

BATCH FILES

and MEMORY MANAGEMENT

By Bill Hayles

An occasional course for DOS gurus

A BRIEF EXPLANATION OF BATCH FILE COMMANDS

From the original, found on your floppy disk, which is written in pure ASCII so that a batch file can be written to print it out!

Batch files can contain any command which could be issued from the command line, i.e.

COPY M:\DOWN\THE\GARDEN\PATH\DEEPDOWN.EXE Q:\FILES\NOTDEEP.EXE

In addition, there are other commands, which are normally used in batch.

====

CALL <batchfile>

Only available in versions 3.3 or greater.

CALL runs the batchfile subtask from within the main file, and returns to the main file when the subtask has finished. If you run the subtask from the main file without the CALL command, after the subtask is finished it will return you to the prompt without the main file ever finishing.

Suppose you want to use NEWFILE.BAT from within another file, i.e.

```
C:
CD \TEMP
COPY A:\*.SYS
CALL NEWFILE
COPY A:\*.EXE
CALL NEWFILE
```

This will execute NEWFILE.BAT twice.

```
C:
CD \TEMP
COPY A:\*.SYS
NEWFILE
COPY A:\*.EXE
NEWFILE
```

In this case, the command prompt is returned after NEWFILE is run for the first time, hardly what we want!

====

ECHO

ECHO ON turns the command echoing feature

ECHO OFF turns it off

ECHO (on its own) will display whether it's on or off

ECHO [message] will display the message on the screen

ECHO. will leave a blank line in ECHOed messages

@ECHO OFF will turn ECHO off without displaying that it has done so!

=====

=

FOR

General syntax:

FOR %%a in (set) DO DOS command [parameters]

IMPORTANT

When used in batch files, the variable consists of TWO percent signs and a single LETTER. Numbers are not permitted (these are the replacement variables!). Also the IN and DO are compulsory.

Examples

To type all .TXT files to screen from a specific directory:

```
FOR %%A IN (*.TXT) DO TYPE %%A
```

To copy all the executable files from one directory to another:

```
FOR %%A IN (*.COM *.EXE *.BAT) DO COPY C:\DIRONE\%%A C:\DIRTWO\%%A /C
```

=====

====

IF

An awkward one, this. The official line is that it "performs conditional processing in batch files". Very illuminating!!.

It can only be used to test in three ways, all of which can be reversed by the use of NOT, i.e.

IF EXIST filename command

only does something if the specified file exists, i.e.

```
IF EXIST C:\COMMAND.COM ECHO The hard disk has a command processor.
```

or

IF NOT EXIST filename command

IF string1==string2 command

note the two equal signs.

string1 and string2 can either be literals or replacement variables, i.e.

```
IF %1==Bill Hayles echo The tutor did this
```

IF ERRORLEVEL number command.

Explanation.

When they finish and return you to the command line, most programs return a code to DOS, called the ERRORLEVEL. An errorlevel code of 0 means a successful conclusion, others generally mean something wrong. There is no general rule about which problem gives which errorlevel. Our BE batch enhancer file makes use of ERRORLEVEL to enable us to make a choice.

DOS tests the errorlevel in a dumb way. The line:

```
IF ERRORLEVEL 7 Echo the errorlevel is 7
```

would echo the message for all errorlevels of 7 OR GREATER (!).

Errorlevels therefore have to be tested for in descending order, and care taken to "jump" using GOTO. Consider the behaviour of the following files:

(a)

```
IF ERRORLEVEL 4 ECHO ERRORLEVEL IS 4
IF ERRORLEVEL 3 ECHO ERRORLEVEL IS 3
IF ERRORLEVEL 2 ECHO ERRORLEVEL IS 2
IF ERRORLEVEL 1 ECHO ERRORLEVEL IS 1
IF ERRORLEVEL 0 ECHO ERRORLEVEL IS 0
```

