Example: "/mydir/aa**m**aa.cpp" but not "/mydir/aabb.cpp"

```
ufind / -name '*/*m[^/]\*.cpp' -print
```

Note the escaped '*' after [^/] to enforce repetition of that pattern instead of default '.'

Show all file details (-list)
To show find files in your /WINDOWS and /DOS directories enter:
```
ufind /dos /windows -ls
```

Might show:
```
-rw-rw-rw-     6404 Nov 11  1991 /dos/4201.cpi
-rw-rw-rw-      720 Nov 11  1991 /dos/4208.cpi
-rw-rw-rw-      395 Nov 11  1991 /dos/5202.cpi
-rw-rw-rw-     9065 May 31  1994 /dos/ansi.sys
-rwxrwxrwx    10774 May 31  1994 /dos/append.exe
-rw-rw-rw-     8369 Nov 11  1991 /dos/appnotes.txt
-rwxrwxrwx     6399 Nov 11  1991 /dos/assign.com
-rwxrwxrwx    11208 May 31  1994 /dos/attrib.exe
-rwxrwxrwx    12241 May 31  1994 /dos/chkdsk.exe
...
-rw-rw-rw-      545 Apr 19  1993 /windows/_default.pif
-rw-rw-rw-    14867 Sep 30 17:54 /windows/accessor.grp
-rw-rw-rw-     5408 Sep 04 17:18 /windows/apstudio.ini
...
-rw-rw-rw-     1312 Jan 07  1994 /windows/system/alger.fot
-rw-rw-rw-    69496 Apr 21  1993 /windows/system/alger.ttf
-rw-rw-rw-    99888 Mar 25  1994 /windows/system/anibuton.vbx
-rw-rw-rw-      440 May 15  1993 /windows/system/antqua.fif
-rw-rw-rw-     1320 May 13 16:58 /windows/system/antqua.fot
-rw-rw-rw-    59776 Aug 16  1994 /windows/system/antqua.ttf
...
-rw-rw-rw-     3679 Sep 30 17:44 /windows/system.ini
-rwxrwxrwx    10774 May 26  1995 /windows/system32/append.exe
-rwxrwxrwx    24640 May 26  1995 /windows/system32/attrib.exe
-rw-rw-rw-     1453 Aug 19 09:43 /windows/system32/autoexec.nt
-rwxrwxrwx    36092 May 26  1995 /windows/system32/backup.exe
-rw-rw-rw-    43520 May 26  1995 /windows/system32/backup.fts
-rw-rw-rw-    59555 May 26  1995 /windows/system32/backup.hlp
```

```
-rw-rw-rw-      11904 May 26  1995 /windows/system32/banana.ani
...
-rw-rw-rw-     338208 Jun 03  1980
/windows/system32/spool/drivers/w32x86/hppcl5ms.dll
-rw-rw-rw-      72976 Jun 03  1980
/windows/system32/spool/drivers/w32x86/rasdd.dll
...
drwxrwxrwx          0 May 10 16:23 /windows
```

In this example you can see how UFIND only uses the specified directories (/DOS) and (/WINDOWS) and then descends into all subdirectories in alphabetic order (unlike UNIX/MKS!).
The (-ls) option show the file details in LS style (see "LS").

Find files after/before a certain date and time point (-newer/-older/-NEWER/-OLDER)
This option allows you to search for files based on a certain time point.
You need one "reference file" as a time reference.
The following options are available:
- -newer   after reference file
- -NEWER   after of equal to reference file
- -older   before reference file
- -OLDER   before of equal to reference file

In this example we first create a "reference file" named "/ref" with a time stamp for Sept 18, of the current year with a time of 12:00 noon:
```
touch 09181200 /ref
```

Just to verify that this new files exists we enter:
```
ls -l /ref
```
And should see:
```
-rw-rw-rw-          0 Sep 18 12:00 /ref
```

Now we can instruct UFIND to use your "reference file" (/ref) and find all files which are newer (or have the same time) and show all file details (-ls):
```
ufind / -NEWER /ref -ls
```

Find files/directories up to a specified level/depth (-level)
This option allows you to limit the search to a particular depth of files and directories.
The root directory has level 0, the main directories and files   in the root directory have level 1 and so forth.

To find all the main directories in your root directory enter:
```
ufind / -type d -level 1 -print
```

Might display:
```
/dos
/msoffice
/msvc
/windows
```

. . .
Find files within a time frame from now (-time)
This option allows you to search for files based on a specified age relative to the current time.
The following specify the interval in the following units:
- years   {n}y
- days   {n}   (default)
- hours   {n}h
- minutes   {n}m

In the following example we show all files created or changed within the last 10 hours:
```
ufind / -time 10h -ls
```

You can use the negation to invert the time frame.
In this example we show all files which are older than 3 years :
```
ufind / '!' -time 3y -ls
```
Note the single quotes around ! to avoid argument expansion.

Copy all files with confirmation which have been modified in the last week (7 days):
```
ufind / -time 7 -ok cp -lv {} \week ;
```

Find files of a certain size (-size)
This option allows you to search for files based on their file size.
You can specify larger "+{val}" or smaller than "-{val}".
You can specify the threshold file size in the following increments:
- 1k blocks (1,024 bytes) (default)   -50   = less than 51,200 bytes
- megabytes   +2m   = more than 2,000,000 bytes
- bytes   -180000c   = less than 180,000 bytes

In this example we look for all files on your entire hard disk which are larger than one megabyte (1,000,000 bytes):
```
ufind / -size +1m -ls
```

Might show:
```
-rw-rw-rw-  1281701 Oct 07 15:37 /all.lst
-rw-rw-rw-  4325728 Aug 16  1994 /msoffice/access/msacc20.hlp
-rw-rw-rw-  1326516 Aug 16  1994 /msoffice/access/msaccess.cue
-rwxrwxrwx  1909024 Aug 16  1994 /msoffice/access/msaccess.exe
-rw-rw-rw-  2097152 Oct 07 14:17 /msoffice/access/wzfrmrpt.mda
-rwxrwxrwx  4246784 Aug 16  1994 /msoffice/excel/excel.exe
-rw-rw-rw-  2992100 Aug 16  1994 /msoffice/excel/mainxl.hlp
-rwxrwxrwx  3490816 Aug 16  1994 /msoffice/winword/winword.exe
-rw-rw-rw-  2006714 Aug 16  1994 /msoffice/winword/winword.hlp
-rw-rw-rw-  1895620 Sep 16  1994 /msvc/bin/vb.hlp
-rw-rw-rw-  2738544 Jan 13  1995 /msvc/help/mfc.hlp
-rw-rw-rw-  1578988 Sep 16  1994 /msvc/help/vcbks15.hdx
-rw-rw-rw- 20322482 Sep 16  1994 /msvc/help/vcbks15.hlp
-rw-rw-rw- 15581184 Sep 16  1994 /msvc/help/vcbks15.ind
-rw-rw-rw- 16610031 Sep 16  1994 /msvc/help/vcsdk15.hlp
-rw-rw-rw-  3386753 Sep 16  1994 /msvc/help/win31wh.hlp
```

```
-rw-rw-rw-  1505064 Feb 08  1993 /windows/system/mfc200d.dll
-r--r--r--  1914140 Sep 09 04:57 /windows/system/mfc250d.dll
-rw-rw-rw- 20971520 Sep 30 17:46 /windows/win386.swp
```

Execute a program for each file found (-exec and -ok)
This option allows you to run a program on each file found in the search.
The filename is referred to by the symbol: "{}"
The "-exec" option will execute without prompting, while the "-ok" will prompt you before executing the program for each file.

In this example we will produce a cleanup procedure which removes all temporary files (*.tmp) from the system. We search for all files with the extension "*.tmp" and execute the UnixDos RM utility "rm" with the force and list options "-lf". The current filename is referred to by "{}". The constructed command line has to be terminated with the ";" (or "^") character:
```
ufind / -name '*.tmp' -exec rm -lf {} ;
```

Might show:
```
/src/x1.tmp
/roma/rfapps/x1.tmp
/roma/rmbase/x1.tmp
/roma/rmmgr/x1.tmp
/roma/src.tmp
/src/mail.tmp
/stdtools/mail.tmp
/temp/mail.tmp
/temp/pwd.tmp
/temp/src_get.tmp
/temp/~df148a.tmp
/temp/~df148c.tmp
/temp/~df148d.tmp
/temp/~df148e.tmp
/temp/~df1529.tmp
/temp/~wrf0000.tmp
/temp/~wrs0004.tmp
```

To run the same program but with confirmation prompts enter:
```
ufind / -name '*.tmp' -ok rm -lf {} ;
```

Might show:
```
< rm -lf /madhur/doc/~wrl0003.tmp  >? n
< rm -lf /temp/~df148a.tmp  >? y
/temp/~df148a.tmp
< rm -lf /temp/~df148c.tmp  >? n
< rm -lf /temp/~df148d.tmp  >? n
< rm -lf /temp/~df148e.tmp  >? y
/temp/~df148e.tmp
< rm -lf /temp/~df1529.tmp  >? n
< rm -lf /temp/~wrf0000.tmp  >? n
< rm -lf /temp/~wrf0001.tmp  >? n
< rm -lf /temp/~wrs0004.tmp  >? n
```

Find files with specified pattern/regular expression (-grep)

This option allows you to search for a specific pattern within each selected file and display hits. You could also run UnixDos GREP using the "-exec" option, but the internal grep facility within UFIND is obviously much faster!

In this example we will search for the pattern "EGREP" in all documents (*.doc) in the machine:

```
ufind / -name '*.doc' -grep EGREP
```

Might show:

```
/madhur/doc/ud_ref.doc: EGREP: can search for several expressions in the
specified
/madhur/doc/ud_ref.doc: EGREP/FGREP/GREP will exit with the following exit
status:
/madhur/doc/ud_ref.doc: To search for EGREP in all word documents enter:
/madhur/doc/ud_ref.doc: grep -d EGREP *.doc
/madhur/doc/ud_ref.doc: ufind / -name "*.doc" -grep EGREP
/madhur/doc/~wrl0390.tmp: EGREP belongs to the following family of ôgrepö
utilities:
/madhur/doc/~wrl0390.tmp: EGREP/FGREP/GREP will exit with the following exit
status:
/madhur/doc/~wrl0390.tmp: If you only want to know specifically how many
matches occured within each file, but not actually see all the matches this
option allows you instruct EGREP/FGREP/GREP to provide this information
/madhur/doc/~wrl0390.tmp: The UNIX EGREP cannot handle lines longer than 1024
bytes
/madhur/doc/~wrl0390.tmp: See the EGREP pages for more details.
/madhur/src/src_ud.zip: UD/UNIX/EGREP.LST
/madhur/src/src_ud.zip: UD/UNIX/EGREP.CP^A^B^T
/madhur/src/ud/pub/ud_ref.doc: See EGREP.
/madhur/src/ud/unix/egrep.mak: CEGREP = cl
/madhur/src/ud/unix/egrep.mak: FIRSTC = EGREP.C
/temp/~df152c.tmp: ufind / -name "*.doc" -grep EGREP
```

Find only directories or plain files (-type)
This option allows you to search for directories or only for plain files.

In this example we will search for the directories in the "/DOS" and "/WINDOWS" directories:

```
ufind /dos /windows -type -d -print
```

Might show:

```
/dos
/windows/msapps/msgraph
/windows/msapps/msquery
/windows/msapps
/windows/msmail/msmail
/windows/msmail
/windows/repair
/windows/system
/windows/system32/config
/windows/system32/drivers/etc
/windows/system32/drivers
/windows/system32/spool/drivers/w32x86/1
```

```
/windows/system32/spool/drivers/w32x86
/windows/system32/spool/drivers
/windows/system32/spool/printers
/windows/system32/spool/tmp
/windows/system32/spool
/windows/system32
/windows/temp
/windows
```

Find all read-only or read-write files (-perm)
This option allows you to search for files with a particular permission value:
- read-only files   `-perm 444`
- read/write files `-perm 666`

This might be helpful if you work in an environment where you are checking out individual work files from a central archive depository. Once you have "checked out" a file it becomes read/writable so that you can make changes - otherwise the files are read/only.

In this example we will search for all C files which are read-only
```
ufind /src ( -name '*.cpp' -o -name '*.hpp' ) -perm 444 -ls
```

Might show:
```
-r--r--r--    14099 Oct 04 10:37 /src/dbapps/cfuncs.cpp
-r--r--r--     2346 Aug 13 15:58 /src/dbapps/cfuncs.hpp
-r--r--r--     3348 Jun 19 15:26 /src/dbapps/dbapinit.cpp
-r--r--r--      293 Aug 15  1994 /src/dbapps/dbapinit.hpp
-r--r--r--    42901 Apr 10  1996 /src/dbapps/dbexpedt.cpp
-r--r--r--    21007 Jan 08  1996 /src/dbapps/dbexpedt.hpp
-r--r--r--      520 Jul 14  1994 /src/dbapps/gendefs.h
-r--r--r--    11195 Feb 09  1996 /src/rfapps/tangrsrl.cpp
-r--r--r--     3053 Feb 09  1996 /src/rfapps/tangrsrl.hpp
-r--r--r--    29929 Jul 13 11:03 /src/rmbase/dbcalc.cpp
-r--r--r--    13068 Jul 13 11:05 /src/rmbase/dbcalc.hpp
-r--r--r--     2340 Jul 13 11:02 /src/rmbase/dbcavg.cpp
-r--r--r--     3326 Jul 13 11:05 /src/rmbase/dbcavg.hpp
-r--r--r--     1584 Jul 13 11:03 /src/rmbase/dbccount.cpp
-r--r--r--     2628 Jul 08 11:48 /src/rmbase/dbccount.hpp
-r--r--r--     4360 Oct 04 12:02 /src/rmmgr/sax_main.hpp
-r--r--r--      465 Mar 30  1995 /src/rmmgr/stdafx.h
```

Nesting search criteria (-a and -o and "(" and "))
This option allows you to combine and nest several individual search items.
The following two operators are available:
- AND   -a
- OR   -o
- Start Group   (
- End Group   )
- Negate   '!'   (single quotes to avoid early argument expansion)

In the following example we are searching for all C source files (*.c) or C header files

(*.h) starting from the root directory "/". Then we start a group with the open parenthesis "(" and specify the first search criteria: -name '*.c', followed by the OR operator "-o" and the second search criteria: -name '*.h'. Then we have to close the group with ")" and to see the results we use the "-ls" show all file details command:

```
ufind / ( -name '*.c' -o -name '*.h' ) -ls
```

(Note the single quotation marks in '*.c' and '*.h' to let UFIND doing the expansion instead of the default argument expansion process!)

Might show:

```
-rw-rw-rw-      8694 Oct 18  1994 /bchkw/demo/animate.c
-rw-rw-rw-     22106 Oct 18  1994 /bchkw/demo/bitmap.c
-rw-rw-rw-     15801 Oct 18  1994 /bchkw/demo/demo.c
-rw-rw-rw-      3437 Oct 18  1994 /bchkw/demo/pre.c
-rw-rw-rw-     17569 May 02  1995 /bchkw/test.c
-rw-rw-rw-      1473 May 02  1995 /bchkw/test.h
-rw-rw-rw-       260 Jun 06  1994 /cvision/demo.c
-rw-rw-rw-     15116 May 25  1995 /cvision/hreport.c
-rw-rw-rw-       169 Jan 09  1995 /cvision/lib-bcl.h
-rw-rw-rw-      1824 Nov 04  1994 /cvision/show.c
-rw-rw-rw-     32778 Mar 29  1994 /cwright/brief/brief.c
-rw-rw-rw-      3076 Mar 29  1994 /cwright/brief/brief.h
-rw-rw-rw-     14861 Mar 29  1994 /cwright/brief/briefini.c
-rw-rw-rw-     28856 Mar 29  1994 /cwright/cua/cua.c
-rw-rw-rw-     17783 Mar 29  1994 /cwright/cua/cuaini.c
-rw-rw-rw-     36920 Mar 29  1994 /cwright/cwhelp/cwdata1.c
-rw-rw-rw-     26027 Mar 29  1994 /cwright/cwhelp/cwdata2.c
-rw-rw-rw-     10584 Mar 29  1994 /cwright/cwhelp/cwdata3.c
-rw-rw-rw-     24431 Mar 29  1994 /cwright/cwhelp/cwhelp.c
-rw-rw-rw-      2782 Mar 29  1994 /cwright/cwhelp/cwhelp.h
-rw-rw-rw-     11307 Mar 29  1994 /cwright/cwhelp/cwhelpid.h
```

To invert the last search (all files except '*.c' and '*.h' files) enter:

```
ufind / '!' ( -name '*.c' -o -name '*.h' ) -ls
```

(Note the single quotation marks in '!', '*.c' and '*.h' to let UFIND doing the expansion instead of the default argument expansion process!)

Notes/Remarks:
The following UnixDos UFIND features are not available under UNIX/MKS:
• Search file contents for pattern (-grep option)
• file   LS style display (-ls option) (only in Berkeley UNIX)
• the megabyte qualifier for filesize search (-size option)
the minute(m), hour(h) and year(y) qualifier for time searches (-time option)

The following UNIX options do not apply to DOS:

```
-type b/c/p/l/s, -links, -[no]user, -[no]group, -inum, -xdev
```

getlines (Get range of lines from a file)

Syntax:
```
getlines {start} {end} {file}
{start} = starting line number
{end}   = ending   line number
{file}  = file to read
The errorlevel is 0 if lines are read, 1=if line out of range,
2=error
```

Description:
GETLINES allows you to extract the specified range of lines from a text file. This might be useful when you have a very big file and want to extract a particular subset.
To find specific line numbers use "grep -n".
The exit code depends on:
- 0   lines(s) are in range
- 1   line(s) are out of range
- 2   other errors

Example:
In the following example we extract one line at a time from the list of files "test_lst.txt":
```
x1.bat
x2.bat
x3.bat
x4.bat
cc.bat
go.bat
```
Here is the batch file performing this task using GETLINES to extract one lines and also to determine if all lines have been read via the exit code 1 of EOF (see "test_lst.bat"):
```
@echo off
set LINE=1
set LST=test_lst.txt
:LOOP
    getlines %LINE% %LINE% %LST% > name.tmp
    if errorlevel 1 goto END

    call setenv "NAME=@name.tmp
    echo %LST% LINE %LINE% = %NAME%
    call setenv "LINE=`expr $LINE + 1`"
    goto LOOP

:END
del name.tmp
```

Will show:
```
test_lst.txt LINE 1 = x1.bat
test_lst.txt LINE 2 = x2.bat
test_lst.txt LINE 3 = x3.bat
```

```
test_lst.txt LINE 4 = x4.bat
test_lst.txt LINE 5 = cc.bat
test_lst.txt LINE 6 = go.bat
```

grep (Search files for one pattern/expression)

Syntax:
```
grep [-abcdilnpsvwx] [-e {pattern}|{pattern}] {file(s)}
-a      = always  : always show filename (even in a single file
search)
-b      = bytes   : show the byte offset in file of match
-c      = count   : show only count of matches
-d      = descend : search also descending subdirectories for 'file(s)
-h      = hide    : dont show the filename
-i      = ignore  : upper or lower case
-l      = list    : show only the filename on match
-n      = number  : show line number of match
-s      = suppress: dont show error messages
-v      = inVert  : find all NON MATCHES
-w      = Word    : find pattern as a word (no alphanumeric around
match)
-x      = eXact   : find only exact match on whole line
-e {p}  = expr.   : search pattern {p} (only if {p} start with '-')
pattern = text or regular expression to be matched
NOTE: to disable the argument expansion use single quotes: grep '*.c'
*.c
file(s) = file(s) to be searched for match
grep -d bak *.bat :search for 'bak' in all batch files also in sub
directories
grep -n -i abc *.c:search for 'abc' in '*.c' files, show line# and
ignore case
(Maximum line length is 10000 bytes)
```

Description:
This is a excellent tool for searching files for text matches.
EGREP belongs to the following family of "grep" utilities:
- EGREP: can search for several expressions in the specified files
- FGREP: can search for several plain text items in the specified files
- GREP: can search for one single expression in the specified files

See the EGREP pages for more details.
GREP is identical to EGREP except it can only search for one single expression.

GREP will exit with the following exit status:

0   Found match(es)
1   Found no match
2   Syntax error or inaccessible file

Example1:
Search for the pattern 'main' in all C source files in the current directory:
```
grep main *.c
```

Example2:
Show only how many lines match the pattern '1044':
```
grep -c 1044 numbers.lst
```

Example3:
Show only lines which start with 'main':
```
grep  '^main' *.c
```

Example4:
Show the line number and ignore the case of pattern 'main' (can be Main,MAIN,main):
```
grep -ni 'main' *.c
```

Example5:
Look for any line with a capital letter followed by a lower case letter:
```
grep '[A-Z][a-z]*' *.c
```

Example6:
Show all lines which don't contain "main":
```
grep -v main *
```

Example7:
Look for all lines containing a '*' (escape special meaning of '*'):
```
grep '\*' *.c
```

Notes/Remarks:
The UnixDos GREP will always display the filename with the '-a' while the UNIX GREP will show the filename only when you specify at least two filenames. If you want to see the filename with only one file you have to append '/dev/null' to fool GREP into thinking it is processing two files to display the filename which seems rather awkward.

The UNIX GREP cannot handle lines longer than 1024 bytes while the UnixDos GREP limit is 10,000 bytes.

head (Display top lines of a file)

Syntax:
```
head [-l] [-{n}] [file(s)]
-{n}    = number of lines to be displayed (default is 10)
-l      = list: precedes line with filename
if no files are specified stdin is used
(Maximum line length is 10000)
```

Description:
HEAD will show the first lines of all specified files (default is the first 10 lines).
If no files are specified HEAD will use STDIN.
Lines can be up to 10,000 bytes long.

Examples:
Display the first 4 lines of all C source files in the current directory:
```
head -4 *.c
```

Display also   the filenames:
```
head -l -4 *.c
```

Notes:
The UNIX HEAD does not provide the list option (-l).

inpbox (Show message and prompt for input)

Syntax:
```
inpbox -f{mf} [-e{ef}]
-f{mf} = Define the input definition file containing the details
-e{ef} = Define the file were to write the exit code
```

Description:
INPBOX allows you to bring up a box containing a message with and edit window allowing the user to enter some data from the command line.
The details about the requested operation are defined in the the input definition file.

Exit Code:
The exit code reflects the users response.
It is written as a decimal value to the specified file using the '-e' option.
If no file is specified '%TEM%/inpbox.tmp' is used.
The following exit values are available:.
1   First button clicked
2   Second button clicked or Esc key
3   Third button clicked
4   Fouth button clicked

Input Definition File:
The INPBOX message file contains several sections specifying the desired behaviour:
It has the following three sections:
• Title (defining the box title)
• Edit   (defining the behaviour of the data input/edit box)
• Data (defining the data exchange parameter)
• Button (defining the buttons)
• Text (defining style of the message text)
• Message (defining the actual message text)

Title:
```
TITLE={text of title}
```
This instruction is defining the title for this message box.
It can be up to 200 characters long.
If no title is specified "Input Box" is used.
Example: `TITLE=Enter filename`

Data:
```
DATA={output file}[|initial data]
```
This instruction is defining to which file the entered data is written.
You can also specify the initial text in the edit window here.
Example: `DATA=c:/temp/getdata.tmp|enterhere`

Button:
```
BUTTON={b1[,b1[,b3[,b4]]] | DEFBUTTON{n}
```
This instruction is defining how many buttons are available and the text on each button.
Also you can define which of there buttons is the default button (if any).
Example: `BUTTON=&Yes,&No,May&Be|DEFBUTTON2`
Will create three buttons "<u>Y</u>es", "<u>N</u>o" and "May<u>B</u>e".
The "<u>N</u>o" button is the default.

Text:
TEXT={CENTER|LEFT|RIGHT}
This instruction is defining in which style the mesaage text is displayed.
Left is the default.
Example: `TEXT=CENTER`

Edit:
EDIT={Flag1|Flag2|...}
This instruction defines the flags related to the input/edit control.
You can specify several flags separating with the pipe character:
Here are all the available codes:
```
UPPERCASE|LOWERCASE = convert to case
NUMBER              = allow numbers
OFF                 = don't allow edit
```

Message:
All the other lines will become part of the actual message

Example:
Below is an example of a input file 'inp1.txt':
```
BUTTON=&Yes,&No,&Edit,&Go|DEFBUTTON3
TEXT=LEFT
EDIT=UPPERCASE
Here
TITLE=My Title
DATA=\tmp\x1|edit here
is
my
meeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeesage!
!
Do you want to continue?
```

To activate a input box with these parameters use:
```
inpbox -finp1.txt -e%TEMP%/myexit.tmp
```

Once the use has responded you will find in '`%TEMP%/myexit.tmp`' the exit code (1-4)

io (Input/output file areas)

Syntax:
```
io (Input/Output copy)
usage: io {input} {output} {start} {length} [offset]
input     = input file
output    = output file
start     = offset in input file (in bytes)
length    = number of bytes to copy (in bytes)
offset    = offset within output file before copying (in bytes)
```

Description:
IO allows you the extract from any file a particular area and "move" it to a new file to any position.

Example:
Extract the third megabyte from 'big1' and write to the output file 'out' starting at offset 500,000:
```
io big1 out 2000000 1000000 500000
```

Remarks:
This utility is a little easier to use than the "dd" "iseek/oseek/keep" command set.

(u)join (Join two sorted text files via matching field)

Syntax:
```
ujoin [-a{f}] [-e{s}] [-j{f} {n}] [-t{c}] [-o {l}] {f1} {f2}
-a{f}    = add   : show also lines of unpaired line in file {f}
                   f=1: in file1, f=2: in file2, f=3: in both
files
-e{s}    = empty : replace empty output fields with {s}
-j{f}{n} = join  : Specify the joining field
                   (default is first field in both files):
                    -j1 f#   Set joining field in file1 to
field 'f#'
                              (1 is the first field)
                    -j2 f#   Set joining field in file2 to
field 'f#'
                    -j fld   Set joining field in both files to
'f#'
-t{c}    = tab   : use {c} as field separator (default is
blank/tab)
         (You can also specify the hex value of the character:
-t\7C)
-o {l}   = output: specify output fields and their sequence in
{l}
                   {l} = {f}.{n} [{f}.{n}....]
                    f   file reference (1=file1, 2=file2)
                    n   field number (1 is the first field)
                   (first argument without leading number ends
list)
f1       = input file1 (if '-' use stdin)
f2       = input file2
{f}      = file number, 1=first file or 2=second file
{n}      = field number, starting with 1
```

Description:
UJOIN will compare two sorted text files and join lines with the same "key". Each line contains one or more fields which are separated by the field separator. The '-t' option allows you to change this field separator from blank/TAB to any other character.

UJOIN will use the first field in both files by default to determine if the line in the files match. You can specify different join fields with the '-j' option.

UJOIN will print the common field, then the remaining fields in file1 and then the remaining fields in file2. You can specify a different output field sequence using the '-o' option.

If an output field is blank UJOIN will usually also print nothing for that field unless you specify a replacement string using the '-e' option.

Examples:
File 'join1.txt' contains the sequence number, customer number and the name:
```
1|101|SIM SMITH
2|213|JOAN LEAPER
3|224|TERRY COUPER
4|224|
```

File 'join2.txt' contains the customer number and the city:
```
101|DENVER
213|LOS ANGELES
224|NEW YORK
```

The following command will join the two files based on the customer number:
```
ujoin -e (name) -o 1.2 1.3 1.1 2.2 -j1 2 -j2 1 -t\7c join1.txt
join2.txt
```

Will produce the following output:
```
101|SIM SMITH|1|DENVER
213|JOAN LEAPER|2|LOS ANGELES
224|TERRY COUPER|3|NEW YORK
224|(name)|4|NEW YORK
```

The arguments in the example:
```
-e (name)
```
The 4th line in 'join1.txt' has no customer name - this option will insert '(name)' instead.

```
-o 1.2 1.3 1.1 2.2
```
Specifies the output fields:
1.2	field2 in file1
1.2	field3 in file1
1.1	field1 in file1
2.2	field2 in file2

```
-j1 2
```
specifies that field2 is the joining field in file1
```
-j2 1
```
specifies that field1 is the joining field in file1
```
-t\7c
```
Replaces the field separator with the pipe '|'
(Older DOS versions don't allow you to escape the pipe character, so here is an example of the hex character notation: 0x7C = '|')

Example2:
```
ujoin  -j1 2 -j2 1 -t\7c \u\join1.txt \u\join2.txt
```

Will produce:

```
101|1|SIM SMITH|DENVER
213|2|JOAN LEAPER|LOS ANGELES
224|3|TERRY COUPER|NEW YORK
224|4||NEW YORK
```
Notice   the different field sequence and the missing name on line four.

kill (Terminate Process)

```
usage: kill [-c] [-f] [-y] [-w] {PID1|Search1} {PID2|
Search2} ...
-c      = Case   : don't ignore case in text searches
-f      = Force  : kill ungracefull (no Close message)
-m{m,w} = Message: define new message:
                   m = message (0x=hex, 0=oct, 0xb=binary
rest=decimal)
                   w = wParam  (0x=hex, 0=oct, 0xb=binary
rest=decimal)
-w      = Window : use window handles instead of PID's
-y      = YesNo  : confirm each kill with yes/no
PID     = Process ID (or HWND with -w option):
          0xFFFFFFFF = hexadecimal number
          0777777777 = octal number
          0b01010101 = binary number
Search = Search pattern
```

Description:
KILL will attempt to terminate the specified process.
You can either use a more gracefull approach by sending a window close message (default) or be more forcefull by just terminating (-f option) and not giving the process achange to close up.

You can specify the process(es) by:
• PID (Process ID number) or
• HWND (Window Handle) or(or
• search pattern for the window title of the process (case insensitive and sensitive)
You can specify one or several preocesses by PID or search patterns.
You can also be more specific and specify one window using the window handle (HWND) with the -w option. To get the details of all processes use the PS command.
The text search is case insensitive unless you specify the '-c' option.
You can enter PID's at hexadecimal numbers (0x6A) or octal/binary/decimal numbers.
You can specify a different message to be sent to the process(es) with the -m option in case the application needs a different signal than the default WM_SYSCOMMAND(0x112) and SC_CLOSE (0xF060), which is equal to specifying: -m0x112,0xF060 on the comand line.

Example1 (confirm termination)
Confirm termination of process running Microsoft "word" :
```
kill -y word
```

Example2 (several processes)

Kill sevaral processes by PID without confirmation:
```
kill 0x6A 0xFFFB0631 word
```

Example3 (window handle)
Kill the windows with the handle 0x3a0 and all processes containing "word"
without any confirmation:
```
kill -w 0x3a0 word
```

Example4 (send a different close message)
Kill the windows with the handle 0x3a0 and all processes containing "word"
without any confirmation, but use WM_COMMAND(0x111) and wParam decimal 40003
to close:
```
kill -m0x111,40003 MyApplication
```

line (Read one line from user/keyboard)

Syntax:
```
line [-{l|u}]
-l   = convert input to lower case
-u   = convert input to upper case
```

Description:
LINE will read one line from the keyboard. By default it will display the input as it was entered on the screen and write it to STDOUT.

The '-u' and '-l' will convert the letters accordingly.

Example:
Confirm deletion of all files in /tmp directory:
```
ucd /tmp
uecho "Delete all file in /tmp? (Y/N): \c"
test "`line -u` ~= "Y" run: "rm -rfl *"
```

ls (List files)

Syntax:
```
ls [-CFRdelrstxy] [files]
-C = COLUMN   : suppress multi column output
-F =          : suppress '/' for directories, '*' for programs
-R = RECURSIVE: list also all subdirectories
-d = DIRECTORY: show only directory detail NOT all files in it
-e = EXTEND   : show filesize in extended format (30,764,678)
-l = LONG     : show all file details:
                {d|-}rwxrwxrwx {size} {date/time} name
                 d  directory
                 -  plain file
                 r  read permission
                 w  write permission
                 x  execute permission (only programs)
-r = REVERSE  : reverse the current sort order
-s = SIZE     : sort by file size (show 1k increments)
-t = TIME     : sort by time
-x = ACROSS   : sort across lines instead of down the column
-y = TYPE     : sort by file type/extension
(If the output is redirected to a file LS automatically prints
each file
on a separate line: ls > lst.tmp
(default sort is by filename)
```

Description:
LS will display details about the specified files or directories.
LS will list all the files for specified directories.
LS without any options will list all the files in the current directory sorted by name.

Example1:
List entire disk:
```
ls -lR /
```

Example2:
Display files in current directory sorted by size:
```
ls -s
```

Example3:
Display in reverse time order with long list:
```
ls -ltr
```

mak2mk (Convert Visual C Makefile to UnixDos makefile)

Syntax:
```
mak2mk (Convert Visual C makefile (*.mak) to UnixDos makefile
(*.mk))
usage: mak2mk [-c] {mak file}
-c    = CLEANUP: remove also the error files (*.er*)
```

Description:
MAK2MK will convert the automatically generated Visual C makefile into the UnixDos makefile. The UnixDos makefile will use the CC front end (cc.bat/cc1/cc3) to compile your modules and catch the errors and warning into a separate error file.

(see "CC" documentation for details)

Example:
To convert a makefile (in this example "llib.mak) enter:
```
mak2mk llib.mak > llib.mk
```

Now we use this UnixDos makefile (llib.mk) to compile the modules.
It will use the UnixDos   CC front end so that the screen/output is not cluttered with compiler options interspersed with the diagnostics:
```
nmake /nologo DEBUG=1 -f llib.mk
```

The screen would now show something like:
```
        cc -c add_func.c                          # LLIB
        cc -c bgetl.c                             # LLIB
        cc -c bname_t.c                           # LLIB
        cc -c bread.c                             # LLIB
        cc -c bwrite.c                            # LLIB
        cc -c chrchk.c                            # LLIB
        cc -c cmp_file.c                          # LLIB
        cc -c cntdel.c                            # LLIB
        cc -c cntdeler.c                          # LLIB
        cc -c crbe.c                              # LLIB
        cc -c ctyp.c                              # LLIB
        cc -c debfun.c                            # LLIB
        cc -c dir.c                               # LLIB
        cc -c do_arg.c                            # LLIB
        cc -c dx_chdir.c                          # LLIB
        cc -c dx_getdi.c                          # LLIB
        cc -c fclose_.c                           # LLIB
        cc -c fgetl.c                             # LLIB
        cc -c fldcmp.c                            # LLIB
        cc -c fn_cp.c                             # LLIB
        cc -c fopen_.c                            # LLIB
        cc -c get_fram.c                          # LLIB
        cc -c get_free.c                          # LLIB
```

```
        cc -c getopt.c                                 # LLIB
        cc -c get_fram.c                               # LLIB
        cc -c get_free.c                               # LLIB
        cc -c getopt.c                                 # LLIB
        cc -c getsize.c                                # LLIB
getsize.c(34) : warning C4101: 'NotUsed' : unreferenced local variable
getsize.c(29) : error C2065: 'Error' : undeclared identifier
getsize.c(29) : error C2146: syntax error : missing ';' before identifier
'here'
getsize.c(30) : error C2065: 'here' : undeclared identifier
getsize.c(30) : error C2143: syntax error : missing ';' before 'type'
getsize.c(33) : error C2065: 'statbuf' : undeclared identifier
getsize.c(33) : warning C4049: 'argument' : indirection to different types
getsize.c(33) : error C2224: left of '.st_size' must have class/struct/union
type
        cc -c htoi.c                                   # LLIB
        cc -c htol.c                                   # LLIB
        cc -c llib.c                                   # LLIB
        cc -c mem.c                                    # LLIB
        cc -c otol.c                                   # LLIB
        cc -c pk.c                                     # LLIB
        cc -c prin.c                                   # LLIB
prin.c(50) : warning C4013: 'system' undefined; assuming extern returning int
        cc -c qsort_.c                                 # LLIB
        cc -c read.c                                   # LLIB
        cc -c regexp.c                                 # LLIB....
```

Now you can easily spot diagnostics and see the module name.
All diagnostics are also captured in the "error C files" (in this case 'getsize.erc' and 'prin.erc').
To detect all the modules which compiled with errors or warnings enter:
```
wc -nl *.erc
```

The screen would now show something like:
```
        8 getsize.erc
        1 prin.erc
        9 TOTAL
```

M4 (Advanced Text Generator)

Syntax:
```
usage:
m4    [-o{f}] [-t] [-Bn] [-Dn=v] [-Hn] [-Sn] [-s] [-Tn] [-Un] [fn1 [fn2]...]
m4cgi [-o{f}] [-t] [-Bn] [-Dn=v] [-Hn] [-Sn] [-s] [-Tn] [-Un] [fn1 [fn2]...]
fn1.. = input filename(s) (if none specified 'm4' will use
STDIN)
-o{f} = OUTPUT   : send all output to file 'f' instead of STDOUT
-t    = TRACE    : switch trace by default (for debugging)
                   (default is OFF unless you use 'traceon'
macro
-Bn   = BLOCKSIZE: specify the block size (default = 4096)
-Dn=v = DEFINE   : define symbol 'n' to value 'v'
-Hn   = HASH     : set hash size to 'n' (default = 199)
-Sn   = STACK    : set stacksize to 'n' (default = 100)
-s    = SOURCE   : genrate '#line' directives to sync line to
input
-Tn   = TOKEN    : set tokensize to 'n' (default = 512)
-Un   = UNDEFINE : undefine symbol 'n'
(If no files are specified M4 will use STDIN)
```

Description:
M4 is a hidden gem from the UNIX world and unfortunately often overlooked because the documentation is VERY sparse. I hope the following pages will open the world of M4 to you and making available to you one of the most powerful command line text generators. M4 is a very sophisticated text generator which allows you to generate any kind of structured text: You can create C source, C++ source, HTML scripts, serial letters, plain text files etc.

You can enter M4 instructions from the command line but usually you will create a M4 script file first and then call M4 with the name(s) of all the script files. You could create a library of several set of you standard M4 function and then just pull them in on demand. M4 will use the TEMP environment variable for its temporary files (m4*), if TEMP is not set M4 will place the temporary files into "\temp", "\tmp" or "\" directory - whichever is first available.
To avoid one more problem source M4 is stand alone and does not need the UnixDos DLL (udbase.dll).
Many additional inbuilt macros have been added to augment the standard set of macros:

First I will present a few examples to give yoy a first idea of M4.
Then we will show how you can use M4 to generate CGI scripts (M4CGI).
And finally we will explain each M4 command or macro in detail.