(b)

```
IF ERRORLEVEL 0 ECHO ERRORLEVEL IS 0
IF ERRORLEVEL 1 ECHO ERRORLEVEL IS 1
IF ERRORLEVEL 2 ECHO ERRORLEVEL IS 2
IF ERRORLEVEL 3 ECHO ERRORLEVEL IS 3
IF ERRORLEVEL 4 ECHO ERRORLEVEL IS 4
```

(c)

```
IF ERRORLEVEL 4 GOTO 4
IF ERRORLEVEL 3 GOTO 3
IF ERRORLEVEL 2 GOTO 2
IF ERRORLEVEL 1 GOTO 1
ECHO ERRORLEVEL IS 0
GOTO END
:4
ECHO ERRORLEVEL IS 4
GOTO END
:3
ECHO ERRORLEVEL IS 3
GOTO END
:2
ECHO ERRORLEVEL IS 2
GOTO END
:1
ECHO ERRORLEVEL IS 1
:END
```

Example (c) is the only one which works as planned.

ERRORLEVEL is often used to test for something happening. Returning to our old favourite MOVEIT.BAT, it can be used to test for a successful copy, i.e.

```
COPY %1 %2 /C
IF NOT ERRORLEVEL 0 GOTO ABORT
DEL %1
ECHO One file moved
GOTO END
:ABORT
ECHO Something wrong! Delete not done.
:END
```

=====

GOTO

Has already been used in the examples for IF. The syntax is

GOTO label

where label can be almost anything

The batch file will then transfer execution to a line starting with a colon and the label, i.e. :label

Example:

```
REM Example of GOTO
@ECHO OFF
GOTO END
ATTRIB -R -S -H C:\*.*
DEL C:\*.*
REM WHOOPS, WE'VE JUST WIPED OUT THE SYSTEM ON THE HARD DISK
:END
REM NO WE HAVEN'T, WE JUMPED STRAIGHT HERE.
```

When using GOTOs, you have to be aware that once the label has been jumped to, execution continues from that point, including any other labelled sections. It is often necessary to use two GOTOs in order to use self-contained sections. See example (c) under IF

=====

PAUSE

An easy one! When PAUSE is encountered in batch file processing, execution stops until a key is pressed. PAUSE echoes on screen the message "Press any key to continue . . ." which is often sufficient. However, if you want to PAUSE with a message of your own, this is easily achieved:

```

PAUSE > NUL
REM The Press any key message is echoed to nowhere rather than the screen.
ECHO I'm on strike until you press a key
FOR %%A IN (*.*) DO IF NOT EXIST B:\FILES\%%A COPY %%A B:\FILES\%%A
REM Work it out!
ECHO All done.

```

```

=====
=====

```

SET

Although not a batch file command as such (it can sometimes be useful from the command line), here is a good place to discuss SET. It makes use of a small area of memory, typically 256 bytes, called the ENVIRONMENT, in which are stored environmental variables (EVs). They are used to overcome the problem where you want to store some information specific to a situation. For example, many programs make use of temporary files. The good ones allow you to specify where these temporary files are to be kept. You do this by specifying an EV, often in AUTOEXEC.BAT

```
SET TEMP=C:\TEMP
```

The program looks in the environment for the variable TEMP, finds it and knows where to store the files.

So, the syntax is:

```
SET variable=string
```

Certain EVs are set whenever DOS loads. These include PROMPT, PATH and COMSPEC.

To use an EV in a batch file, use a % sign at BOTH ends of its name, i.e. following on from above

```
ECHO %TEMP%
```

will echo C:\TEMP to screen.

SET with no parameters displays all the currently set EVs.

```

=====
=====

```

SHIFT

The not-very-useful one.