Example1:

Here is s small script defining a macro "mymacro" which just writes a fixed text and inserts a name (see "ex1.m4"):

```
define(mymacro,Now getting name: $1)
mymacro
mymacro(Name)
mymacro(The Name)
mymacro(First,Middle,Last)
```

To run it enter:

```
m4 ex1.m4
```

The result is:

```
Now getting name:
Now getting name: Name
Now getting name: The Name
Now getting name: First
```

You can see how M4 inserted at the "$1" placeholder the first argument supplied when the "mymacro" is called. The extra arguments on the last call

```
mymacro(First,Middle,Last)
```

are ignored.

Example2:
In the following script we introduce the argument counter "$#". It has to be enclosed in the left and right single quotes so that it is not used as a literal text.
I also introduce the use of an internal function "substr(text,start,len)" which extracts a sub-string "text" (the second name) starting at "start" (0 = first column) and extracting up to "len" characters. In effect this script will print the supplied names with the only the initial extracted from the middle name.

```
define(mymacro,Now I got `$#' names: $1 `substr($2,0,1)' $3)
mymacro
mymacro(Name)
mymacro(The Name)
mymacro(First,Middle,Last)
mymacro(John,Fitzgerald,Kennedy)
```

To run it enter:

```
m4 ex2.m4
```

The result is:

```
Now I got 0 names:
Now I got 1 names: Name
Now I got 1 names: The Name
Now I got 3 names: First M Last
Now I got 3 names: John F Kennedy
```

Example3:
Now we will separate the previous script into the following two sections:
- the macro definition section or library (file "ex3_lib.m4")

```
define(mymacro,Now I got `$#' names: $1 `substr($2,0,1)' $3)
```

- the application file (ex3_use.m4)

```
mymacro
mymacro(Name)
mymacro(The Name)
mymacro(First,Middle,Last)
mymacro(John,Fitzgerald,Kennedy)
```

To run it enter:

```
m4 ex3_lib.m4 ex3_use.m4
```

The result is again:

```
Now I got 0 names:
Now I got 1 names: Name
Now I got 1 names: The Name
Now I got 3 names: First M Last
Now I got 3 names: John F Kennedy
```

If you reverse the order M4 will not interpret the "mymacro" calls since they are not yet defined:

```
m4 ex3_use.m4 ex3_lib.m4
```

The result is again:

```
mymacro
mymacro(Name)
mymacro(The Name)
mymacro(First,Middle,Last)
mymacro(John,Fitzgerald,Kennedy)
```

Example4:

Now we will re-use the name macro calls in "ex3_use.m4" and introduce another internal macro:

ifelse(expr,val,true,false).

expr   expression to test

val   value is the value to compare the result against

true   result to return if "expr = val"

false   result to return if "expr != val"

This demonstrates that to can NEST the macros. In this example the "getmiddle" macro is either returning nothing if the middle name does not exist or extracts the first character and appends a dot. This "getmiddle" macro is the called from out original "mymacro" (see "ex4_lib.m4):

```
define(getmiddle,`ifelse(len($1),0,,substr($1,0,1).)')
define(mymacro,Now I got `$#' names: $1 `getmiddle($2)' $3)
```

To run it enter:

```
m4 ex4_lib.m4 ex3_use.m4
```

The result is now:

```
Now I got 0 names:
Now I got 1 names: Name
Now I got 1 names: The Name
```

```
Now I got 3 names: First M. Last
Now I got 3 names: John F. Kennedy
```

Example5 (Serial Letter):

The following file (see "let_lib.m4") contains the framework for your serial letter:

```
define(date,Aug 12th 1990)
define(letter,
$1
$2
$3
$4
                                                          date()
Dear $5`,'

Please find enclosed our new product line presentation.
(your text...)

Sincerely`,'

Marketing Director
(NEWPAGE)
)
```

The following file (see "let_use.m4") makes use of the frame work and creates two example serial letters:

```
letter(XYZ Inc.,Attn: Mr. Green,PO BOX 12345,CITY CA 90010,John)
letter(ABC Inc.,Attn: Mr. Blue,PO BOX 7890,CITY NY 10010,Jack)
```

To generate the letter text enter:

```
m4 let_lib.m4 let_use.m4
```

Here is the output:

```
XYZ Inc.
Attn: Mr. Green
PO BOX 12345
CITY CA 90010
                                              Aug 12th 1990
Dear John,

Please find enclosed our new product line presentation.
(your text...)

Sincerely,

Marketing Director
(NEWPAGE)

ABC Inc.
Attn: Mr. Blue
PO BOX 7890
CITY NY 10010
                                              Aug 12th 1990
Dear Jack,
```

Please find enclosed our new product line presentation.
(your text...)

Sincerely,

Marketing Director
(NEWPAGE)

M4 CGI Mode (M4CGI)

If you are using M4 to generate HTML scripts within the CGI environment you are not executing M4 from the command line, which can pose some problems.
To help you use M4 also in that environment we have implemented the following additional features:

- no DLL needed - M4CGI/M4 is stand alone
- to merge the error output(stderr) with the regular output(stdout)
- execute a default startup script for system wide definitions
- You can change automatically to the same directory where M4*.EXE resides by calling a file in the default startup script containing the the `changedir(sysgethome())`' macros,
  (so that you can use filenames without having to type the absolute pathnames)
- switch trace on BEFORE the system wide definitions are loaded,
  so you can trace even those or not having to add the 'traceon' into any M4 script

To activate the CGI mode use the M4CGI.exe/M4CGIT.exe   instead of M4.exe.
:
M4CGI: Merging error output:
Normally M4 will write error to stderr:
```
m4 bad.m4 > out
```
Will show:

m4: stdin(line 1) cannot open file 'bad.m4' (No such file or directory)

But M4CGI will redirect the error to the file and nothing is written to stderr:
```
m4cgi bad.m4 > out
```

M4CGI: Automatic change of default directory:
Normally M4 will stay in the current directory where you started M4:
```
m4 test.m4
```
This example will open 'test.m4' in the current directory.
But when you run M4 and you don't know in which directory you start from you would have to specify the full path:
```
cd \anydir
m4 \unixdos\test.m4
```

To avoid this M4CGI/M4CGIT will first change to the same directory where the Executable is and THEN run M4 (we assume M4CGI.exe is in c:\unixdos\M4CGI.exe):

```
m4 test.m4              will wun './test.m4'
m4cgi test.m4           will wun 'c:/unixdos/test.m4'
```
So in other words: just place all the M4 scripts into the same directory where M4CGI/M4CGIT lives and just you can call these scripts without any absolute path reference

M4CGI: Automatic startup options:
In many cases you have a standard set of macros you use thoughout your other scripts with either systemwide definitions or basic macros. You can obviously just specify this M4 script(s) with your 'standard functions' on the command line:
```
m4 std.m4 script1.m4
m4 std.m4 script2.m4
```

But this could be a problem when you don't get a chance to specify command line options in the CGI environment. So we implemented the automatic load feature: When M4CGI start it will go to the same directory where M4 started and look for a file named 'm4cgi.arg'. If it is found it will use its contents as arguments AS IF you would have entered then on the command line (we assume M4CGI.exe is in c:\unixdos\M4CGI.exe):
c:\unixdos\m4cgi.arg contains:
```
std.m4
```

c:\unixdos\std.m4 contains:
```
define(macIPADDR,121.122.123.124.21)
```

c:\unixdos\href.m4 contains:
```
define(macHREF,<A HREF="macIPADDR">$1</A>)
HREF(My first hyperlink)
HREF(My 2nd hyperlink)
```

From the command line you would enter:
```
cd \anydir
m4cgi std.m4 href.m4
```

From the command line you enter:
```
m4cgi href.m4
```
And you should see:
```
<A HREF="121.122.123.124.21">My first hyperlink</A>
<A HREF="121.122.123.124.21">My 2nd hyperlink</A>
```

Here is what happened:
1. M4CGI checked if c:\unixdos\m4cgi.arg' exists
2. if yes: read each line and interpret as a separate argument (std.m4)
3. when all lines are read and processed use the arguments you actually entered on the command line and process them (href.m4)
So effectively you expanded M4's default behaviour always loading 'std.m4'(or other

script or scripts) BEFORE any files are processed from the command line. This could come in very handy when you just associate '.m4' extensions with c:\unixdos\m4cgi.exe and then have no control what happens when Windows95 is calling M4CGI.

M4CGI: Automatic trace:
Normally M4/M4CGI will switch the debug trace on only when you place the 'traceon' macro into an M4 script or use the '-t' option. This can become a problem if you are debugging code which is in the startup scripts (see above). So to switch trace on BEFORE the internal startup script just use M4CGIT/M4T:
test.m4 contains:
```
define(mac1,just text $1)
mac1(here)
traceoff
the end
```

The following command:
```
m4cgi test.m4
```
will show:
```
just text here
the end
```

But now we switch trace on BEFORE the startup:
```
m4cgit test.m4
```
will show:
```
Trace(0): define(mac1,just text $1)
Trace(0): mac1(here)
just text here
Trace(0): traceoff
the end
```

M4 COMMAND REFERENCE

Additional UnixDos M4 inbuilt macros
1. changedir(d)   change to directory 'd' (exit on error with an error message)
                           Example: changedir(d:/temp)
                           (Note that you can even change drives!)
2. schangedir(d)   change to directory 'd' (continue also on error - silent
   changedir)
                           Example: schangedir(e:/wrong)
                           (Note that you can even change drives!)
3. sysabort(n)   Set abort condition on errors:
                           0=don't abort, 1=abort *default)
4. sysascii(n)   Write ASCII character   'n' to output, formats of 'n' available:
                           0xXX   = hex value XX

0XXX	= octal value		
'c'	= character 'c'		
XXX	= decimal value		
Example:	sysascii(10)	= write a Line Feed(LF)	
	sysascii(0x41)	= write 'A'	
	sysascii(0102)	= write 'B'	
	sysascii('C')	= write 'C'	

5. *sysgetargs(n)   return arguments from the command line (0=prog name 1=arg1 etc)

    Example: Your 1st arg is sysgetargs(1)

6. sysgetenv(e)   return the environment variable 'e'
    Example: Temporary files are in sysgetenv(TEMP)

7. *sysgethome()   return the directory where M4 is located
    Example: M4.exe is in directory sysgethome()

8. *sysgetname()   return just the program name without extensions
    Example: Current M4 program name is sysgetname()

9. *sysgetpwd()   return the current directory
    Example: changedir(getenv(TEMP)) now in dir sysgetpwd()

10. sysgetcnt(s,d)   return number of fields in string 's' using delimter 'd'
    Example: 3rd dir in PATH = sysgetfld(sysgetenv(PATH), ; ,3)

11. sysgetfld(s,d,f)   return field number 'f' from string 's' using delimiter 'd'
    Example: Dirs in PATH = sysgetcnt(sysgetenv(PATH), ;)

12. syssetlf(n)   Set CR/LF supression state:
    0=supress CR+LF, 1=pass on to output (default)

13. (* = return the result in lower case)

changequote (change the "quote" characters)

`Syntax: changequote([start,end])`

M4 allows you to "escape" or "quote" any text and prevent the normal interpretation.
By default M4 will use the single left quote (`) to start the escaping and the single right quote (') to end the escaping.
The "changequote" command allows you to redefine these defaults with up to 5 character patterns each. If no arguments are given M4 will return to the default setting.

In the following example (see "changequ.m4") we first escape the interpretation of the internal "len()" command by using the default single quotes:

`` `len(abcd)' ``

Then we redefine the quotes to "~1~" to start and "~2~" to end a quote:

`changequote(~1~,~2~)`

Then we call the internal "len()" command again which will now be interpreted returning 4 ("abcd" is 4 characters long):

`` `len(abcd)' ``

Then we use the new quote patterns to escape again the "len()" command:

`~1~len(abcd)~2~`

Here is the output:

len(abcd)
`4'

```
len(abcd)
```

changecom   (change the "comment" character)
```
Syntax: changecom([start[,end]])
```
M4 allows you to include comments in the code and prevent the normal interpretation. By default M4 will use the hash (#) to start the comment and the new line (NL) to end the comment.

The "changecom" command allows you to redefine these defaults with up to 5 character patterns each. If no arguments are given M4 will disable the comment facility. If only one argument is given it will redefine the start pattern but leave the new line end pattern.

In the following example (see "changeco.m4") we have the "len()" command with a comment. Then we change the comment pattern and so when we use the same line again we will see the comment in the output unless we use the new comment pattern:
```
len(abcd) # this is just a comment
changecom(~3~)
len(abcd) # this is just a comment
len(abcd) ~3~ this is just a comment
```
Here is the output:
```
4
4 # this is just a comment
4
```

decr (decrement by 1)
```
Syntax: decr(val)
```
The "decr()" command will decrement the specified value by one (see "decr.m4"):
```
decr(10)
decr(decr(18))
```
Here is the output:
```
9
16
```

define (define a new macro)
```
Syntax: define(name,value)
```
The "define()" command is on of the essential commands and allows you to define new macros in M4. The "name" argument is the new name for the macro and "value" is the replacement text whenever the "name" macro is called. The following items have special meaning:

$0   the name of the macro itself
$1   the first argument
$2   the second argument
..
$9   the ninth argument (the arguments after 9 have to be accessed using shift)
$*   all given arguments separated by commas
$@   all given arguments separated by commas and enclosed in the current quotes
$#   the number of given arguments

In the following example we will define "mymacro" - call it a few times with different arguments , then "undefine()" it and define a new "mymacro" and call it once (see "define.m4"):

```
define(mymacro,macro value (name=``$0'', cnt=``$#'', 1=$1, 2=$2,
all=$*))
mymacro()
mymacro(one)
mymacro(1,2,3,4,5,6,7,8,9,10)
mymacro(a,b,c,d,e,f,g,h,i,j)
undefine(`mymacro')
define(mymacro,something else(all=``$@''))
mymacro(a,b,c,d,e,f,g,h,i,j)
```

Here is the output:

```
macro value (name=mymacro, cnt=1, 1=, 2=, all=)
macro value (name=mymacro, cnt=1, 1=one, 2=, all=one)
macro value (name=mymacro, cnt=10, 1=1, 2=2, all=1,2,3,4,5,6,7,8,9,10)
macro value (name=mymacro, cnt=10, 1=a, 2=b, all=a,b,c,d,e,f,g,h,i,j)
something else(all=`a',`b',`c',`d',`e',`f',`g',`h',`i',`j')
```

defn (generate a quoted definition)

`Syntax: defn(name)`

The "defn()" command allows you to generate a quoted macro name.
You might want to use this command to redefine the M4 macros if you need those words (like "len", or "index") in the generated text (see "defn.m4"):

```
define(m4len,defn(`len'))
define(m4dnl,defn(`dnl'))
define(m4incr,defn(`incr'))
define(m4decr,defn(`decr'))
define(m4eval,defn(`eval'))
define(m4index,defn(`index'))
undefine(`len')
undefine(`dnl')
undefine(`incr')
undefine(`decr')
undefine(`eval')
undefine(`index')
len(abc)
m4len(abc)
```

Here is the output:

```
len(abc)
3
```

divert (change from output stream)

`Syntax: divert([stream-number])`

The "divert()" command allows you to write your output to one of the 10 output stream M4 maintains (numbered 0-9). By default M4 will write to output stream 0. Once the M4 script files is finished M4 will write the result in the following order: stream0, stream1... stream9.
This allows you to enforce a certain sequence of output.
In the following example (see "divert.m4") we write to output 0, 2 and 3 to demonstrate

how M4 will re-arrange the output based on the output stream numbers:

```
this is the regular output stream 0
divert(3)
this goes to stream 3
divert(2)
this goes to stream 2
divert
this is again the regular output
```

Here is the output:

```
this is the regular output stream 0
this is again the regular output
this goes to stream 2
this goes to stream 3
```

divnum (return current output stream number)

```
Syntax: divnum
```

The "divnum()" command returns the current output stream number (0-9).
By default M4 will write to output stream 0. Once the M4 script files is finished M4 will write the result in the following order: stream0, stream1... stream9.
In the following example (see "divnum.m4") we change streams and call "divnum()":

```
divnum
this is the regular output stream 0
divert(3)
divnum
this goes to stream 3
divert(2)
divnum
this goes to stream 2
divert
divnum
this is again the regular output
```

Here is the output:

```
0
this is the regular output stream 0
0
this is again the regular output
2
this goes to stream 2
3
this goes to stream 3
```

dumpdef (print current definitions)

```
Syntax: dumpdef([macro])
```

The "dumpdef()" command will show you all or the specified currently defined macros.
Note that the output from "dumpdef()" is written to the diagnostic output (stderr).
To capture this output use the UnixDos utility STDERR:

```
stderr m4 dumpdef.m4 > diag.txt
```

In the following example (see "dumpdef.m4") we define a macro and call "dumpdef()":

```
Before:
```

```
dumpdef(`testfunc')
define(testfunc,this is a text)
After:
dumpdef(`testfunc')
```
Here is the output:
```
Before:
After:
testfunc:        this is a text
```

errprint (print to the diagnostic output)
```
Syntax: errprint(message)
```
The "errprint()" command allows you to write to the diagnostic output (stderr).
To capture this output use the UnixDos utility STDERR:
```
stderr m4 errprint.m4 > diag.txt
```
In the following example (see "errprint.m4") we write to call "errprint()":
```
this is regular output
errprint(this is an error message)
```
To run this script enter:
```
m4 errprint.m4 > out.txt
```
Here is the output:
```
this is an error message
```
The regular output is in "out.txt"

eval (evaluate the arithmetic expression)t
```
Syntax: eval(expression[,base[,length]])
```
The "eval()" command allows you to calculate an integer arithmetic expression.
"base" defines the output base to be used (by default 10=decimal, 8=octal, 16=hex, 2=binary).
"length" defines the minimum number of digits to be used for the output.
The following operators are available:

+	add	-	subtract
*	multiply	/	divide
%	modulo	()	parenthesis
|	bitwise OR	&	bitwise AND
0xN	hex number	0N	octal number
N	decimal number		

In the following example (see "eval.m4") we call "eval()" to demonstrate several operands:
```
1 : CALCULATION eval(10*20 - 4*5)
2 : NESTING     eval(incr(eval(10*20 - 4*5)) * 2)
3 : MODULO      eval(10 % 4)
4 : BITWISE AND eval(0xFF & 0x05)
5 : BITWISE OR  eval(0x0F | 0x10)
6 : OCTAL       eval(052)
7 : HEX OUT     eval(100,16)
8 : OCTAL OUT   eval(100,8)
```

```
9 : BIT OUT     eval(100,2)
10: OUT LENGTH  eval(100,10,6)
```

Here is the output:
```
1 : CALCULATION 180
2 : NESTING     362
3 : MODULO      2
4 : BITWISE AND 5
5 : BITWISE OR  31
6 : OCTAL       42
7 : HEX OUT     64
8 : OCTAL OUT   144
9 : BIT OUT     1100100
10: OUT LENGTH  000100
```

ifdef (check if macro defined)
Syntax: `ifdef(macro,yes[,no])`
The "ifdef()" command allows you check if a specified macro has bee defined.
If it is defined the "yes" result is returned, otherwise "no" (or nothing if "no" not specified).
The following example (see "ifdef.m4") illustrates the point:
```
ifdef(`mymacro',YES,NO)
define(mymacro,macro value (name=``$0'', cnt=``$#'', 1=$1, 2=$2,
all=$*))
ifdef(`mymacro',YES,NO)
mymacro()
mymacro(one)
mymacro(1,2,3,4,5,6,7,8,9,10)
mymacro(a,b,c,d,e,f,g,h,i,j)
undefine(`mymacro')
ifdef(`mymacro',YES,NO)
```
Here is the output:
```
NO
YES
macro value (name=mymacro, cnt=1, 1=, 2=, all=)
macro value (name=mymacro, cnt=1, 1=one, 2=, all=one)
macro value (name=mymacro, cnt=10, 1=1, 2=2, all=1,2,3,4,5,6,7,8,9,10)
macro value (name=mymacro, cnt=10, 1=a, 2=b, all=a,b,c,d,e,f,g,h,i,j)
NO
```

ifelse (check for a condition)
Syntax1: `ifelse(string1,string2,same[,notsame])`
Syntax2: `ifelse(string1,string2,same,string3,string4,same[,...])`
The "ifelse()" command allows you compare two string to check if they are equal.
If they are equal the next argument is returned. If not then either the following argument is returned - or the next comparison starts (string3 = string4) and so on...
To check numeric values use the "eval()" command.
In the following example (see "ifelse.m4") we define three macros:
morethan10   check if the supplied argument is larger or equal 10

| checkfora | checks if the starting letter of the given argument is an "a" |
| mymacro | will check if the supplied argument is either equal to "match" or "alternative". If it is neither it return "unknown": |

```
define(morethan10,$1 is `ifelse(substr(eval($1-10),0,1),-,smaller than
10,10 or larger)')
morethan10(9)
morethan10(10)
morethan10(11)
define(checkfora,'$1' `ifelse(substr($1,0,1),a,starts with a,no
starting a)')
checkfora(a begin)
checkfora(begin)
checkfora()
define(mymacro,`ifelse($1,match,found first match,$1,alternative,an
alternative,
unknown)')
mymacro(match)
mymacro(alternative)
mymacro(blabla)
```

Here is the output:
```
9 is smaller than 10
10 is 10 or larger
11 is 10 or larger
'a begin' starts with a
'begin' no starting a
'' no starting a
found first match
an alternative
unknown
```

include (include the contents of the specified file)
```
Syntax: include(filename)
```
The "include()" command allows you to include the entire contents of another file.
You can use this to include files depending on other conditions.
If the specified file does not exist "include()" will display an error message.
The first file included contains the macro to check if the argument is a number ("isnum.m4"):
```
define(ISNUM,`ifelse(substr(translit($1,-
0123456789,~~~~~~~~~~~),0,1),~,1,0)')
```
The other files being conditionally included are ("number.m4"):
```
handling numbers
```
and ("alpha.m4"):
```
handling text
```
In the following script (see "include.m4") we first confirm that the ISNUM macro works and then
include one of the files "number.m4" or "alpha.m4" depending on the type of argument given:
```
include(isnum.m4)
ISNUM(18)
ISNUM(abc)
```

```
define(handleany,`ifelse(ISNUM($1),1,include(number.m4),include(alpha.
m4))')
handleany(18)
handleany(abc)
```
Here is the output:
1
0
handling numbers

handling text

incr (increment by 1)
```
Syntax: incr(val)
```
The "incr()" command will increment the specified value by one (see "incr.m4"):
```
incr(10)
incr(incr(18))
```
Here is the output:
11
20

index (search for text)
```
Syntax: index(text,search)
```
The "index()" command allows you to search for a specific text. "text" is the text to be scanned while "search" is the pattern to be found within "text". If "search" is found M4 will return a zero-based index of the position (otherwise -1).
In the following example (see "index.m4") we demonstrate a few cases:
```
index(this is a text,is)
index(this is a text,` is')
index(this is a text,xxx)
```
Here is the output:
2
4
-1

len (return the length)
```
Syntax: len(text)
```
The "len()" command will return the number of character in "text".
In the following example (see "len.m4") we demonstrate a few cases:
```
len()
len(abc)
```

Here is the output:
0
3

maketemp (generate a unique temporary filename)
```
Syntax: maketemp(name)
```
The "maketemp()" command allows you to create unique filenames. The name has to contain a trailing pattern "XXXXXX" which will be replaced by the unique mark.

In the following example (see "maketemp.m4") we add "aa" in front and the extension ".tmp":

```
write to: maketemp(aaXXXXXX).tmp
```

Here is a possible output:

```
write to: aa023330.tmp
```

m4exit (immediate exit)

```
Syntax: m4exit([exit code])
```

The "m4exit()" command allows you immediately exit - for example on a fatal condition.
In the following example (see "m4exit.m4") we exit in the middle of the script":

```
this
is
m4exit(3)
regular
output
```

Here is the output:

```
this
is
```

m4wrap (specified macro will be executed on exit)

```
Syntax: m4wrap(macro)
```

The "m4wrap()" command allows you define a macro which will always be called before M4 terminates. This gives you an easy way to implement a cleanup macro:
In the following example (see "m4wrap.m4") we can see the cleanup kicking in automatically:

```
define(cleanup,Now it is cleaning time)
m4wrap(`cleanup')
Regular output here
```

Here is the output:

```
Regular output here
Now it is cleaning time
```

popdef (remove the current "define" and make the previous one current)

```
Syntax: popdef(macro)
```

The "popdef()" command allows you make a previous definition of the specified macro again current.
In the following example (see "popdef.m4") we push two definitions for "mymacro" on the stack and the call "popdef()" to re-activate the first one:

```
pushdef(mymacro,This is the first)
mymacro
pushdef(`mymacro',This is my second)
mymacro
popdef(`mymacro')
mymacro
```

(Note the quotes in the second "pushdef()" to escape the execution of "mymacro()")

Here is the output:
This is the first
This is my second
This is the first

pushdef (push a new definition a new macro)
```
Syntax: pushdef(macro,value)
```
The "pushdef()" command allows you define a macro like "define()", but the previous macro definition is kept on a stack.
In the following example (see "pushdef.m4") we push two definitions for "mymacro" on the stack and the call "popdef()" to re-activate the first one:
```
pushdef(mymacro,This is the first)
mymacro
pushdef(`mymacro',This is my second)
mymacro
popdef(`mymacro')
mymacro
```
(Note the quotes in the second "pushdef()" to escape the execution of "mymacro()")

Here is the output:
This is the first
This is my second
This is the first

shift (shift arguments and quote them with comma separation)
```
Syntax: shift(arguments)
```
The "shift()" command allows you to extract all arguments shifted by one.
In the following example (see "shift.m4") we demonstrate "shift()":
```
shift(`a',`b',`c',`d')
```

Here is the output:
b,c,d

sinclude (include the contents of the specified file)
```
Syntax: sinclude(filename)
```
The "sinclude()" command is identical to "include()" except that is does NOT display an error message if the file is not found.

substr (extract a substring)
```
Syntax: substr(text,start[,len])
```
The "substr()" command allows you to extract a sub string. The ""start" argument defines the beginning column (zero-based) while "len" defines the number of characters to extract from that point onwards. If "len" is not specified "substr()" will extract until the end if the text.
In the following example (see "substr.m4") we demonstrate "substr()":
```
substr(This is a text,5)
substr(This is a text,5,4)
```

Here is the output:

is a text
is a

syscmd (execute a system command)

```
Syntax: syscmd(command)
```

The "syscmd()" command allows you execute programs of batch files while in M4.
In the following example (see "syscmd.m4") we first define a "runit" macro which just executes the specified argument. Then we just run it on UnixDos utility "udate".
Then we define the "getfiles()" macro which finds out how many files you have in the current directory and writes the result into "getfiles.tmp".
Then we just display a text with the result:

```
define(cleanup,syscmd(rm -s x1.tmp x2.tmp x3.tmp getfiles.tmp))
m4wrap(`cleanup')
define(runit,`syscmd($1)')
runit(udate)
define(getfiles,
runit(ls > x1.tmp)
runit(wc -lt x1.tmp > x2.tmp)
runit(cut -f1 x2.tmp > x3.tmp)
runit(`uecho -s @x3.tmp '\c'' > getfiles.tmp)
runit(rm x1.tmp x2.tmp))
getfiles()
This directory contains include(`getfiles'.tmp) files
```

Here is a possible output:

Tue Oct 29 18:16:17 1996
This directory contains 53 files

traceon/traceoff (switch tracing on/off)

```
Syntax: traceon|traceoff
```

The "traceon/traceoff()" commands allow you to monitor the internal steps M4 is taking in processing your script. It is an excellent debugging tools to see how macros are expanded

In the following example (see "trace.m4") we first switch the trace facility on "traceon" and then define a "cleanup" macro. Then we define "aamacro1()" which calls in turn the lower level macros "aamacro2/3". Then we call macro "aamacro1()".

Then we do the same process but first switch the trace of "traceoff".

Before we exit we switch the trace back on to demonstrate the automatic call to "cleanup()":

```
traceon
define(cleanup,doing the cleaning)
m4wrap(`cleanup')
define(aamacro3,bottom reached arg=$1)
define(aamacro2,$1)
define(aamacro1,aamacro2(aamacro3($1)))
aamacro1(Trace is on)
traceoff
define(bbmacro3,bottom reached arg=$1)
define(bbmacro2,$1)
define(bbmacro1,bbmacro2(bbmacro3($1)))
bbmacro1(Trace is off)
traceon
```

Here is the output:

```
Trace(0): define(cleanup,doing the cleaning)
Trace(0): m4wrap(cleanup)
Trace(0): define(aamacro3,bottom reached arg=$1)
Trace(0): define(aamacro2,$1)
Trace(2): aamacro3($1)
Trace(1): aamacro2(bottom reached arg=$1)
Trace(0): define(aamacro1,bottom reached arg=$1)
Trace(0): aamacro1(Trace is on)
bottom reached arg=Trace is on
Trace(0): traceoff
bottom reached arg=Trace is off
Trace(0): cleanup
doing the cleaning
```

Note that the M4 trace facility is always showing you the level (0=base, 1=nesting level 1...) as it expands your definition.

translit (translate characters)

```
Syntax: translit(string,from-list,to-list)
```

The "translit()" commands allows you to translate or convert a specific set of characters into another set. You can for example convert lower case letters to upper case.

The "string" will be scanned and any character matching the "from-list" will be translated into the corresponding character in the "to-list" at the same position. Any characters not

found in the "from-list" will be passed unchanged.

In the following example (see "translit.m4") we first define a few macros:

CAT to concatenate up to three arguments into one text
UP to convert any text to upper case
LOW to convert any text to lower case
FIRST to extract the first letter in a text
REST to extract the text after the first letter
NAME to generate a name with the first letter in upper case, the rest in lower case
ISNUM to check if the input is a number(returns1) or a text (returns 0)
ISNUMTXT to give feedback about the type of input in text form

Then we run a few examples:
```
#-------------- concatenate function
define(CAT,$1$2$3)
#-------------- convert to upper case function
define(UP,`translit($1,abcdefghijklmnopqrstuvwxyz,ABCDEFGHIJKLMNOPQRSTUVWXYZ)
')
#-------------- convert to lower case function
define(LOW,`translit($1,ABCDEFGHIJKLMNOPQRSTUVWXYZ,abcdefghijklmnopqrstuvwxyz
)')
#-------------- extract first letter function
define(FIRST,`substr($1,0,1)')
#-------------- extract rest after the first letter function
define(REST,`substr($1,1)')
#-------------- Generate name with first letter capital rest lower case
define(NAME,`CAT(UP(FIRST($1)),LOW(REST($1)))')
#-------------- check if input is a number function
define(ISNUM,`ifelse(substr(translit($1,-
0123456789,~~~~~~~~~~~),0,1),~,1,0)')
define(ISNUMTXT,'$1' is a `ifelse(ISNUM($1),1,number,text)')
#-------------- applications
ISNUMTXT(18)
ISNUMTXT(text)
UP(YourNameHere)
LOW(YourNameHere)
FIRST(YourNameHere)
REST(YourNameHere)
NAME(YourNameHere)
```

Here is the output:
```
'18' is a number
'text' is a text
YOURNAMEHERE
yournamehere
Y
ourNameHere
Yournamehere
```

undefine (remove macro)
```
Syntax: undefine(macro)
```
The "undefine()" commands allows you to remove a macro definition permanently.

This might be useful before you want to change the definition of an existing macro.
In the following example (see "undef.m4") we first define "mymacro()" and call it a few
times. Then we remove the first definition and create second one:

```
define(mymacro,first definition args=$*)
mymacro()
mymacro(one)
mymacro(a,b,c,d,e,f,g,h,i,j)
undefine(`mymacro')
define(mymacro,second definition cnt=`$#')
mymacro()
mymacro(one)
mymacro(a,b,c,d,e,f,g,h,i,j)
```

Here is the output:

```
first definition args=
first definition args=one
first definition args=a,b,c,d,e,f,g,h,i,j
second definition cnt=1
second definition cnt=1
second definition cnt=10
```

Possible M4 Problems:

If M4 find an error it will show you the file name where the error occured and the
corresponding line number: m4: {filename}:{line number}   {error message}

```
m4: loop.m4:2 pushed back more than 4096 chars
```

Infinite recursion:

If you use the macro name within its definition M4 will go in an infinite loop.
In the following example (see "loop.m4") we define the macro "cleanup()" but use the
word cleanup within its definition:

```
define(cleanup,Now it is cleanup time)
cleanup()
```

Here is the result:

```
Now it is Now it is Now it is Now it is Now it is Now it is Now it is Now it
is
.....
Now it is Now it is Now it is Now it is Now it is Now it is Now it is Now it
is
Now it is Now it is Now it is Now it is Now it is Now it is Now it is
m4: loop.m4:2 pushed back more than 4096 chars
cleanup()
```

Missing escape:

If you call a macro you might have to divert the expansion using the quote mechanism.
In the following example (see "quote.m4") we will define macro "mymacro()" and then
undefine it and then try to define a new definition of it:

```
define(mymacro,First definition)
mymacro
undefine(mymacro)
```

```
define(mymacro,Second definition)
mymacro
```

Here is the result:
First definition

m4: quote.m4:5 bad macro name
define(First definition,Second definition)

The "undefine()" was not actually getting the macroname "mymacro" but its result ("First definition"). To actually undefine the "mymacro()" use the quotes:
```
undefine(`mymacro')
```

Here is the result:
First definition
Second definition

Using an existing macro name by mistake:
In some cases you might need to use words which are already defined as internal or our own M4 macros.
In the following example (see "sysname.m4") we will define macro "mymacro()" with system command names in it:
```
define(mymacro,There is a shift in the DowJones index)
mymacro
```
Here is the (maybe surprising) result:
There has been a in the DowJones -1

To actually use those macro names place the double quotes around the names:
```
define(mymacro,There has been a ``shift'' in the DowJones
``index'')
```
Here is the result:
There has been a shift in the DowJones index

Using a comma within the definition:
If you use a comma within your definition of the value M4 will stop processing and ignore the remaining text. To avoid this just enclose the comma in quotes.
In the following example (see "comma.m4") we will define macro "date()":
```
define(date,The date is Aug 12,1996)
mymacro
```
Here is the (maybe surprising) result:
The date is Aug 12

To actually use commas within the definition place the quotes around the comma:
```
define(mymacro,The date is Aug 12`,'1996)
```
Here is the result:
The date is Aug 12,1996

mf (Show free memory)

<u>Syntax:</u>
`mf`

<u>Description:</u>
MF will display the free memory available in bytes.

<u>*Example:*</u>
`mf`

Might show:
542576 bytes free

mkdir (Make a directory)

Syntax:
```
mkdir [-c] [-l] {directory name(s)}
-c      : CHANGE: change to the last new directory
-l      : LIST: show newly created directories
```

Description:
MKDIR   will create one or more directories specified on the command line.

It is very similar to the DOS equivalent MD except that it can create several directories and that MKDIR can accept backward and forward slashes (e.g. \USR\SRC or /USR/SRC)

If the directory name starts with '/' or '\' MKDIR will create a new directory in the root directory - independent of your current position. Otherwise MKDIR will create new subdirectories from your current position.

Example1:
Create a new directory and change directly to it:
```
mkdir /mynewdir
```

Will show:
```
c:\MYNEWDIR
```

Example2:
Create a several new directories and list them:
```
mkdir -l /tmp1 /tmp2 /tmp3
```

Will show:
```
Created directory: /tmp1
Created directory: /tmp2
Created directory: /tmp3
```

(u)more (Browse thru text file)

Syntax:
```
umore [-c{n}] [-t{n}] [-w] [file{s} ...]n
-c{n}     = column: set column offset to 'n' (default is 0)
-t{n}     = tab  : set tab increment to 'n' (default is 8)
-w        = wrap : don't wrap lines
```

Description:
UMORE will display the contents of a text file one screenful at a time.

If a line is longer than the screen width UMORE will wrap onto the next line until the full line is displayed. You can switch the wrapping off (-w option) in which case UMORE will only display the first 80 characters of a line. You can specify a different line offset with the -c{n} option in which case UMORE will display the next 80 characters from that offset onwards.

Any TAB character is expanded to the next multiple of 8 columns unless
you define a different TAB stop with the -t{n} option.

If no file has been specified UMORE will use STDIN.

Once a screen is displayed UMORE will display the following prompt:
```
-- more -- (per = help) (17%)
```
(where "per" is the percentage of the last line on the screen in relation to the entire file)

Once the prompt is displayed UMORE will accept the following commands:
SPACE BAR Display next screen
-              Goto previous file (if several files specified)
+   Goto   next file (if several files specified)
=   Display the current filename, line and total bytes
/{pat}   Search for the next pattern or regular expression
               (Example: `/^[a-m]` search for line starting with 'a' thru 'm')
b   Backup to previous page displayed
g{n}[c|l]   Goto line 'n' in current file (trailing 'c' specifies byte offset)
               After you hit the 'g' key UMORE will display the prompt:
               `Enter location:'`
               Example:
               `g100`   goes to line number 100
               `g1010c`   goes to byte offset 1010
               If you specify a negative number UMORE will count from the end of the
file:
               `g-50`   goes to line which is 50 lines before the end
               `g-200c`   goes to 200 bytes before the end of file

```
h or ?        Display this help
l             Display all the current files list in process and point to the current filename
n             Repeat the pattern search and find the "next" hit
q or CTRL-C Quit
v             Call the editor specified with the 'EDITOR' environment for the current file.
              (If the 'EDITOR' environment is not set UMORE will call VI)
!{cmd}        Run the command'{cmd}' (DOS gateway)
              (Example: !udate   shows the current date/time)
```

Examples:
Display all C source file (one at a time):
```
umore *.c
```

Display only column 81 thru 160 without wrapping:
```
umore -w -c80
```

Notes:
There are many variations of the MORE command since it is not part of the UNIX V standard.

mv (Move or rename files)

Syntax:
```
mv (Move files: copy and remove)
usage: mv [-{l|f|v}] {src} {dest}
-f     = force : always remove old files (even if read-only)
-l     = list  : show which file is currently copied
-v     = verify: verify that input is identical to output
       (this only applies when the file had to be copied between
drives)
src    = source file(s)
dest   = destination
         (has to be directory if several source files specified)
```

Description:
MV will move one or several files to the specified destination.

The modification time and file modes are preserved.

MV will first check if the source file exists and then remove the destination file. If that fails and the '-f' option has not been specified MV will abort the move of that file and continue with the next one (if any).
MV will then use the rename function of DOS. If that fails (e.g. move between different drives) MV will copy the file and then remove the source file.

Example:
```
mv file1 file2
```
Will move file1 to file2. If file2 exists already it will be overwritten.
If file2 is read-only protected MV will show an error message.

```
mv *.c \tmp
```
Will move all C source files (extension '.c') into the directory '\tmp' and preserve all modification times and file modes. If '\tmp' is not a directory and there are several C source files MV will print an error message and not copy.

```
mv file1 file1
```
MV will print an error message because you tried to move a file onto itself.