In the unlikely event that you write a batch file which needs more than the ten available replacement variables, you can get out of trouble with SHIFT. What SHIFT does is to discard %0, move %1 to %0, %2 to %1 and so on. The first parameter that didn't have a variable is moved to %9. Confused?. Maybe this will help

```
@ECHO OFF
REM Assume file is named manycopy.bat
REM This batfile copies any number of files to a directory.
REM Syntax manycopy directory file1 file2 .....
SET destiny=%1
:GETFILE
SHIFT
IF "%1"==" " GOTO END
REM run out of parameters
COPY %1 %DESTINY%\%1
GOTO GETFILE
:END
SET DESTINY=
REM "UNSET" Environmental variable
ECHO All done
REM Freely adapted from the MS-DOS5 handbook example
```

ANSI.SYS

A display system for ASCII.

ANSI.SYS is a device driver for the keyboard and screen, loaded in CONFIG.SYS. There are also third-party ANSI emulators which may be .COM or .EXE files and are thus loaded in AUTOEXEC.BAT or from the command line.

ANSI provides extra options for programs that need to move the cursor, alter the screen display or assign keyboard use. These options take the form of ANSI ESCAPE SEQUENCES, so called because they all start with the ASCII escape code (27 or 1B hex). Although its use has lessened as computers get more sophisticated, some applications still make use of ANSI and expect it to be present. Without an ANSI driver, none of what follows will work.

The general form of an ANSI escape sequence is:

The ESC character (ASCII 27)

The [character (ASCII 91)

One or more numbers separated by semicolons

A single command character.

In order to create an ANSI sequence, you need to force the escape character into your word processor or editor. In both the DR-DOS and MS-DOS 5/6 editors, you do this by typing ^P (control-P) and then the character you want embedded (i.e. Esc). Although this feels strange at first, you soon get used to it.

The commonest use of ANSI for the batch file writer is to make their screen displays more colourful. By far the easiest way to do this is to create a text file to display, THEN embed the ANSI sequences, without altering the formatting. The file can then be called from a batch file by the TYPE command.

Here is a list of ANSI sequences - note my syntax:

ESC is the escape character

[is the square bracket (I)

x indicates row number (1 at top)

y indicates column number (1 at left)

other letters are typed "as is" and are case significant!

To position the cursor:

ESC[y;xH

positions the cursor according to the values of x and y

To move cursor up:

ESC[yA

moves the cursor up y rows

To move cursor down:

ESC[yB

moves the cursor down y rows

To move cursor right

ESC[xC

moves the cursor right by x columns

To move cursor left

ESC[xD

moves the cursor left by x columns

To save the current cursor position

ESC[s

and to restore to the saved position

ESC[u

which returns to 0,0 if no save command has been issued.

To clear the screen:

ESC[2J

has the same effect as the DOS command CLS

To change display mode:

general form: ESC[n;n;----nm

where n is one of the figures below (several in one sequence)

Number	Effect
0	Return to steady white on black
1	Bold (Bright)
4	Underscore (MDA only)
5	Flashing
7	Inverse (black on white)
8	Invisible (black on black)
30	Black foreground
31	Red foreground
32	Green foreground
33	Brown foreground
34	Blue foreground
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black Background
41	Red Background
42	Green Background
43	Brown Background
44	Blue Background
45	Magenta Background
46	Cyan Background
47	White Background

Note: Yellow is bright brown!

Set and reset screen modes

Set screen mode: ESC[?nh

Reset screen mode: ESC[?nl

where n takes the following values:

0	40x25 mono
1	40x25 colour
2	80x25 mono
3	80x25 colour
4	320x200 mono (CGA mode 0)
5	320x200 colour (CGA graphics mode)
6	640x200 mono (CGA hi-res mono mode)
7	Enable (set) or disable (reset) wrapping at the ends of lines. If disabled, extra characters are lost.

Key assignments

One of the seldom used backwaters of ANSI is the ability to redefine keys. The general form is:

```
ESC[a;b
```

where *a* is the existing character for the key and *b* the new one, as ASCII codes. For example: ESC[65;66p would cause a B to come on the screen every time you pressed the A. The lower case *a* and *b* would not be affected. Another way, if you don't know the ASCII codes is to define the keys directly, thus: ESC["A";"B"p

You can also assign a set of keystrokes to a key. In other words, if you issued the command: ESC["A";"Bill"p every time you typed the A the word Bill would appear.