Notes/Remarks:
The UNIX MV does not offer the '-l' list option and will not warn you when you move several files into one file instead of a directory.

msgbox (Display message box)

```
msgbox -f{mf} [-e{ef}]
-f{mf} = Define the message file containing the details
-e{ef} = Define the file were to write the exit code
```

Description:
MSGBOX allows you to bring up a message box from the command line.
The details about the requested operation are defined in the the message file.

Exit Code:
The exit reflects the users response.
It is written as a decimal value to the specified file using the '-e' option.
If no file is specified '%TEM%/msgbox.tmp' is used.
The following exit values are available:.

1   OK
2   CANCEL or ESC key
3   ABORT
4   RETRY
5   IGNORE
6   YES
7   NO

Message File:
The MSGBOX message file contains several sections specifying the desired behaviour:
It has the following three sections:
• Title (defining the box title)
• Flags (defining the behaviour)
• Text (defining the actual message text)

Title:
TITLE={text of title}
This instruction is defining the title for this message box.
It can be up to 200 characters long.
If no title is specified "Message Delivery" is used.

Flags:
CMD={Flag1|Flag2|...}
This instruction defines the flags altering the available options and attributes.
You can specify several flags separating with the pipe character:
```
OKCANCEL|ICONERROR
```
Here are all the available codes:
```
OK, OKCANCEL, ABORTRETRYIGNORE, YESNOCANCEL, YESNO, RETRYCANCEL
ICONHAND, ICONQUESTION, ONEXCLAMATION, ICONASTERISK,
```

```
ICONINFORMATION, ICONSTOP, ICONWARNING, ICONERROR
DEFBUTTON1, DEFBUTTON2, DEFBUTTON3, DEFBUTTON4,
HELP, RIGHT
```

Message:
All the other lines will become part of the actual message

Example:
Below is an example of a message file 'msg1.txt':
```
TITLE=Confirmation
CMD=YESNO|ICONSTOP
Do you want to continue?
```

To activate a message box with these parameters use:
```
msgbox -fmsg1.txt -eexit.txt
```

Once the use has responded you will find in 'exit.txt' the exit code (1=Yes, 2=no)

mvdir (Rename a directory)

```
usage: mvdir [-l] {old} {new}
-l      = LIST: show which directory is currently moved
old     = old directory (can be relative or with leading \)
new     = new directory (can be relative or with leading \)
```

Description:
MVDIR will allow you to "rename" a directory. MVDIR will create the new directory and then descend thru the old directory and "move" all files and sub-directories.

Example:
```
mvdir -l c:\tmp c:\temp
```

nl (Line numbering)

Syntax:
```
nl [-f] [-i{n}] [-n{f}] [-p] [-s{c}] [-v{n}] [-w{n}] {file(s)}
List file with line numbers (separates files with a form-feed
lines)
-f     = filename  : Precedes the line with the filename
-i {n} = increment : Set increment to 'n' (default is 1)
-n {f} = number fmt: Set line number format:
                     rn    = right justified (default)     : '
1016'
                     ln    = left  justified               :
'1016    '
                     rz    = right justified with leading 0:
'0001016'
-p     = no reset  : Do not reset the counter between files
-s {c} = separator : Set separator after line number (default is
TAB)
-v {n} = start     : Set first line number to 'n' (default is 1)
-w {n} = width     : Set width of line number to 'n' (default is
6)
```

Description:
NL will read lines from each specified file, attach a line number to the beginning and send the output to STDOUT. NL will reset the counters for each new file. NL will insert a blank line between each new file.

Example:
Print all C source files with '.   ' after the line number and include the filename:
```
nl "-s. " -f *.c
```

Print file 'fn' with a 10 digit counter starting at 1000 and leading zeros, increment is 10:
```
nl -w10 -v1000 -nrz -i10 fn
```

Notes:
The following differences exist:
UNIX NL can process only one file
UNIX NL -s option accepts only a single character
UNIX NL does not provide the -f option (print filename)
UnixDos NL does not provide the special formatting options:
```
-b/-h/-f/-l/-d
```

od (Octal, hex, decimal, ascii file dump)

```
od [-cCbhmodxfsODXFS] [-v] [file] [+{o}[.[b]]]
-b   octal byte display (3 digits per single byte)
-m   decimal byte display (3 digits per single byte)
-h   hex byte display ('x' and 2 digits per byte)
-c   ASCII character display
     (all non-printable chars as '\ooo' octal code except: null=\0,
     backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t,
0377=\FF)
-C   Extended ASCII character display
     (all non-printable chars as IBM graphics chars, rest as above)

2 Byte Values (short):
-o   2 byte octal value (always 6 digits with leading zeros)
-d   2 byte unsigned decimal (up to 5 digits)
-s   2 byte signed decimal (up to 5 digits with leading '-' if
negative)
-x   2 byte hex value ('0x' and 4 hex digits)

4 Byte Values (long):
-O   4 byte octal value (always 10 digits with leading zeros)
-D   4 byte unsigned decimal (up to 9 digits)
-S   4 byte signed decimal (up to 9 digits with leading '-' if
negative)
-X   4 byte hex value ('0x' and 8 hex digits)

Floating point Values:
-f   4 byte floating point value (6 digit precision:
[-]n.nnnnnE[+|-]ee)
-F   8 byte floating point value (19 digit precision:
[-]n.nnnnnnnnnnnnnnnnnnE[+|-]eee)

Other options:
-v   Do not skip identical lines (default will skip)

file
specify the input file (if omitted use STDIN)

+off[.[b|B]      offset:
          Specify the offset within the 'file' and set the offset
display type
          (default is 7 digit octal display).
          off. decimal offset and display (800. = offset at byte 800)
          0xoff hex offset display (0x1FF = offset at byte 511)
          b    multiply 'off' by 512 bytes (small block)
          B    multiply 'off' by 1024 bytes (large block)
```

Description:

OD will display the contents of the specified 'file'. The first argument specifies the display format(s). If no format is given OD will use '-o' (2 byte octal display). If no file is given OD will use STDIN. OD will skip identical lines unless you specify the '-v' option. OD will start the display from the beginning of the file and display the offset value in 7 digit octal value at the beginning of the line unless you specify an offset. You can change the offset display type to decimal ('.' after the offset) or hex display ('0x' an front of the offset). To just change the offset display type but still start from the beginning of the file enter '+0.' for decimal or '+0x0' for hex

If you specify more than one display format in the first argument OD will display the file contents always in the following sequence:

Single Bytes first: b, m, x, c, C
2 Bytes second: o, d, s, x
4 bytes third: O, D, S, X
Floating point last: f, F
(So it doesn't matter in which sequence you specify the formats!)

Example:

Display all formats of input file 'od.exe':

```
od -bmxcCodsxODSXfF od.exe
```

Will print:

```
00000 115 132 200 000 147 000 016 000 040 000 274 001 377 377 250 015
```
(byte octal)

```
       77  90 128   0 103   0  14   0  32   0 188   1 255 255 168  13
```
(byte decimal)

```
      x4D x5A x80 x00 x67 x00 x0E x00 x20 x00 xBC x01 xFF xFF xA8 x0D
```
(byte hex)

```
        M   Z 200  \0   g  \0 016  \0      \0 274 001 377 377 250  \r
```
(byte ASCII)

```
        M   Z      \0   g  \0      \0      \0     \FF \FF   (  \r
```
(byte Ext ASCII)

```
      055115  000200  000147  000016  000040  000674  177777  006650
```
(2byte octal)

```
       23117     128     103      14      32     444   65535    3496
```
(2byte uint)

```
       23117     128     103      14      32     444      -1    3496
```
(2byte int)

```
      0x5A4D  0x0080  0x0067  0x000E  0x0020  0x01BC  0xFFFF  0x0DA8
```
(2byte hex)

```
             8411725          917607        29098016       229179391
```
(4 byte uint)

```
             8411725          917607        29098016       229179391
```
(4 byte int)

```
           0x805A4D         0xE0067       0x1BC0020      0xDA8FFFF
```
(4 byte hex)

```
       1.17873E-038     1.28584E-039     6.90605E-038     1.04154E-030
```
(4 byte float)

```
          1.947158195916215000E-308        7.322733909935053000E-243
```
(8 byte float)

```
00020 270 013 261 241 252 105 000 000 036 000 000 000 001 000 205 105
      184  11 177 161 170  69   0   0  30   0   0   0   1   0 133  69
```

```
xB8 x0B xB1 xA1 xAA x45 x00 x00 x1E x00 x00 x00 x01 x00 x85 x45
270 013 261 241 252   E  \0  \0 036  \0  \0  \0 001  \0 205   E
  8       1   !   *   E  \0  \0      \0  \0  \0      \0       E
005670  120661  042652  000000  000036  000000  000001  042605
  3000   41393   17834       0      30       0       1   17797
  3000  -24143   17834       0      30       0       1   17797
0x0BB8  0xA1B1  0x45AA  0x0000  0x001E  0x0000  0x0001  0x4585
    2712734648          17834              30      1166344193
   -1582232648          17834              30      1166344193
    0xA1B10BB8          0x45AA            0x1E      0x45850001
  -1.19971E-018    2.49908E-041    4.20390E-044    4.25600E+003
      3.784501320507053000E-310      8.123987410768453000E+026
```

paste (Merge lines from several files)

Syntax:

List files parallel (max 18 files):
paste [-d{list}] file(s)
-d{list} = delimiter list: define delimiter list (default TAB)
          Once the list is used up paste will start again from
beginning
          The following special characters are available:
          null=\0, backspace=\b, formfeed=\f, newline=\n,
          return=\r, TAB=\t, \xNN=hex code NN

List files fixed serial:
paste -d{list}] - [- ...] < file

List lines serial:
usage: paste [-s] [-d{list}] [file(s)]
-s   = serial: serial merging - all line on one new line

Description:
PASTE   will combine lines of several files.

PASTE will use TAB character to separate each line unless you specify a new
separating character with the '-d' option. If you specify several characters PASTE will
cycle thru them and start again from the beginning once the last one is reached.

The following modes are available:
Parallel:
Read all input files (up to 15) at the same time and paste each line of each file next to
each other:
line1/file1,line1/file2, line1/file3...
line2/file1,line2/file2, line3/file3...
...

Fixed serial:
Read from STDIN and substitute each '-' on the command line with the next line read.
Once all specified '-' are substituted PASTE will start a new line and start again
substituting '-'.

Serial:
Read the specified file(s) and display all lines separated
with the separation character. Once all lines are displayed PASTE
will start a new line and read the next specified file until all files
are read.

File 'paste1.txt' contains:
line1-file1
line2-file2
line3-file1

File 'paste2.txt' contains:
line1-file1
line2-file2

Merge files parallel:
```
paste paste1.txt paste2.txt
```

Output:
line1-file1\tline1-file2
line2-file1\tline2-file2
line3-file1

Merge serial with delimiter list (' ' and '\n') to combine line pairs:
```
paste -s -d\x20\n paste1.txt paste2.txt
```

Output:
line1-file1   line2-file1
line3-file1
line1-file2   line2-file2

List files in two columns separated by ',':
```
ls | paste -d, - -
```

Notes:
The UNIX PASTE is limited to max 12 parallel files and 511 bytes per output line while the UnixDos PASTE can handle 18 parallel files and 10,000 bytes per output line.

pr (Print files)

```
pr [options] {files}
```
+{n}   begin page: Start printing at page 'n'
-{n}   multi column: Print in 'n' column output:
                  Example:
```
The     of     text
lines a        file
```
-a   across: Switch to horizontal multi column style (with -{n} option):
```
The     lines of
a       text  file
```
-d   double: double-space the output
-e[c]{n}   expand: Set tab char to 'c' and spacing to 'n' (default TAB and 8)
-f   form-feed: use form-feeds for new page instead of line-feeds
-h   {h}   header: Use 'h' instead of filename for the page header
-i[c]{n}   implode: Substitute blanks with 'c' and spacing set to 'n'
                  (default   is TAB and spacing 8)
-l{n}   page length: Set the page length (default is 66 lines)
-m   merge: Print all files parallel (overwrites '-{n}' option):
```
lines lines lines
of    of    of
file1 file2 file3
```
-n[c]{n}   number: Precedes each line with the line number of width 'n'
                         and insert 'c' after the number (default is width 5 and TAB)
-o{n}   offset: Indent all lines with 'n' blank positions (default none)
-p   pause: Pause after each printed page
-r   report: Don't print error message if file does not exist or is empty
-s{c}   separator: Separate columns by one single char 'c' instead of blanks
-t   text: Just print the file text (no header, footer, trailer or form-feed)
-w{n}   width: Set the line width to 'n' (default is 72 columns)
[file(s)]   File(s) to be used as input (if none are specified STDIN is used)

Description:
PR will print one or more files according to the 'options' on the command line. By default the file is separated into pages. Each page contains   a header containing the date, filename and page name. If no files are given PR will use STDIN. If the input file contains the form-feed character '\f' PR will start a new page.

PR writes the output to STDOUT. Use the redirection to send the output to the printer.

Examples:
Print all C source files on the printer:
```
pr *.c >> prn
```

Print   filelist one screen at a time:

```
ls -l | pr -l24 -h "DIRECTORY LIST" | umore
```

Print a condensed file list (6 rows) for the output 6x14 columns wide without header and sorted horizontally:
```
ls | pr -6 -w `expr 6 x 14` -at | umore
```

`ls`   Generate short list of files in current directory
`` `expr 6 x 14` `` run calculation 6 x 14 and take the result (78) as the width
                 argument for PR (6 columns and 14 characters wide,
                 filename is 8 characters long + '.' + file extension 3 letters)
`pr -6 w 78 -at` Print 6 columns '-6' with width 78 '-w 78' sorted
                 horizontally '-a' without header '-t'.
`umore`   Display result one screen full at a time.

Print alphabetical list of files (not directories) with file attribute/size/time
and name horizontally sorted but no page header:
```
ls -l | grep -h ^-|cut -c1-4,11-42|pr -2 -at|umore
```

`ls -l`   generate the long filelist in alphabetical order
`grep -h ^-`   select only files: lines starting with '-' (directories start with 'drwx...')
`cut -c1-4,11-42` Extract only columns 1 thru 4 (file attributes) and
                   11 thru 42 (file size, date, time and name)
`pr -2 -at`   Print in 2 column format '-2', display horizontally '-a' and no page
header   '-t'
`umore`   display the result one screen at a time

<u>Notes:</u>
The UNIX PR expects all arguments to be followed directly by the parameter (e.g.   '-w{width}') while the UnixDos PR allows also space(s) (e.g. '-w78' and '-w   78' is valid)

pwd (Print working directory)

Syntax:
```
pwd [-d]
-d   Display the directory in DOS style:
     capital letters and '\' between files and directories
     The default is UNIX style:
     small letters and '/' between files and directories
```

Description:
PWD will display the drive and directory name you are currently in.

PWD will display the directory name in UNIX style unless you specify
the '-d' for the DOS style display.

Example:
```
pwd
```

Might show:
```
c:/src/unix
```

```
pwd -d
```

Might show:
```
C:\SRC\UNIX
```

ps (Show current proceses)

```
ps (Show current processes)
Windows 95/NT:
usage: ps

Windows3.X/MSDOS:
usage: ps [-f]
-f      = full detailed memory map
default = only command lines with space occupied
SIZE    = Total size occupied from program in bytes
MCD     = Memory  Control Block Address (Hex)
PSP     = Program Segment Prefix Address (Hex)
OWN     = PSP of Owner/Caller (hex)
PAR     = Number of paragraphs(16bytes) occupied by segment
(hex)
BYTES   = Bytes occupied by segment (decimal)
COMMAND = original command line with arguments
```

Description:
PS will display the current programs in memory.
Under Windows 95/NT PS will separate the processes by threads and sort them by
Thred/PID.

Example:
```
ps
```

Might show (Window 95/NT):

```
HWND   THREAD        PID = PROGRAM
=============================================================
0E0 FFF03D2D FFF03919 = Spooler Process

0AC FFF07109 FFF07D15 = Program Manager
4F8 FFF07109 FFF07D15 = OleMainThreadWndName
4FC FFF07109 FFF07D15 = DDE Server Window
500 FFF07109 FFF07D15 = OleRpcNotify0xfff07109

110 FFF0B239 FFF0BE51 = MS-DOS Prompt

1B4 FFF1412D FFF0AF49 = LPT1

140 FFF14B0D FFF0AE21 = Mail
170 FFF14B0D FFF0AE21 = MS Mail Notification
1C0 FFF14B0D FFF0AE21 = Outbox
```

```
1D4 FFF14B0D FFF0AE21 = Inbox

198 FFF14E21 FFF149D5 = HP DeskJet Status Monitor
1A0 FFF14E21 FFF149D5 = HP DeskJet 660Cse
1A8 FFF14E21 FFF149D5 = &Cancel Print

1B8 FFF15BFD FFF14775 = MS Mail Spooler (auto)
1F8 FFF15BFD FFF14775 = MS Mail Notification

598 FFF1E421 FFF1EE55 = Microsoft Word - UD_REF.RTF
5A0 FFF1E421 FFF1EE55 = DDE Server Window
5A8 FFF1E421 FFF1EE55 = OleRpcNotify0xfff1e421
5FC FFF1E421 FFF1EE55 = TODO.DOC
914 FFF1E421 FFF1EE55 = UD_REF.RTF
CE8 FFF1E421 FFF1EE55 = UD_GEN.RTF

128 FFF32D1D FFF31785 = PS
```

rm (Remove files/directories)

Syntax:
```
rm [-filr] {file/directory}
-f      = FORCE    : remove also read/only files
-i      = INQUIRE  : confirm each file removal
-l      = LIST     : show removed files
-r      = RECURSIVE: remove also files in all subdirectories
-s      = SILENT   : don't display not found error messages
```

Description:
RM will remove files or entire directories. The file contents will be lost!

If a file is protected (read-only) RM will confirm if you really want to remove the file. RM will display the file with attributes, size, date and name and wait for your response: 'y' will remove, enter or any other key will NOT remove.

Example:
Remove all files not starting with 'a-m' and prompt for confirmation:
```
rm -i [^a-m]*
```

Might show:
```
-rw-rw-rw-    11267  Nov 04 1988 old.txt: ? y
-rw-rw-rw-    10563  Dec 03 1989 old.zip: ?
-rw-rw-rw-     9004  Nov 16 1989 old2.zip: ? y
```

All lines where you enter 'y' RM will remove the file.
If you just hit enter or any other key RM will not remove that file.

Remove the entire '/src' directory including all subdirectories, remove
also read-only files and list each file as it is removed:
```
rm -rlf /src
```

Notes/Remarks:
The UNIX RM does not provide the '-l' (list) option and will not show the size or date/time in the confirmation prompt.

saveexit (Save errorlevel of previous command)

Syntax:
```
saveexit
```

Description:
SAVEEXIT will save the errorlevel/exit status of the previous program into the file:
```
c:\udexit.txt
```
It will set the environment variable UDEXIT to "c:\udexit.txt" which is used by the UnixDos TEST utility to retrieve the last exit value asynchronous. SAVEEXIT is a batch file, because there is no reliable way to access the exit code of the previous program from within a program.