This really comes into its own when used in conjunction with the function keys. These are assigned Extended Character Codes (ECCs) The table below lists the ECCs

Key	Code	+Shift	+Ctrl	+Alt
F1	59	84	94	104
F2	60	85	95	105
F3	61	86	96	106
F4	62	87	97	107
F5	63	88	98	108
F6	64	89	99	109
F7	65	90	100	110
F8	66	91	101	111
F9	67	92	102	112
F10	68	93	103	113

ANSI recognises ECCs by a 0 in front of the. I.e. 0;68 means F10 and 0;101 means Control-F8. Thus to assign the word "Macro" to the Alt-F3 key, the sequence would be: ESC[0;106;"Macro"p

It is difficult, but not impossible, to generate ANSI sequences direct from the keyboard. The problem is in the escape code, and the answer lies in the DOS prompt. As you should remember, \$e is a special code in the prompt to give an escape code, so we could do something like:

```
Set oldprompt=%prompt%
rem so we don't lose what we've already got!
prompt $e[0;106;"Macro"p
rem at this point the word macro is assigned to Alt-F3
but we have no prompt at all. So at the unprompt type
Prompt %oldprompt%
and all should be restored.
```

If you are using 4DOS, then at the 4DOS prompt ^X gives an escape character - very useful!

MEMORY MANAGEMENT.

The many different types of memory encountered on a DOS-based PC is NOT a function of the PC itself. There is a modern operating system, OS/2, which makes no attempt to be backwardly compatible, and for which memory is memory is memory - up to a theoretical limit of 16 Gigabytes. The mish-mash of different types of DOS memory has come about purely as a result of history.

Types of Memory and the historical background.

DOS was originally written for the Intel 8086 processor. This could address (control) a maximum of 1Mb of memory, and so DOS was written to these limits. 640 Kb was allowed for DOS and applications, called BASE memory, while the other 384K was the area where video cards and other hardware devices were to be found (called UPPER memory). Today, the first megabyte of memory is called CONVENTIONAL memory. The 640K limit still applies, and even today, all programs running under DOS (including Windows) must somehow load themselves into whatever remains of the 640K after DOS and any TSRs have pinched their bit.

It was soon found that 640K was insufficient to run the large programs which were being developed, and a way was found to overcome it, called the LIM EXPANDED memory specification. This took the form of a memory card, usually 1 or 2Mb controlled by a device driver loaded in CONFIG.SYS. This device driver paged the expanded memory through a convenient 64K unused area in upper memory. DOS knew nothing about this, but the application did, and could make use of it. Lotus 1-2-3 was one of the first to use expanded memory.

The 80286 chip can talk to 16Mb of memory, which can all be on the motherboard and does not need to be accessed by a device driver. Any memory available in this way over 1Mb is called EXTENDED memory. Unfortunately, DOS can't see it. On most 80286 computers, there's nothing much you can do with extended memory except to use it for Ramdisks and disk caches.

The 80386 and all subsequent chips were designed to address more memory than you can think of (many gigabytes). They have one outstanding feature - the virtual 8086 mode which enables them to pretend to be multiple 8086 chips, and thus to multitask 8086 programs (i.e. all DOS software). This feature is controlled by the EMM386 family of device drivers. All modern versions of DOS include one, called EMM386.SYS, EMM386.EXE or something similar. MS-DOS 5 uses EMM386.EXE to confuse matters since it is a device driver and is loaded in CONFIG.SYS.

There are two other types of memory you will encounter.

HIGH memory is that section of memory between 1024K and 1088K. Due to a fortunate bug in the design of the 286 and later chips, this 64K chunk can be used to store DOS and other TSRs.

VIRTUAL memory is the term for a disk emulating RAM. When some programs run out of memory, they will use some hard disk space instead. Although this works, it is very slow. It is often called SWAPPING.

Where this leaves the modern PC I'll discuss later.

Maximising base memory.

Don't forget that although our software may be able to make use of many megabytes of memory, it must initially load into base memory. How much base memory it needs varies widely, but a figure of 575K is not untypical for a modern application. It doesn't matter how many megabytes is available in total, if there is not 575 K base memory free, the application will not load. And that's that! It therefore makes sense to free as much base memory as possible. On an XT or 286, this simply means checking your CONFIG.SYS and AUTOEXEC.BAT files, and seeing if you really need all the device drivers and TSRs (terminate and stay resident programs) specified. If you can do without them, don't load them. On a 286, you may also be able to make use of HIMEM.SYS or something similar, discussed below. MEMORY MANAGEMENT on 386 or better using DOS 5 or greater.

What follows only applies if you have a 386, 486 or Pentium based PC and are using MS, PC or DR DOS version 5 or later. It does not necessarily apply to Windows 95
except when operating in MSDOS mode

The advice about removing unwanted device drivers or TSRs still applies.

BASIS.

Although there is only 640K base memory, there is 1024K total conventional memory plus the 64K high memory. The 384K upper memory is not fully utilised, and there is usually about half of it free. Ways have been found to load some of DOS, and many TSRs in this memory, thus freeing space in base memory. What follows is for MS-DOS 5 but similar principals apply for DR-DOS, QEMM386, 386MAX and others of their ilk.

Sometimes there is a memory maximisation facility that comes with the package, i.e. in MS-DOS 6 and 6.2 there is a MEMMAKER.EXE. These will improve your base memory above what it would be by default, but you can still do better manually if you know what you're doing!

HIMEM.SYS is a device driver to manage extended memory. It enables modern software which can use extended memory to do so. It is also needed to load Windows. It also enables access to high memory. It therefore pays to make a line such as:

```
DEVICE=C:\DOS\HIMEM.SYS
```

the first in your CONFIG.SYS.

HIMEM.SYS has a great number of switches, none of which you're likely to need.

Upper memory is controlled by EMM386.EXE, which also acts as an expanded memory emulator for the dwindling bunch of programs that use it. It has a frighteningly complex syntax. Don't worry, you'll never need most of the switches - I'm not even going to list or explain them. All you really have to decide is if you need expanded memory emulation, and you obviously have to know where the file it to give the full path. If you don't want expanded memory the line will read something like:

```
DEVICE=C:\DOS\EMM386.EXE NOEMS
```

whereas if you do, use:

```
DEVICE=C:\DOS\EMM386.EXE RAM
```

IMPORTANT.

EMM386.EXE won't work without HIMEM.SYS, which MUST be loaded first. If you play with the EMM386 switches (which I haven't listed) you may lock yourself out of the system. Have a system floppy with an editor handy!

Only one program can normally access high memory, so the most obvious candidate is DOS itself. This is carried out by the CONFIG.SYS line:

```
DOS=HIGH,UMB
```

placed after the EMM386.EXE line.

If the management system detailed above is installed, devices and TSRs can be loaded in upper memory provided there is room. For device drivers loaded in CONFIG.SYS this is enabled by using DEVICEHIGH instead of DEVICE, i.e.

```
DEVICEHIGH=C:\DOS\RAMDRIVE.SYS 512 /A
```

Some device drivers cannot be loaded high; you will have to experiment

To load a TSR in upper memory, use the LOADHIGH command. Although this can be done from the command line, it is more common to do so in AUTOEXEC.BAT, i.e.

```
LOADHIGH C:\DOS\KEYB UK,,C:\DOS\KEYBOARD.SYS
```

Sensible memory management can free as much as 627K to be available at the command prompt.

CONFIG.SYS and AUTOEXEC.BAT

=====

These are the two user-defined text files which determine the system configuration loaded at boot time.

AUTOEXEC.BAT is an ordinary batch file and as such can contain any commands acceptable to a batch file. The only special feature it possesses is that it is AUTOMATICALLY EXECUTES when the system boots.