Example:
```
cf file1 file2
call saveexit
```
... more commands...
```
rem ****** NOW TEST THE LAST EXIT CODE ****
test '!' -e 0 run: uecho file1 and file2 are different
```

sed (Stream editor)

```
sed [-n] [-e {expr}] [-f {script}] [file(s)]
-e {expr}   = EXPRESSION: replace according to expression 'expr'
                          (see below)
-f {script} = FILE       : process expressions from script file
'script'
-n          = NOT ALL   : output only the lines which are edited
```

Description:
SED is a great workhorse in filtering and editing any text file according to your editing instructions. SED has no limits in terms of file size. SED uses it's own language and you can even build your own SED scripts and apply them to text files using the -f option.
SED will read the input from each file (or STDIN if none are specified) and apply the editing expressions {expr}. The output is written to STDOUT. If only one expression is used the '-e' option is not required ("sed s/A/a/ file" works, but "sed -e s/A/a/ -e s/B/b/ file" needs the -e prefix).
SED will print all lines (edited and untouched) unless you specify the "-n" option in which case SED will only print the edited/accessed lines. (i.e. sed -n   3,4p file).

SED script language:
The general format of SED to apply specific functions to the text is as follows:
```
{address}[!]{function}[arguments]
```

Defining the address {address} (required)
This is a required specification which defines which lines should be used for the following function. There are two formats available:
1. Line range using the format: `line1[,line2]`
      line1   = starting line as decimal number ($ = last line in file)
      line2   = ending line as decimal number ($ = last line in file)
            (if not specified same as line1,line2)
      Examples:
      3,10   = line 3 to 10
      5,$ = line 5 to the end of file
      $,$ = only the last line in the file
      9   = only line 9
2. Regular expression using the format of regular expressions
      Examples:
      /abc/   = all lines containing pattern 'abc'
      /p[a-z]/   = all lines containing 'p' followed immediatelly with any lowercase letter
      /tmp.*32/   = all lines containing 'tmp' followed by '32' after 0 or more other letters
      /ZIP$/   = all lines ending with pattern 'ZIP'

/^tmp/   = all lines starting with pattern 'tmp'

Negating the address [!]   (optional)
You can invert the address range of line by placing the exclamation mark in from of the function.
Examples:
sed -n "3,10!p" file   = print all lines except lines 3 to 10
sed -n "/ZIP$/!p" file = print all lines except those ending with 'ZIP'
sed -n "/^tmp/!p" file = print all lines except those starting with 'tmp'

SED Functions {function} (required)
Below is a complete list of all the SED functions or commands in alphabetical order.
Some command can only be used within a SED script file. In some cases the address can only be all liens or only one single line.
You can insert a special character or escape the special meaning of a character using '\':
\xNN in text will be translated to the hex ASCII code NN except:
\b=backspace, \f=formfeed, \n=line feed, \r=return, \t=TAB

Append:	append 'text' after each line
a\	(Can only be used from a SED script file)
{text}	(Can only be applied to one single line or all lines)
Example1:	will create a new line after each original line with the 'NewLine'
a\	
NewLine	
Example2:	will create a new line after line 8 with the text 'Line8'
8a\	
Line8	

Branch:	goto to the specified 'label' (or end of script if no label specified)
b[label]	(Can only be used from a SED script file)

Change:	change/replace the specified lines with {text}
c\	(Can only be used from a SED script file)
{text}	
Example1:	will replace line 9 with "this is a replaced line"'
9c\	
this is a replaced line	
Example2:	will replace lines 4 to 7with one single line "line 4 to 7 are gone"'
4,7c\	
line 4 to 7 are gone	

Delete:	delete lines at current address (regular expression or line range)
d	
Example1:	will delete lines 5 to 10 from the output
sed 5,10d file	

Example2:   will delete all lines starting with 'tmp'
sed /^tmp/d file
Example3:   will delete all lines NOT ending with 'zip'
sed "/zip$/!d" file

Insert: insert 'text' before each line text
i\   (Can only be used from a SED script file)
{text}   (Can only be applied to one single line or all lines)
Example1:   will create a new line before each original line with the 'BEFORE'
a\
BEFORE
Example2:   will create a new line before line 8 with the text 'after line7'
8a\
after line7

List: display text and convert non printable characters to \xHH (HH=hex code)
l
Example1:   will show lines 3 to 6 with any non printable characters as hex code
sed -n 3,6l file

Print: print specified lines
p
Example1:   will print lines 18 to 1000
sed -n 18,1000p file
Example2:   will print lines 632 to the end of file
sed -n 632,$p file
Example3:   will print all lines except 1000 to 2000 of file
sed -n "1000,2000!p" file

Quit: exit script
q   (Can only be used from a SED script file)

Read file: insert another file
r {f}   (Can only be used from a SED script file)
              (Can only be applied to one single line or all lines)
Example1:   will insert file 'sep_line' after each
r sep_line
Example2:   will insert file 'x1' after line 3
3r x1

Substitute: substitute: replace any occurrence of regular expression {regexp} with
'new'
s/{regexp}/new/[flags]
              flags: g=global (replace all occurances in each line, default is only the
first)
                    p=print if replacement has been made

{n}=replace only the {n}th occurance of {regexp} (default is only first)

w {f} = append to file 'f' if replacements have been made

The first character after the 's' becomes the delimiter ('/' used here)

Example1:   will replace each occurance of 'tmp' with 'temporary'
sed s/tmp/temporary/g file

Example2:   will replace only the first occurance in each line of 'tmp' with 'temporary'
sed s/tmp/temporary/ file

Example3:   will replace all occurances of 'tmp' with 'temporary' only between lines 100 to 200
sed 100,200s/tmp/temporary/g   file

Example4:   will replace all occurances of 'tmp' with 'new' only on lines which start with 'extra'
sed /^extra/s/tmp/new/g   file

Example5:   will replace all occurances of 'tmp' with 'new';   also write the changed lines to 'edit.txt'
sed s/tmp/new/gw edit.txt   file

Example6:   will replace only the 2nd occurances of 'tmp' with 'new' on lines 30 to 90
sed 30,90s/tmp/new/2   file

Test: goto 'label' (or end of file if omitted) if replacement took place
t [label]   (Can only be used from a SED script file)

Write: write output also to file {f}
w {f}   (Can only be used from a SED script file)
Example1:   will write all lines containing 'get' to file 'get.txt'
/get/w get.txt

Transform: convert any character in list 'lst1' to the corresponding character in list 'lst2'.
y:lst1:lst2:   Both lists must have equal length! (Similar to the TR UnixDos utility)
Example1:   will convert all lower case letters to upper case
sed y:abcdefghijklmnopqrstuvwxyz:ABCDEFGHIJKLMNOPQRSTUVWXYZ: file

Label: Define a label (Only for 'b' and 't' command)
:label   (Can only be used from a SED script file)

Line Numbers: print current line number on a new line (counted across files!)
=   (Can only be applied to one single line or all lines)
Example1:   will append one line containing the line number before each line
sed =   file
Example2:   will append one line containing the line number only before line 10
sed 10=   file

Comments: comments within a script only on very first line
comment   (Can only be used from a SED script file)

Grouping of commands { }
SED allows you to group SEVERAL functions with their addresses together and apply them only to certain addresses. All commands withing the curly brackets apply to the previously defined address space.
Example1:
# apply the following functions only to lines with 'here'
/here/{
s/tmp/temporary/
y:abcdefghijklmnopqrstuvwxyz:ABCDEFGHIJKLMNOPQRSTUVWXYZ:
a\
===
}
In this example only lines containing 'here' are edited by all the functions withing the "{}" pair:
1. replace 'tmp' with 'temporary'
2. convert from lower to upper case
3. append =========================== after each line

Examples:
Select line 10 thru 20 and 100 thru 105 of file:
```
sed -n -e 10,20p -e 100,105p file
```

Substitute commas with LineFeeds:
```
sed "s:,:\n:g" file
```

Delete lines 2 thru 8 and 12 thru 14 from file:
```
sed -e 2,8d -e 12,14d file
```

Insert 'NEW: ' in front of each line in file (\: is used to not terminate the replacement text):
```
sed -e "s:^:NEW\: :" file
```

Append   ',END' after each line in file:
```
sed -e "s:$:,END:" file
```

Substitute '1' by 'ONE' and '2' by 'TWO' in file:
```
sed -e "s:1:ONE:g" -e "s:2:TWO:g" file
```

Append ',END' only in line 2 thru 10 in file:
```
sed -e "2,10s:$:,END:" file
```

Append '(new text)' after each line but remove lines 2 thru 8 and 10 thru 14 first:
using script file 'app.sed':
2,8d
10,14d
a

```
(new text)
```

```
sed -f app.sed file
```

Append the file 'more.txt' after line 10 thru 20 from file: using script file 'add.sed':

```
1,9d
21,$d
r more.txt
```

```
sed -f add.sed file
```

Notes:
The UnixDos   SED can process lines up to 10,000 bytes long while the UNIX SED can handle only up to 1024 bytes and does not provide the hex character substitution.
The UNIX "sed" cannot substitute text for LineFeeds (\n) etc.
UnixDos allows you to substitute also with control codes (\n or \hex code).

setenv (Set environment variable(s))

Syntax:
```
usage: setenv {VAR}={value} [VAR=value ...]
VAR    = Variable name
value  = New value for variable (cannot contain '=')
Output should be redirected into a temporary batch file:
setenv VAR=value ... > tmpenv.bat
call tmpenv.bat
```

Description:
SETENV change or set environment variables.
The functionality is identical to the DOS "SET" internal command except for:
- SETENV allows you to set several variable at once
- SETENV allows make the powerful UnixDos argument expansion available to set values

Unfortunately each argument for SETENV has to be enclosed in double quotes, because funny DOS will break up arguments containing '=' into separate arguments:
```
setenv "LIB=c:\msvc\lib"
```

Since there is no way to set or change the environment for the parent process I had to create SETENV.BAT as a batch file which internally calls "SETENV_.EXE". This utility creates a temporary batch file to set the variable(s) and is removed after its duty is done.

Example1:
Set FILECNT to the number of files in the current directory:
```
call setenv "FILECNT=`ls | wc -lt | cut -f1`"
```

Example3:
Multiply variable VAR by two:
```
setenv VAR=`expr $VAR x 2`
```

Example3:
Below is an elaborate example of a full script prompt the user for files to copy using SETENV:
```
@echo off
rem ****** CHECK IF COPY WANTED *************************
usrchar Do you want to copy a file? (Y/N)\c yn
IF ERRORLEVEL 2 GOTO END

rem ****** GET THE INPUT FILENAME ***********************
usrprmpt Enter the source filename:\c -r
call setenv "IN=@usrprmpt.tmp"

rem ****** CHECK IF THIS FILE EXISTS ********************
```

```
test -r "$IN" runneg: echo FILE %IN% DOES NOT EXIST
IF ERRORLEVEL 1 GOTO END

rem ****** GET THE OUTPUT FILENAME **********************
echo Enter the destination filename:
call setenv "OUT=`line`"

rem ****** CHECK IF IT IS WRITABLE IF IT EXISTS **********
test -r $OUT
IF ERRORLEVEL 1 GOTO DO_COPY
test -w $OUT runneg: echo CANNOT WRITE TO FILE %OUT%
IF ERRORLEVEL 1 GOTO END

rem ****** EXECUTE THE ACTUAL COPY *********************
:DO_COPY
cp -l %IN% %OUT%
goto END

:END
```

Notes:

The UNIX SETENV is and internal command in the C-Shell and not a standalone program and does not allow you to set several variables.

show_exp (Show UnixDos argument expansion)

Syntax:
```
usage: show_exp {arguments to expand}
```

Description:

SHOW_EXP will expand the arguments you specify and display their expanded results:
The very powerful UnixDos argument expansion can get a bit confusing, so I created a small test utility (SHOW_EXP), which just expands the arguments and shows you the final arguments which would be passed into your program.

Example:
```
set UDSHOW_EXP=au*.bat;*.sys
show_exp $LIB `udate`

Adding default environment: $UDSHOW_EXP=au*.bat;*.sys
Arguments before expansion:
--------------------------------------------------------
IN:  argv[ 0] = c:\put\SHOW_EXP.EXE
IN:  argv[ 1] = $LIB
IN:  argv[ 2] = `udate


Arguments after expansion:
--------------------------------------------------------
OUT: argv[ 0] = c:/put/show_exp.exe
OUT: argv[ 1] = autoexec.bat
OUT: argv[ 2] = config.sys
OUT: argv[ 3] = io.sys
OUT: argv[ 4] = msdos.sys
OUT: argv[ 5] = pagefile.sys
OUT: argv[ 6] = c:\lib;c:\msvc\lib
OUT: argv[ 7] = Thu Aug 29 16:00:06 1996
```

Notice how the following expansions take place:
First the automatic environment for SHOW_EXP is pulled in and expanded,
I used the separator ';' to specify two regular expressions "au*.bat" and "*.sys":
```
UDSHOW_EXP  ->    OUT: argv[ 1] = autoexec.bat
                  OUT: argv[ 2] = config.sys
                  OUT: argv[ 3] = io.sys
                  OUT: argv[ 4] = msdos.sys
                  OUT: argv[ 5] = pagefile.sys
```

Then the general environment variable "LIB" is expanded:
```
$LIB        ->    OUT: argv[ 6] = c:\lib;c:\msvc\lib
```

And finally the UDATE utility is called and the result becomes the last argument:
```
`udate`     ->    OUT: argv[7] = Thu Aug 29 16:00:06 1996
```

slash (Convert file slashes)

Syntax:
```
slash [-d] [file(s)]
-d       = display in DOS style (default UNIX style)
file(s)  = the filename(s) to be converted
If no 'file(s)' are specified slash will read STDIN and convert
each line
```

Description:
SLASH will convert the file separator '/' in UNIX or '\' in DOS.

You can either specify one or more filenames on the command line or provide a list of filenames in STDIN if you dont specify any filename.

This conversion might be needed for programs which need the DOS style for filenames (e.g. PKZIP).

Example:
Display all files in DOS style:
```
ufind / -exec slash -d {} ^
```

Build filelist for PKZIP:
```
ufind / -print | slash -d >> x.lst
pkzip all @x.lst
```

sleep (Wait for specific time)

Syntax:
```
sleep {number of seconds to sleep}
```

Description:
SLEEP will wait for the specified number of seconds and then return back.

Example:
Wait for 10 minutes and then print a banner:
```
sleep  `expr 10 x 60`
banner done
```

(u)sort (Sort/merge text files)

Syntax:
```
usort   [-bcdfimMnru]   [-t{c}]   [-T{d}]   [F]   [-o{f}]   [I]

-b    blank: ignore leading blanks
-c    check: check the sort sequence (Create output only if unsorted)
-d    dictionary: use dictionary order (only A-Z,a-z,0-9 are relevant)
-f    fold: fold lower case letters to upper case letters
-I    ignore: ignore non-printable characters (outside ASCII range
0x20-0x7f)
-m    merge: only merge the specified input files (sorted already)
-M    month: compare as months (JAN,...DEC) non-month default to JAN
-n    numeric: use numeric sort sequence (convert text to binary value)
-r    reverse: sort in descending order (default ascending)
-u    unique: make the output unique (suppers all duplicate sort keys)
-t{c} field separator: use character 'c' as separator (default is
blank)
      You can specify also hex codes: -t\7C defines the '|' character
-T{d} temporary directory: define directory 'd' for temporary files
      (Example: '-Td:' will use the ram disk 'D:' for work files)
F     field(s): {+|-}f[.c[o]]
      +     start of sort key
      -     end of sort key
      f     field number (0=first, 1=2nd...)
      c     column offset within field (0=first, 1=2nd...)
      o     options for field: b=blank, d=dictionary, f=fold,i=ignore
      M=month, n=numeric, r=reverse(descending)
-o{f} output: set the output field (if not specified STDOUT is used)
I     input file(s): specify one or more input files (if non STDIN is
used)
-y{n} work ram: set size to 'n'*1024 bytes (-y0=use min, -y=use max)
-z{n} record length: set to 'n' (default is 1024)
      (only used with '-c' or '-m' since USORT does not scan lines)
```

Description:
USORT   will read all lines from all input files, sort them according to the specifications and write the output to STDOUT (unless the '-o' option specifies an output file). If no input files are specified USORT will read from STDIN.

USORT will sort based on the specified sort key(s): pairs of '+f' and '-f' options. If no keys are specified USORT will use the entire line for comparisons. The second sort key is only used if the first sort keys are identical (and so on).

Sort keys:
Sort keys are defined with pairs of '+f' and '-f' options. If no '-f' is specified USORT will assume the end of line. Each field is separated by the separator character (default is white space: blank or TAB). You can redefine the separator with the '-t{c}' option:

```
field1  field2    field3 filed4    field5 (default)
field1,field2 ,field3,field4,  field5 (-t,)
```

To use field3 as the sort key (field numbers are zero based 0=first):
```
usort +2 -3
```

You can skip a certain number of columns within the field.
To use column 3 thru 5 of field3:
```
usort +2.2 -3.5
```

You can associate various options to the sort key.
To sort on field3 using column 3 thru 5 and convert to upper case and dictionary order:
```
usort +2.2fd -3.5
```

Examples:
Sort   file 'in' based on field2 and 3 (separated by ','), write to 'out' and use ram disk D:
```
usort +1 -3 -t, -TD: in -o out
```

Sort entire line based on column 10 thru 20 in descending order:
```
usort +0.9 -1.20 -r
```

Sort current directory by file extension and filename:
```
dir | usort +0.9 -0.12 +0.0 -0.9 | umore
```

Sort current directory by file extension and file size (descending order):
```
dir | usort +0.9 -0.12 +2rn -3 | umore
```

Note:
USORT   can sort also huge files (we successfully sorted 25MB files!) in the 32bit version. In the 16 Bit version the limit is around 2-3MB!

split (Split text file into chunks)

Syntax:
```
split [-l] [-{n}] [file  [name]]
-l    : list   : show new files as they are created
-{n}  : number : set the number of lines per piece (default is
1000)
file  : input  : input filename (default is stdin)
name  : output : output name prefix (default is 'x')
(Maximum line length is 10000)
```

Description:
SPLIT will read the input file 'in' (STDIN if none or '-' given) and writes this file in chunks
of 1000 lines (unless changed with '-n' option) into the a set of output files until the
entire input file
has been read. The first output file has the name 'xaa', the second 'xab' and so on until
'xzz'. You can change the output prefix from 'x' to any letter sequence up to 6 character.

Example:
File '`long.txt`' contains 11093 lines.
You want to split it in 2000 line chunks with prefix 'long':
```
split -l -2000 long.txt long
```

Might show:
```
split: Starting 'longaa' ...
split: Starting 'longab' ...
split: Starting 'longac' ...
split: Starting 'longad' ...
split: Starting 'longae' ...
split: Starting 'longaf' ...
split: Read 11093 lines, split into  6 files
```

Check lines per file with:
```
wc long*
```

Might show:
```
  11093    110930   long.txt
   2000     18000   longaa
   2000     18000   longab
   2000     18000   longac
   2000     18000   longad
   2000     18000   longae
   1093      9837   longaf
  22186    210767   TOTAL
```

Note that the last chunk 'longal' contains no exactly 2000 lines but just the remainder of
1093 lines.

<u>Note:</u>
UnixDos SPLIT line length limit is 10,000 bytes which exceeds the UNIX limit.

stderr (Redirect stderr to stdout)

Syntax:
```
stderr [cmd [argument(s)]]
cmd    = command/program to be executed under the new
environment
arg    = optional arguments for the command
To avoid expansion of arguments enclose in single quotes:
stderr uecho '`cat x`'
```

Description:
STDERR can be used to redirect output from programs which is written to STDERR.
STDERR will merge the STDERR with STDERR so that then you can redirect the
output into a file.