CONFIG.SYS is a very special file. It is NOT a batch file, and only accepts certain commands. It CONFIGURES the SYSTEM, and is the place where device drivers are defined and loaded. A device driver is a special file which controls or modifies a piece of hardware. For example, the device driver ANSI.SYS is a primitive way of controlling screen colour in text mode. There are hundreds of device drivers, and your system will obviously contain only the ones you need. Very often when you buy a new piece of hardware, such as a mouse, it will come with its own device driver on floppy, which you must copy to your hard disk and load in CONFIG.SYS. Since CONFIG.SYS is the first file to be processed after the system has booted, it always needs full file paths.

Aside from ANSI.SYS, the commonest drivers found in CONFIG.SYS are the memory managers discussed separately. There are also other customisations defined in the file, as follows.

BREAK [on|off]

If **BREAK** is on, the system searches frequently for the abort combination CTRL-C or CTRL-BREAK, thus halting program execution. This means that a program which is running out of control but has not hung can be brought swiftly to a halt. If **BREAK** is off, the keyboard is only checked ("polled") when the program next reads or writes to a device.

In practice, your computer will run marginally slower with **BREAK** off, but you have more control. The choice is yours!

BUFFERS [number]

Buffers are small blocks of memory used during disk reads and writes as a primitive cache to speed up operations. You can have up to 99, the default is 15.

The number to specify is a compromise as the more buffers, the faster the disk access, but each buffer takes 512 bytes of conventional memory. Unless you have a specialised disk cache loaded, a figure of 20 to 30 is considered a sensible number.

COUNTRY

Syntax:

COUNTRY=number,cp,[path]\country.sys

The COUNTRY.SYS file provides date, time and currency information as appropriate. You may also specify a code page other than the default for your country (cp), actually very rarely needed. So, assuming that COUNTRY.SYS could be found in the DOS directory, a typical hard disk entry could be:

```
COUNTRY=044,,C:\DOS\COUNTRY.SYS
```

044 is the code for the UK; note the two commas.

Tip: The country code is the same as the International Dialling Code.

DEVICE

Has already been discussed. The device drivers which come with DOS include:

ANSI.SYS - enables ANSI escape sequences

DRIVER.SYS - enables non-standard floppy drives

DISPLAY.SYS - enables code page switching - rarely used

EMM386.EXE - memory manager - very important

HIMEM.SYS - memory enabler - very important

PRINTER.SYS - enables code page switching - rarely used

VDISK.SYS or - define a RAM disk, quite useful if you

RAMDRIVE.SYS - have the spare memory.

DRIVPARM

Sets the physical characteristics for a floppy drive. Very seldom needed nowadays.

FCBS

To maintain compatibility with DOS version 1, there is still the facility to specify the number of open files by means of File Control Blocks. Only a very few very old applications need it. Ignore!

FILES

Syntax:

FILE=number

Specifies the number of files that DOS can have open at any one time. The default varies on the DOS version, and this is one parameter that should always be specified in CONFIG.SYS. Although having FILES set to a large figure takes up a small amount of memory, it is better to err on the side of generosity.

FILES=30 is generally enough; if you multitask Windows applications consider setting FILES=50

INSTALL

Is a new feature, only found in versions 5 and above, that enables you to install Terminate and Stay Resident programs (TSRs) in CONFIG.SYS rather than AUTOEXEC.BAT

Its use is never compulsory.

Example:

```
INSTALL=C:\DOS\KEYB.COM UK
```

loads the UK keyboard driver.

LASTDRIVE

Syntax:

```
LASTDRIVE=[letter]
```

By default, DOS only recognises drives A to E. If you need more, you must set the LASTDRIVE parameter. Since LASTDRIVE takes up very little memory, many people just set

```
LASTDRIVE=Z
```

and leave it at that. You don't need to have a drive just because it's available.

STACKS

Syntax:

```
STACKS=n,s
```

Where n is the number of stacks, and s is the size in bytes of each stack.

Stacks are used to store information from hardware while it is being processed.

If you set STACKS to 0,0 (the default), the information is stored by the program and not by DOS. This works for almost all programs except Windows. If you use Windows, you are advised to use the figures

```
STACKS=9,256.
```

Don't ask *me* why, ask Microsoft!

SHELL

Syntax (typical)

```
SHELL=[path]\COMMAND.COM /E:nnnn /P
```

SHELL defines where DOS will find your command processor, and what it is. If there is no SHELL line in CONFIG.SYS, DOS will assume COMMAND.COM in the root directory with an environment of 256 bytes and AUTOEXEC.BAT also to be run from the root directory if present.

If SHELL cannot find the specified processor, or there is no SHELL line and no COMMAND.COM in the root directory, DOS will issue a blunt message and lock solid.

/E:nnnn defines the size of the environment in bytes (default 256, sensible figure 512)

/P tells DOS to load the command processor as the PRIMARY SHELL (i.e. it cannot be unloaded or EXITed from. Some versions allow for a different file to be specified other than AUTOEXEC.BAT :filespec after the /P. I.e. if the (DR-DOS) line were:

```
SHELL=C:\DOS\COMMAND.COM /E:512 /P:C:\MYEXEC.BAT
MYEXEC.BAT would be run instead of AUTOEXEC.BAT
```

The command processor does not have to be COMMAND.COM. Those of us which use 4DOS specify it in the SHELL line, i.e.

```
SHELL=C:\4DOS\4DOS.COM [many parameters]
```

In which case COMMAND.COM need not be present anywhere

AUTOEXEC.BAT

As already stated, this can accept any normal batch file commands. As it is AUTOMATICALLY EXECuted on boot, it is a convenient place to specify universal parameters. These include the environmental variables (SET), sometimes a mouse driver, and any TSRs demanded by applications.

From the above, you may surmise, correctly, that there is no such thing as a typical AUTOEXEC.BAT. You have to tailor it to your own particular needs. There are, nevertheless, two lines present in almost all of them, specifying the keyboard and the prompt.

Keyboard layouts vary slightly from country to country, and to cater for this DOS has a file called KEYB.COM. This takes information from a second file called KEYBOARD.SYS, and so the syntax is:

```
[path]KEYB country,,[path]KEYBOARD.SYS, i.e.
C:\DOS\KEYB UK,,C:\DOS\KEYBOARD.SYS.
```

The command prompt is fully user specifiable, and exotic and fancy prompts are possible (just look at the CMH prompt). If no prompt is specified, no directory information appears, and so the commonest prompt line reads:
PROMPT \$P\$G

Which means show the current directory (\$P) and then a > sign (\$G)

PROMPT can accept a whole range of ANSI commands.

BATCH FILES ON DISK

The default CONFIG.SYS on the disk.

```
DEVICE=\DOS\SETVER.EXE
DEVICE=\DOS\ANSI.SYS
DEVICE=\DOS\RAMDRIVE.SYS 192 128 32
COUNTRY=044,,\DOS\COUNTRY.SYS
DEVICE=\DOS\PRINTER.SYS LPT1=(4201,437,4)
INSTALL=\DOS\NLSFUNC.EXE
DOS=HIGH
FILES=30
BUFFERS=30
LASTDRIVE=Z
STACKS=9,256
SHELL=A:\DOS\COMMAND.COM /E:512 /P
```