Example:
Capture the usage of USORT:
```
stderr usort >> x
```

strings (Extract text strings from binary files)

Syntax:
```
strings [-d] [-o] [-x] {file(s)}
-d     = decimal : print offset in decimal
-e     = extended: include foreign characters
-o     = decimal : print offset in octal
-x     = decimal : print offset in hexadecimal
-{n}   = length  : set minimum text length (default is 4)
```

Description:
STRINGS will scan the input file(s) for ASCII strings. A string a sequence with at least 4 printable ASCII character. You can include also foreign character with the '-e' option. To increase the requirement of 4 characters use the '-{n}' option.

Example:
Find strings in file '\command.com' with minimum length of 10 characters and display the file offset in decimal:
```
strings -12 -d \command.com
```

Might show:
```
2172  %1 drive %2
2211  %1 device %2
2230 &Please insert volume %1 serial %2-%3
2277 %File allocation table bad, drive %1
2346 Invalid COMMAND.COM
2373 !Insert disk with %1 in drive %2
2435 !Press any key to continue . . .
2502 Terminate batch job (Y/N)?
2535 Cannot execute %1
2561 Error in EXE file
2604 "Program too big to fit in memory
2652 No free file handles
2677 Bad Command or file name
2732 Access denied
2753 Memory allocation error&
3005 Cannot load COMMAND, system halted
3054 Cannot start COMMAND, exiting
3116 Top level process aborted, cannot continue
3222 Write protect errorInvalid unit
3275 Invalid device request
3324 Data error!Invalid device request parameters
3414 Invalid media type
3437 Sector not found
3460 Printer out of paper error
3513 Write fault error
3535 Read fault error
3556 General failure
3576 Sharing violation
3620 Lock violation
```

```
3637 Invalid disk change
3663 FCB unavailable
```

sum (Display checksum and block count of file(s))

Syntax:
```
sum [-r] {file(s)}
-r = use alternate algorithm
```

Description:
SUM will read each specified file and print a 16 bit checksum with the file size in 1024 byte blocks to STDOUT.

SUM is an excellent tool to confirm the correct file transmission
- any lost or changed characters will change the checksum.

Example:
Print checksums of system files and '.com' files in the root directory:
```
sum  /*.sys /*.com
```

Might show:
```
3939 47 /command.com
6224 9 /ansi.sys
54960 1 /config.sys
9010 11 /himem.sys
12951 33 /io.sys
58891 37 /msdos.sys
7519 9 /smartdrv.sys
```

tail (Print last part of a text file)

Syntax:
```
tail [{+|-}[n][blc]] [-f] [file]
+n      = begin : count from beginning of file
-n      = end   : count from end of file
b       = block : count in blocks (1024 bytes)
c       = char  : count in characters
l       = lines : count in lines (default)
-f      = follow: wait and print lines with are appended
Example: tail -200 text (print the last 200 lines of file
'text')
(Maximum backup 20,000 bytes)
```

Description:
TAIL will display the input file starting at the specified location.
If no input file is given TAIL will use STDIN.

If the '-f' option is specified TAIL will display the text and then wait for new text appended to the input and display. This option applies only in an multi user environment to "watch" files for new input.

Example:
To display the last 20 lines of a file:
```
tail -20 file
```

To view the 15 most recent files in your current directory:
```
ls -lt | tail -15
```

To view the file starting at line 1000:
```
tail +1000 file
```

Notes:
UnixDos TAIL will allow to "back up" of to 20,000 bytes while the UNIX TAIL uses only a 4096 byte buffer.

tee (Display STDIN and copy to file(s))

<u>Syntax:</u>
```
tee [-i] [-a] [file(s)...] [-w] [file(s)...]
-a      = append: append output to the next specified file(s)
-w      = write : write(don't append) output to the next
specified file(s)
-i      = ignore: ignore CTRL-C
file(s) = output file(s) to write to
You can output in parallel to up to 20 files
```

<u>Description:</u>
TEE will read STDIN and write it to the screen AND the listed output file(s).
By default TEE will remove any old contents in output file(s) and write the input (-w option).
With the append option (-a) you can append to an existing file.

<u>Example:</u>
Display information from all text files (*.txt) and append to "big.log" and write to "small.log":
```
ls *.txt | tee -a big.log -w small.log
```

Produce the checksum of file 'large.dat', display and store in file 'large.chk':
```
sum large.dat | tee large.chk
```

test (Test expression)

```
test [-l] {expr} [run[neg]: prog1 [arg..] ["^" prog2
[arg...]]...]
(returns errorlevel 0 if all condition are true otherwise
returns 1)
SINGLE FILE CONDITIONS:          | STRING CONDITIONS:
-d   {fn} = file is a directory  | -n {s}      = string length
> 0
-f   {fn} = file is a regular file | -z {s}      = string length
is 0
-r   {fn} = file is readable      | {s1} = {s2} = strings are
equal
-s[n] {fn}= size is larger than 'n'| {s1} != {s2}= strings
unequal
         (if fn=D: then free bytes)   | {s1} ~= {p} = pattern 'p'
appears in s1
-t   {h}  = handle'h' is a terminal|
-w   {fn} = file is writable      | INTEGER CONDITIONS:
-x   {fn} = file is program       | {n1} -gt {n2} = greater
than
SPECIAL:                           | {n1} -ge {n2} = greater
equal
-e[=nlg?] {l} = check errorlevel   | {n1} -eq {n2} = equal
 = equal(default) n not equal      | {n1} -ne {n2} = not equal
 l less than       g = greater than | {n1} -le {n2} = less equal
 ? display level and test equal    | {n1} -lt {n2} = less than
COMBINING CONDITIONS:
-a  = and         as in "-r file1 -a -f file2"
-o  = or          as in "-r file1 -o -f file2"
-!  = negate      as in "'!' -r file1" (Note single quotes for
non-expansion)
()  = grouping    as in "'(-r file1 -o -f file2) -a -d directory"
NOTE: Always enclose the '!' with single quotes to avoid
expansion

DOUBLE FILE CONDITIONS:
(f1} -Tn {f2} = file 'f1' newer than file 'f2'
(f1} -TN {f2} = file 'f1' newer or equal than file 'f2'
(f1} -To {f2} = file 'f1' older than file 'f2'
(f1} -TO {f2} = file 'f1' older or equal than file 'f2'
(f1} -Te {f2} = file 'f1' has same time as file 'f2'

UNIX SPECIFIC (included just for compatibility - always untrue):
-b {fn} = block special device
```

```
-c {fn} = character special device
-g {fn} = file has group-id set
-h {fn} = file has symbolic link
-k {fn} = file has sticky bit
-p {fn} = file is pipe
-u {fn} = file has user-id set

if {expr} is TRUE test will run all the specified program(s)
-l   = list: show the exit level

Special conditions:
-e[=|n|l|g|?]   {l}

errorlevel: checking the errorlevel of previous program
-e= {l}   true if previous errorlevel is equal to 'l' (default)
-en {l}   true if previous errorlevel is not equal to 'l'
-el {l}   true if previous errorlevel is less than 'l'
-eg {l}   true if previous errorlevel is greater than 'l'
-e? {l}   true if previous errorlevel is equal to 'l'
     (display the errorlevel before the comparison)
(run 'SAVEEXIT' before calling test under DOS 4 or earlier!)

Run program(s):
run: prog1 [arg..] ["^" prog2 [arg..] ["^" prog3 [arg...]]
(runs the specified programs if '{expr}' is true)
runneg: prog1 [arg..] ["^" prog2 [arg..] ["^" prog3 [arg...]]
(runs the specified programs if '{expr}' is NOT true)
```

Description:
TEST will evaluate the expression formed out of one or several conditions and exit with errorlevel 0 if the expression or TRUE - otherwise errorlevel 1.
So you can easily integrate the test in your batch files:

```
test {condition}
if errorlevel 1 goto ERROR
rem *** condition is true
...
goto END

:ERROR
condition is not true

:END
```

TEST will also run the specified programs if you specify the 'run:'
switch on the command line.

Conditions can be combined with AND '-a' or OR '-o'. Conditions can be inverted with the '!' operand in front of the condition (remember to enclose the ! operand in single quotes to avoid argument expansion).

Examples:
Check if file 'log' is writable and append file 'newlog' to 'log' if true:
```
test -w log run: cat newlog >> log
```

Check   if directory '/usr/tmp' exists and run MKDIR to create it:
```
test -d /usr/tmp run: mkdir /usr "^" mkdir /usr/tmp
```

Check if there is 800,000 bytes left on drive D: and copy file 'data1.zip' if true:
```
test -s800000 d: run: cp -l data1.zip d:
```

Check if filesize of logfile 'my.log' is below 50,000 bytes and 'l' is writable, append 'newlog' to 'l' if these conditions are true:
```
test -w my.log  -a '!' -s50000 my.log  run: "cat   newlog >>
my.log"
```


Show the last saved exit value:
```
cp blabla x1
```
cp: Cannot find 'blabla'

```
call saveexit
uecho do something else... here
```
do something else...

```
test -e? 0
```
LAST EXIT VALUE WAS: 1

Check if the environment variable OUT is set - if not set OUT to \default.out:
```
test -z $OUT
IF NOT ERRORLEVEL 1 SET OUT=\default.txt
```

Check if environment DEBUG is set to ON and set DEBFLAG to '-Zi' if true:
```
test "%DEBUG%" '!=' "ON"
IF NOT ERRORLEVEL 1 SET CCFLAG=-Zi
```

Check if directory 'UnixDos' is not found in the PATH environment variable and print banner:
```
test '!' $PATH ~= UnixDos run: banner NO UnixDos PATH
```

Check if there are at least 500,000 bytes free in conventional memory, print a banner if not:
```
test "`chkdsk | grep -h free | cut -f1`" -lt 500000 run: banner
low ram
```

Check if the current directory contains the same number of C source files as make files and print a message if not:

```
test "`ls *.c | wc -l`" -ne "`ls *.mk | wc -l`"
IF NOT ERRORLEVEL 1 banner "mk/c" mismatch
```

Check if file 'x.c' is newer the 'x.obj' and print message if true:

```
test x.c -Tn x.obj run: uecho x.obj is older
```

Check   if two files 'dat1' and 'dat2' are identical and print banner if not and copy dat1 to dat2:

```
cmp dat1 dat2
test -e= 0 run: banner copy ^ cp -lv dat1 dat2
```

Check if   the either file 'm1.err' or 'm2.err' contain more than 0 bytes and the file '\showerr' exists and display the errors:

```
test ( -s m1.err -o -s m2.err ) -a -r \showerr run:
cat m1.err m2.err
```

User Prompt Example:
Prompt the user for the files to copy with checks (test_cp.bat):

```
@echo off
rem ****** CHECK IF COPY WANTED *************************
usrchar Do you want to copy a file? (Y/N)\c yn
IF ERRORLEVEL 2 GOTO END

rem ****** GET THE INPUT FILENAME ***********************
usrprmpt Enter the source filename:\c -r
call setenv "IN=@usrprmpt.tmp"

rem ****** CHECK IF THIS FILE EXISTS ********************
test -r "$IN" runneg: echo FILE %IN% DOES NOT EXIST
IF ERRORLEVEL 1 GOTO END

rem ****** GET THE OUTPUT FILENAME **********************
echo Enter the destination filename:
call setenv "OUT=`line`"

rem ****** CHECK IF IT IS WRITABLE IF IT EXISTS *********
test -r $OUT
IF ERRORLEVEL 1 GOTO DO_COPY
test -w $OUT runneg: echo CANNOT WRITE TO FILE %OUT%
IF ERRORLEVEL 1 GOTO END

rem ****** EXECUTE THE ACTUAL COPY *********************
:DO_COPY
cp -l %IN% %OUT%
```

```
goto END

:END
```

Output example1:

Do you want to copy a file? (Y/N) Y
Enter the source filename: blabla
FILE blabla DOES NOT EXIST

Output example2:

Do you want to copy a file? (Y/N) Y
Enter the source filename: x1.bat
Enter the destination filename:
xnew
CANNOT WRITE TO FILE xnew

Output example3:

Do you want to copy a file? (Y/N) Y
Enter the source filename: x1.bat
Enter the destination filename:
newfile
x1.bat   -> newfile
1 files written successfully

Loop Example:

Print   a table of the powers of two (test_exp.bat):

```
@echo off
set EXP=1
set BASE=2
:LOOP
    echo 2 power %EXP% = %BASE%
    call setenv "EXP=`expr $EXP + 1`"
    call setenv "BASE=`expr $BASE x 2`"
    test $EXP -le 16
IF NOT ERRORLEVEL 1 GOTO LOOP
```

Will print:

```
2 power 1 = 2
2 power 2 = 4
2 power 3 = 8
2 power 4 = 16
2 power 5 = 32
2 power 6 = 64
2 power 7 = 128
2 power 8 = 256
2 power 9 = 512
2 power 10 = 1024
2 power 11 = 2048
2 power 12 = 4096
2 power 13 = 8192
2 power 14 = 16384
2 power 15 = 32768
2 power 16 = 65536
```

Notes:
The UNIX TEST misses the following options found in the UnixDos TEST:
```
-s{n}           (you can only check if the filesize is larger
than 0)
{s1} ~= {s2}    (string search for pattern)
-n  {s}          string non-zero length check)
-T[n|N|o|O|e]   (file modification time comparison)
-e[=|n|l|g|?] {l} (Check errorlevel/exit status of previous
program)
```

(u)time (time a command)

<u>Syntax:</u>
```
utime program [arg...]
```

<u>Description:</u>
UTIME will run the program with the specified arguments and display the elapsed time in seconds.

<u>*Example:*</u>
Call 'sleep' with 10 seconds delay:
```
utime sleep 10
```

Might show:
```
    10.76 seconds
```

<u>Notes:</u>
The UNIX TIME will also display other time periods which relate to the multi user environment.
To time a batch file use:
uptime -I
(call your.bat)
uptime

touch (Update date/time of file)

```
touch [-cl] [MMDDhhmm[YY]] {file(s)}
-c           = create: don't create file if it does not yet
exist
                     (this is the default)
-l           = list  : show file(s) being update
  (UNIX a, m flag does not apply, because DOS does not maintain
them!)
MMDDhhmm[YY] = time stamp:
             MM   Month (01-12)
             DD   Day (01-31)
             hh   Hour (00-23)
             mm   Minute (00-59)
             YY   Year (00-99 = 1900 - 1999)
                  (defaults is the current year)
```

Description:
TOUCH will set the modification time of all specified files to the current time or the time specified.

If the file does not exist TOUCH will create it with the current time stamp (unless the '-c' option is specified).

Examples:
Set all text files to Jan 15, 17:00, 1990 - list each file as it is changed:
```
touch -l 0115170090 *.txt
```

Set all C source files to the current time:
```
touch *.c
```

Find all files updated or created after Jan 15, 17:00, 1991:
```
touch 0115170091 \ref
ufind / -newer ref -ls
```

Notes:
Since DOS does not maintain separate access/creation times the '-a' and '-m' option is not provided in the UnixDos TOUCH.

tr (Translate characters)

Syntax:
```
tr [-cds] {set1} [set2] [file(s)|-]
-c      = complement : complement/inverse set1
-d      = delete     : delete all char found in set1
-s      = squeeze    : squeeze repeated char in 'set2'
set1    = source     : source character range
set2    = destination: destination character set (except -d option)
file(s) = input      : input file(s) ('-' or none use STDIN)
Translate mode: (no -d option)
Convert any character found in set1 to corresponding char in set2

Delete mode: (-d option)
Delete any character found in set1 from input

set: list characters, [{c1}-{c2}] will expand to range c1-c2
'[a-z]\012\015'  = small letter + CR + LF
'[A-Z][a-z]_[0-9]' = letters, underbar and numbers
'[{c}*n]; in set2 will expand 'c' 'n' times (to 'n' fill to end)
Example: tr '[a-z]' '[A-Z]' file1 (lower case to upper case)
Example: tr '/' '\' file1 (translates UNIX slashes to DOS slashes)
Example: tr -cs '[A-Z][a-z][0-9]' '[\012*]' file (list of words)
Example: tr "'[ -~][A-Z][a-z][0-9]'" '[x*96][L*26][l*26][n*26]'
        translate to: L=capital letter, l=small letter, n=numbers,
x=rest
```

Description:
TR will read the input files, translate characters according to the specifications and write the output to STDOUT. If no input file is given tr will use STDIN.

Examples:
Translate lower case letter to upper case of file 'in':
```
tr '[a-z]' '[A-Z]' in
```

Translate the UNIX forward slash to DOS backward slash:
```
tr '/' '\\' in
```

Translate the DOS backward slash to UNIX forward slash:
```
tr '\\' '\/' in
```

Squeeze all multiple spaces to a single space:
```
chkdsk | tr -s ""  " " in
```

Print a list of all words in a file:
```
tr -cs '[A-Z][a-z][0-9]' '[\012*]' file
```

Convert any text to: l(small letter), L(big letter), n(numbers) and x for rest::
```
tr "'[ -~][A-Z][a-z][0-9]'" '[x*96][L*26][l*26][n*10]'
```

ud_mgmt (UnixDos License Management)

Syntax:
```
ud_mgmt
```

```
UnixDos for Windows 95/NT License Management Utility (or)
UnixDos for Windows 3.X/MSDOS Management Utility
=========================================================


Please select from menu below:

I = Install UnixDos License
U = UnInstall UnixDos to move to another machine
V = View current UnixDos License

Q = Quit

Please enter your selection here:
```

Description:
UD_MGMT is the essential UnixDos utility manage and view your license:
- view your current status/remaining time during the 30 day demo period
- install a new license on a new machine
- move your license to a new machine

View Option (Windows 95/NT):
If you have <u>UnixDos license installed</u> you should see something like:
```
Installed: 11/17/1997 19:56:27
Users    : single
Code     : CODE:RXMS-UNIXDOS@POBOX.COM
Key      : K:G19A-6137-37
Serial   : 0
```

If you are <u>in the 30day demo period</u> you should see something like:
```
Used :     0.000 days (         5 seconds)
Left :    30.000 days (  2591995 seconds)
Start:  11/17/1997 19:59:53
Exp  :  12/17/1997 19:59:53
Code     :
Key      :
Serial   : 0
```

If you are <u>past the 30day demo period</u> you should see something like:
```
Used :    30.000 days (  2592002 seconds)
```

```
Left :   -0.000 days (         -2 seconds)
Start:  10/18/1997 20:04:45
Exp  :  11/17/1997 20:04:45
Code      : CODE:RXMS-UNIXDOS@POBOX.COM
Key       : K:G19A-6137-37
Serial    : 0
```

View Option (Windows 3.X/MSDOS):

If you have <u>UnixDos license installed</u> you should see (never expires):

```
Exp  : Never
```

If you are <u>in the 30day demo period</u> you should see something like:

```
Used :    0.000 days (          4 seconds)
Left :   30.000 days (   2591996 seconds)
Start:  Mon Nov 17 20:01:33 1997
Exp  :  Wed Dec 17 20:01:33 1997
```

If you are <u>past the 30day demo period</u> you should see something like:

```
Used :   30.001 days (   2592105 seconds)
Left :   -0.001 days (       -105 seconds)
Start:  Sat Oct 18 21:01:05 1997
Exp  :  Mon Nov 17 20:01:05 1997
```

UnixDos License Installation (Windows 95/NT/3.X/MSDOS):

```
Below are the steps of the UnixDos INSTALL PROCESS:

1. You enter your e-mail address or phone number you used for
the registration
2. You will receive your PERSONAL CODE
3. You submit your PERSONAL CODE with your order to:
   a) Online at http://www.ProfSoftware.com or
   b) per check of $60.00 to: Professional Software Solutions
                                1626 N. Wilcox Suite#252, LA, CA
90028

4. You receive your REGISTRATION KEY per email or mail/phone
5. Now you can enter your REGISTRATION KEY
6. UnixDos License will be installed

Do you want to continue? (Y/N):
```

UnixDos License Un-Installation (Windows 95/NT/3.X/MSDOS):

```
Below are the steps of the UnixDos UNINSTALL PROCESS:

1. You enter your e-mail address or phone number you used for
the registration
2. You enter your REGISTRATION KEY
```

3. You will receive the UNINSTALL CODE
4. UnixDos will be removed and CANNOT be re-installed on this machine!
5. You install UnixDos on a new machine
6. You run the UnixDos installation process on that new machine and
            get your new PERSONAL CODE
7. You submit your new PERSONAL CODE together with UNINSTALL CODE to:
   unixdos@pobox.com or mail to: Professional Software Solutions
   1626 N. Wilcox Suite#252, LA, CA 90028
8. You receive your new REGISTRATION KEY
9. You install UnixDos on the new machine

Are you sure to unistall UnixDos from this machine FOREVER? (Y/N):

uniq (Find unique lines/keys)

Syntax:
```
uniq [-cdu] [-t{c|\xx}] [-n[.m]] [+n] [input [output]]
(default find unique line and display each once)
-c      = count    : display counter in front
-d      = duplicate: display only duplicate items
-u      = unique   : display only unique items
-n[.m] = field    : skip {n} fields (include up to 'm')
+{n}    = char     : skip {n} char in field
-t{c}  = delimiter: define new delimiter (only with +/-n)
                    (default is tab, \xx = specify as hex value)
Example: uniq -t\7c -1.3 +2 -u x1
(Will use field 2,3,4 based in delimter | (hex 7c) skipping the
 first 2 character and print only single occurrences)
```

Description:
UNIQ will read the sorted input from 'in' (STDIN if omitted) and write each unique line to the output file 'out' (STDOUT if omitted). The input has to be already sorted in ascending order.

If the '-d' option is given UNIQ will only display keys which occur at least twice.

If the '-u' option is given UNIQ will only display keys which appear only once.

If the '-c' option is given UNIQ will display all keys with a counter of their respective repetitions in front.

If you specify a key (default is the entire line) UNIQ will check only the specified key area.

Example:
Display all file extensions with counter in the current directory:
```
ls | cut -d. -f2 | usort | uniq -c
```

unix2dos (Convert UNIX text files to DOS)

Syntax:
```
usage: unix2dos {file(s)}
file(s) = file(s) to convert
```

Description:
UNIX2DOS will convert each file from the UNIX text format (only Line Feed(LF) at the end of each line) to the DOS text format (Carriage Return(CR) and Line Feed(LF) at the end of each line).

Example:
To convert "list1.txt" and "list2.txt" into DOS format enter:
```
unix2dos list1.txt list2.txt
```

It will just show you the processed filenames and replace the original file contents:
```
list1.txt
list2.txt
```

uptime (Show elapsed time since reboot or reset)

Syntax:
```
uptime [-i]

-I    init: initialize a reboot
'uptime -i' should be in 'AUTOEXEC.BAT'
```

Description:
UPTIME will report the time elapsed since the last reboot.

When your computer reboots 'uptime -i' should be called (from AUTOEXEC.BAT) to mark the reboot.

Example:
```
uptime
```

May show:
```
1 day and 06:14:34 elapsed since reboot at:

    Wed Aug 05 07:18:00 1991
```

usrchar (Prompt user for single character)

Syntax:
```
usrchar [argument(s)|-d|-e] {lst}
{lst}          = list of answer codes (case insensitive)
-d      = disable       the escape sequence translation
-e      = enable again the escape sequence translation
The following escape sequences can be used in the arguments:
\b      = backspace
\c      = no new line (default is line feed at the end)
\f      = form-feed
\n      = newline
\r      = carriage return
\t      = tab
\v      = vertical tab
\\      = backslash
\0n     = octal code
\xn     = hex code
If the argument starts with '%' or '$' the contents of that
environment variable is echoed)
The errorlevel depends in the column in the {lst}:
usrchar "Type 1/2/3 or X:\c" 123X (exit code is 1=1, 2=2, 3=3,
X=4)
```

Description:
USRCHAR allows you to prompt the user for a single character.
It is not necessary to hit the return key - the keyboard input is processed directly.
You specify a list of all valid characters.
Depending on the position in the list USRCHAR will exit with the position (1=first, ...)
The case is not significant.
Wrong answers are responded to with a beep.

You can use special characters like the UnixDos UECHO command (see there for details)

Example:
```
usrchar  Do you want to copy (Y/N):\c yn
```

If the user types Y the exit value is 1, if N the exit value is 2.

usrprmpt (Prompt user for text)

Syntax:
```
usage: usrprmpt [argument(s)|-d|-e]
(The user input is written to 'usrprmpt.tmp')
-d      = disable      the escape sequence translation
-e      = enable again the escape sequence translation
-r      = the answer is required
The following escape sequences can be used in the arguments:
\b      = backspace
\c      = no new line (default is line feed at the end)
\f      = form-feed
\n      = newline
\r      = carriage return
\t      = tab
\v      = vertical tab
\\      = backslash
\0n     = octal code
\xn     = hex code
If the argument starts with '%' or '$' the contents of that
environment variable is echoed)
The errorlevel is the number of bytes the user entered
```

Description:
USRPRMPT allows you to prompt the user for a text.
The user can enter any length of characters. If you need a responds and want to suppress a non-answer (when the user hits just ENTER) you can specify the "-r" option on the command line to enforce an answer. Wrong answers are responded to with a beep.

The user input is written to the file "usrprmpt.tmp".

USRPRMPT will return the number of characters entered as the exit value, so you can test for no response.

You can use special characters like the UnixDos UECHO command (see there for details)

Example:
```
usrprompt  Enter the filename:\c
if not errorlevel 1 goto END
uecho You entered @usrprmpt.tmp
```

If the user enters no name the exit value is 0, otherwise the number of characters

uudecode (Decode a transmitted binary file)

Syntax:
```
uudecode [input]

input      input file name (if omitted STDIN is used)
      This file has to be created by UUENCODE!
```

Description:
UUDECODE will decode a file prepared by UUENCODE.
The input file contains only printable ASCII character.

The filename of the original file is embedded in the input file and will automatically be used for the decoded data.

The UUENCODE/UUDECODE programs are an excellent tool for data transmission where only ASCII character can be sent/received.

Example:
Encode a binary file:
```
uuencode uudecode.exe temp.txt
...(transmission)...
uudecode temp.txt
```

uuenode (Encode a binary file to text for transmission)

```
uuencode [input] {output|-}

input          input: binary file to be encoded (if omitted
STDIN is used)
output         output: ASCII test output file (if '-' STDOUT is
used)
```

Description:
UUENCODE will convert a binary file into a text file which contains only printable ASCII character.

The filename of the input file is embedded in the output file.
If STDIN is used UUENCODE will use 'stdin'.

The UUENCODE/UUDECODE programs are an excellent tool for data transmission where only ASCII character can be sent/received.

Example:
Encode a binary file:
```
uuencode uudecode.exe temp.txt
...(transmission)...
uudecode temp.txt
```

wait_end (Wait for process completion)

Syntax:
```
usage: wait_end {prog} [args ...]
{prog}    = name of the program you want to start (required)
[args]    = arguments for {prog} (optional)
```

Description:
WAIT_END will start the specified program, but will wait until has completed.
This can be very usefull if you are creating a batch process with user interaction and you have to make sure the user has answered before the batch process continues!
Wait will parse the arguments and enclose them in double quotes to protect it as ONE argument, if it finds one of the following special characters:
space, '>', '<' or '|'.

Example:
In this example we want to present the user with a display of a text file (old.txt).
If the user clicks on the "Delete" button (exit code 1) the file will be deleted,
of not nothing happens:
```
set FILE=old.txt
wait_end txtbox "-BDelete|Keep" %FILE%
call saveexit
test -e 1 run: rm -l %FILE%
```

wc (Word/Line counter)

Syntax:
```
wc [-clntw] {file(s)}
-c      = CHARS   : count only characters
-l      = LINES   : count only lines
-n      = NOT NULL: display only files with more than 0 lines
-t      = TOTAL   : display only the total line
-w      = WORDS   :  count only words
default is -lc
(if no files are given it will read from stdin)
```

Description:
WC will report the number of lines and characters for each specified file.
If no file is given WC will use STDIN.

If there is more than one input file specified WC will append the totals at the end with
'TOTAL' in place of the filename.

You can switch individual counter on with the options.

Example:
Count the number of text files in the current directory:
```
ls *.txt | wc -l
```

Display the lines and bytes of all C source files starting with a and b in the current
directory:
```
wc  [ab]*.c
```

Might show:
```
   181      6094   ar.c
   300     14254   banner.c
    97      2626   basename.c
  2142    123655   bbanner.c
   394      9254   bdiff.c
    28      1027   bname_t.c
  3142    156910   TOTAL
```

Display only the total lines of all C source files starting with a and b in the current
directory:
```
wc  -lt [ab]*.c
```

Might show:
```
  3142    TOTAL
```

which (Locate a program)

```
which {program name1 [,name2...]}
names can be also a regular expression:
which make*     will find all programs starting with 'make'
which make.bat  will find only specifically 'make.bat' files
which *make*    will find all programs containing 'make'
which make      will find all programs with the name 'make'
Programs are all files with extensions: .exe, .com, .bat or
any extension listed in the PATHEXT environment:
set PATHEXT=.exe;.com;.bat;.sh;.cmd
(which does not use the UnixDos argument expansion)
```

Description:
WHICH will search in all directories given in the PATH environment and list any match for the program(s) specified.
You can do the following search types:
- detailed name: `which myprog.exe`
- just name: `which myprog`
    here WHICH will look for any program with exactly the name "myprog" which has the
    extension ".bat", ".exe" or ".com". To specify your custom list define PATHEXT:
    `set PATHEXT=.bat;.exe;.com;.cmd;.sh`
- name expression: `which my*`
    Here WHICH will search for any program starting with "my" using the program
    extensions as specified above.

You can be very specific (filename with extension) or just the filename (WC will find any extension) or do a file expression match (all files containing sort).

Since you will always enter search expressions on the command line the automatic UnixDos argument expansion is disabled for the WHICH command.

Example:
Display all files containing 'sort':
```
which *sort*
```

Might show:
```
C:/U/sort.txt
C:/UnixDos/usort.exe
C:/DOS/sort.exe
C:/UNIFY/BIN/usort.exe
```

Find the 'which.exe' command:
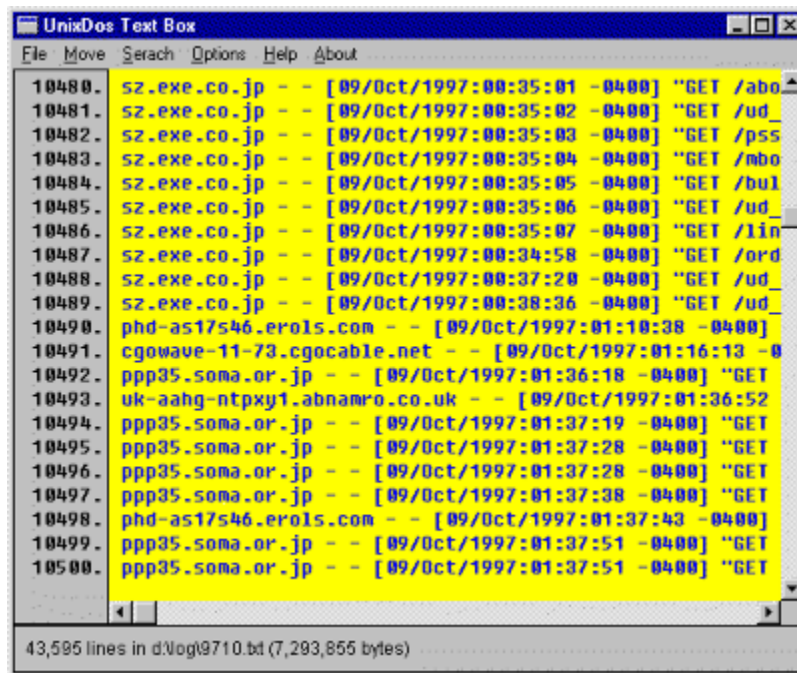```
which which.exe
```

Might show:

`C:/UnixDos/which.exe`

# TXTBOX (Advanced interactive text viewer)

## Description

TXTBOX allows you to open any text file and view it in a user friendly environment with <u>command line options</u> and optional interactive <u>Button Mode</u>
and many <u>examples</u>.
(Another fine product from: **Professional Software Solutions**)



The following features are available:
- Unlimited file size
- maximum line length is 10,000 characters
- horizontal scrolling
- vertical scrolling
- line number display toggle
- goto specified line
- complete restore of all parameter after exit
  (filename, windows position, search parameter, current line, colors, fonts ...)
- regular expression and straight text search
- automatic wrap option for searches
- ignore case option for searches
- color selection for text and background
- font selection for text display
- initial position device independent using percent instead of pixels
- total line count and file size display
- optional button feature to allow user to select choices

- complete control via command line options (<u>see below</u>)
- no disturbing wrapping of long lines
- exit code depending on user response
  (0=exit, 1=button1, 2=button2, 3=button3, 99=error)
  (allowing you to implement TXTBOX in batch files for decisions and branching)
- storage of exit code in special file for later retrieval

## Button Mode

TXTBOX allows you to display the text and then allow the user to respond by pressing a button.
You can specify 1 to 3 buttons with your custom text. The buttons are displaye below the text centered to the window. Each text for a button can be up to 16 characters long.
You can specify that a response via the button click is required (-B) or leave it optional (-b)

<u>Example:</u>
Below you can see two buttons "Accept" and "Don't accept".
This is achived by the command line option (the quites are needed to include spaces and pipe):
```
txtbox "-bAccept|Don't Accept"
```



To make the button selection required use the capital B:
```
txtbox "-BAccept|Don't Accept"
```
To switch the button default off again use the empty -b switch:
```
txtbox -b
```

## Command line options
```
txtbox [options] [file]
options: (can be used in any sequence)
+{n}   = Number   : start display from line number {n}

-b[t1[|t2[|t3]]]
```

```
            = Button    : define up to three buttons not required
                            t1 = text for button1
                            t2 = text for button2
                            t3 = text for button3
-B[t1[|t2[|t3]]
            = Button    : define up to three buttons click
required!
                            t1 = text for button1
                            t2 = text for button2
                            t3 = text for button3
-ct{clr}  = ColorTxt: define color for text
                            clr = RGB(red,green,blue) (0-255) or
                            clr = black,red,green,blue,yellow,
                                cyan,magenta,white,grey,orange
-d        = Default : restore all defaults
-e{fn}    = ExitCode: specify the file to store the exit code:
                            0  = regular exit
                            1  = button1 clicked
                            2  = button2 clicked
                            3  = button3 clicked
                            1XX= error
                            by default code in: %TEMP%\txtbox.tmp

-f{p,n,w,i}= Font    : use specified font for text:
                            p = point size (1 - 100)
                            n = font name
                            w = weight (100-999) bold=700,
regular=400
                            I = italic (0=off, 1=on)
-h{hd}    = Header   : use 'hd' as header for TXTBOX:
                            the following special command are
available:
                            &B   = current filesize
                            &C   = current column offset
                            &E   = current maximum line length
                            &F   = current filename
                            &L   = current line number
                            &M   = current total number of lines
-l{0|1}   = LineNbr  : show line numbers -l0=off, -l1=on
-o{Col}   = Column   : goto column 'Col' (default is 0=first
column)
-p{x,y,cx,cy}
            = Position : open window at position:
                            all values are in percent (0-100)
allowing you
                            device independent positioning!
                            x  = upper left corner horz position in
```

```
percent
                           y  = upper left corner vert position in
percent
                           cx = width in percent
                           cy = height in percent
-s{c,l}   = Scroll   : define overlapping Scroll parameter
                           c = cols to keep for page left/right
scroll
                                (0=no overlap, 1-80=cols,
10=default)
                           l = lines to keep for page up/down (1)
                                (0=no overlap, 1-30=lines,
1=default)
-t{n}     = Tab       : set number of space for TAB character
(default 8)

[file]    = file to be viewed initially
            (can be placed anywhere also between options)
            (no file is OK, screen will just be blank)
```

<u>Example1 (button - optional):</u>

This example will display file '\temp\oldlog,.txt' with two button below "Delete" and "Keep".
The Text is displayed in blue with yellow background.

```
txtbox  "-bDelete|Keep" \temp\oldlog.txt -ctBlue -cbYelloW
```

<u>Example2 (button - required, interactive, exit code):</u>

To make example1 part of an interactive batch file prompting the user if the file should be deleted with a required response:

```
wait_end txtbox "-BDelete|Keep" \temp\oldlog.txt  -ctBlue
-cbYelloW
call saveexit
test -e 1 run: banner delete
```

WAIT_END, SAVEEXIT and TEXT are UnixDos programs at http://www.unixdos.com.
WAIT_END will execute the specified program but wait until it is complete!
SAVEEXIT will save the last error level for later retrieval and TEST.EXE.
TEST is testing the expression (in this case the last exit code) and executing the specified program,
you could also use IF ERRORLEVEL {n}.

<u>Example3 (loading a font):</u>

You can select an existing font already at the command line.
In this example we use "Times New Roman" with point size 15 and bold weight (700) in italc:

```
txtbox \temp\oldlog.txt "-f15,Times New Roman,700,1"
```

Note the double quotes around the font option in order to pass the following arguments into TXTBOX as ONE argument (not broken up into several arguments)

Example4 (show lines, use header, start at a line):
In this exmaple we switch the line number display on, start at line 893 and use a custom header:
```
txtbox -l1 +893 "-hFile: %F at line %l or %m" \temp\oldlog.txt
```

Note the double quotes around the header option in order to pass the following arguments into TXTBOX as ONE argument (not broken up into several arguments)

Example5 (window positioning,column offset,scroll parameter):
In this exmaple we will request a specific position on the screen:
Upper left hand corder at left egde and in the middle of the screen. (0,50)
Width is 70% of the screen and height is 40%.
The display will start at column 40 and use no overlap for line sor columns:
```
txtbox -p0,50,70,40 -c40 -s0,0
```

Note that we don't specify any filename in which case TXTBOX will use automatically.the filename from the last session.

Example6 (reset):
Sometimes you want to start from scratch; remove all the previous settings.
For this situation the '-d' option has been included, which will wipe a clean slate and only add the options you specify to the internal default:
```
txtbox -d -p0,50,70,40 -c40 -s0,0
```

Note the parameter from the last session is stored in the same directory where TXTBOX has been called in the file TXTBOX.DAT. So if you want to maintain several sets of defaults you can store and restore this file at the times you need them.

For more information contact:

Professional Software Solutions
1626 North Wilcox Suite #252
Los Angeles, California 90028
pss@pobox.com
http://www.unixdos.com