Default AUTOEXEC.BAT

```
@ECHO OFF
CTTY NUL
\DOS\MODE CON CP PREP =((437)\DOS\EGA.CPI)
\DOS\MODE CON CP PREP =((437)\DOS\4201.CPI)
\DOS\FASTOPEN C:=64
\DOS\SHARE /L:32
SET TEMP=A:\
SET BILL= YOUR ESTEEMED TUTOR
SET COMSPEC=A:\DOS\COMMAND.COM
PATH A:\;A:\DOS;A:\MENU;A:\UTILS
\DOS\KEYB UK,437,\DOS\KEYBOARD.SYS
CTTY CON
```

```
REM NOTHING.BAT
REM Written by Bill Hayles
REM on 11th June 1993
REM
REM This batch file is perfectly understood by DOS, and so it will be
REM quite happily executed. However, it actually does absolutely
REM nothing as it contains only REM statements!
```

```
REM NOTHING.BAT
REM Written by Bill Hayles
REM on 11th June 1993
REM
This batch file is misunderstood by DOS, as the
programmer missed out some REMs.
However, it does not hang the computer
```

```
REM TRYCTTY.BAT
REM Written by Bill Hayles
REM On 11th June 1993
REM
REM This batch file does nothing, but does it in a dangerous way!
CTTY NUL
ECHO Although you would think this line should be echoed to screen,
ECHO it isn't! It's echoed to NUL, i.e. nowhere
REM If we don't reinstate the con, we'll never regain control
CTTY CON
```

```
REM HANGIT.BAT
REM Written by Bill Hayles
REM On 11th June 1993
REM In a mischievous mood
ECHO The computer has hung! You'll have to three-finger salute!
CTTY NUL
```

This is MENU.TXT which makes use of the simple batch files.

```
+-----+
|                                     |
|      DEMONSTRATION OF SIMPLE BUT USABLE |
|      MENUING SYSTEM USING NOTHING BUT  |
|      BATCH FILES                       |
|                                     |
|      PLEASE TYPE THE NAME OF THE GAME YOU WISH |
|      TO PLAY EXACTLY AS SHOWN           |
|                                     |
|      5BY5                                |
|      AMAZE                               |
|      CONNECT4                            |
|      OTHELLO                             |
|                                     |
+-----+
```

These are the batch files to go with the simple menu method

```
@ECHO OFF
CLS
PROMPT WHAT IS THY CHOICE, O WISE ONE?
TYPE MENU.TXT
```

```
@ECHO OFF
CD \GAMES
5X5
PROMPT $P$G
CD \MENU
MENU
```

```
@ECHO OFF
CD \GAMES
AMAZE
PROMPT $P$G
CD \MENU
MENU
```

```
@ECHO OFF
CD \GAMES
OTHELLO
PROMPT $P$G
CD \MENU
MENU
```

```
@ECHO OFF
CD \GAMES
CONNECT4
PROMPT $P$G
CD \MENU
MENU
```

This is BEMENU.TXT for the single batch file menu method:

```
+-----+
|                                     |
|      DEMONSTRATION OF SIMPLE BUT USABLE      |
|      MENUING SYSTEM USING AN ERRORLEVEL      |
|      PRODUCING UTILITY                      |
|                                     |
|      PLEASE TYPE THE NUMBER OF THE GAME YOU  |
|      TO WISH TO PLAY AS SHOWN              |
|                                     |
|      1 - 5BY5                               |
|      2 - AMAZE                              |
|      3 - CONNECT4                          |
|      4 - OTHELLO                           |
|                                     |
+-----+
```

```
REM BEMENU.BAT
REM THE USE OF THE NORTON BE ASK FACILITY ENABLES US
REM TO WRITE A USEFUL MENU IN ONE BATCH FILE
@ECHO OFF
:START
CLS
TYPE BEMENU.TXT
\UTILS\BE ASK " ",1234E
IF ERRORLEVEL 4 GOTO PROGRAM4
IF ERRORLEVEL 3 GOTO PROGRAM3
IF ERRORLEVEL 2 GOTO PROGRAM2
CD \GAMES
5X5
CD \MENU
GOTO START
:PROGRAM2
CD \GAMES
AMAZE
CD \MENU
GOTO START
:PROGRAM3
CD \GAMES
CONNECT4
CD \MENU
GOTO START
:PROGRAM4
CD \GAMES
OTHELLO
CD \MENU
GOTO START
```