Command Line How-To
Joys of the Command Line

Introduction

A command line window is simply a window on your computer in which you can type commands that do things. The program which shows you a command line window is called *Terminal* under Mac® OS® X, and *Command Prompt* under Windows®.

Command line windows seem very technical and alarming to those who have never used them. However, they are very useful, and anyone who has to deal with a large collection of files quickly becomes addicted to them because of the ease with which they let you do things to many files with a single command line. Command line windows are even very easy to use. The key to learning how to use them usefully is to:

1) remember that you only need to know a few commands,

2) accept that the command names and arguments are entirely arbitrary and inconsistent, and

3) find someone who can tell you exactly what to type for specific operations — the on-line help is usually usable only for programmers and you can lose a lot of time trying to figure out how to do things.

This guide will first tell you how to fix your command window to make it useful, and then tell how to use it.

Configuring your command line window

On both Mac and Windows, the command line windows are much easier to use if you make some changes in their default properties. You need to:

– increase the window size

– increase the command history and widow buffer size

– change the prompt

– on Windows, make it easier to cut-and-paste.

The window size here means how many lines of text the window shows, and how many characters you can type on each line. For reasons of ancient history, default values are 24 lines by 72 characters, which is pretty useless.

Command history is the number of lines that the window remembers. You can recall previous commands by pressing the up-arrow key, or by typing a command that shows all previous commands.

Buffer size is how much text the window can hold. The buffer holds not only the commands you typed, but also all the text that the commands produced. Some commands can produce thousands of lines of output, so you want your buffer to be as a large as possible, on the order of 10,000 lines. By default, the buffer is only as large as the window size, so you lose any text that scrolls off-screen. If the buffer is larger than the window size, then you get a scroll bar on the right side, and can scroll back and see previous command lines and their output.

The prompt is the stuff that gets written on the left side of the bottom line, to prompt you to enter a command. The default prompt usually contains the entire current directory path. If you change to a directory more than a few levels from the top, this will occupy most of the line. It is wise to change this to contain just the current directory name.

Configuring Windows Command Prompt

Open a *Command Prompt* window by navigating to this program name from your *Start* menu. It is usually under *Programs* > *Accessories* > *System Tools*. (Alternatively, you can click in *Run...* and type `cmd`).

Right click on the title-bar of the window, and choose *Properties*.

On the *Options* tab, increase *Command History* buffer size to 200. More than a few hundred isn't useful, as the list gets too long to scan easily.

Still on the *Options* tab, make sure that both *QuickEdit Mode* and *Insert Mode* are selected. With these selected, you can drag and drop a file or folder icon onto the command window, and it will paste in the entire path at the insertion point — very important, as most commands work on file paths, and you really don't want to have to type in long file paths. Also cut and paste becomes practical. If you right-click, anything you have cut or copied in any other window gets pasted into the window. This means you can edit a list of command lines in a word processor, and then paste the lot of them into the command window to execute them. Also, you can copy from the command window by dragging with the left mouse button down to select, and then press the "Enter" key to copy. This allows you to copy part of a command from a previous line to the new line.

On the *Layout* tab, make the Window size at least 80 characters wide and 30 lines high. Experiment with what fits on your screen — I prefer to have them occupy about half of my screen's width and height. The screen buffer height value should be as large as possible. Under Windows XP, this is currently 9999. Make sure the screen buffer width is the same as the window size.

Click "OK" at the bottom of the *Properties* dialog. When prompted whether to modify the current window only, or the shortcut that started this window, choose the latter. This will preserve the changes for the future.

Changing the prompt is a little more technical, but still not hard. In the *Control Panel*, open the *System* control panel. Choose the *Advanced* tab. Click on the *Environment Variables* button at the bottom of the tab. Scroll up and down in the list *User variables for <user name>*. If you see the variable name PROMPT, select it and click on the "Edit" button. If not, click on the "New" button. In the field *Variable name* enter `PROMPT`, and in the value field, enter `$G`. This changes the prompt to be just a greater-than sign (>). To see other text strings you can add, enter the command `help prompt` in the command window.

Click "OK" on the edit dialog, then "OK" on the environment variable dialog, and finally "OK" on the *System* control panel window. If you just close any of these dialogs, the changes are not applied.

Configuring Mac OS X Terminal

Open a *Terminal* window by navigating to this program name from your *Applications* directory. It is usually under *Applications* > *Utilities*.

Drag the *Terminal* application program's icon onto the Dock bar — this is a program you will run a lot, and you want easy access to it.

Start a *Terminal* window by double-clicking on the icon. Under the Terminal menu option (upper left corner of the screen), choose "Window settings…".

Choose "Window" from the drop-down menu, and set the Columns to about 100, and the Rows to about 30. Experiment with what fits on your screen — I prefer to have them occupy about half of my screen's width and height.

Choose "Buffer" from the drop-down menu, and set the scrollback number of lines to Unlimited. Make sure that all the options under "Scrollback" are checked.

Now, at the bottom of the dialog, click on the button "Use Settings as Defaults". This makes your changes apply to new command windows.

To change the prompt, you need to edit an obscure file that sets parameters for the *Terminal* program when it starts up. Unfortunately, the way the *Terminal* program works depends on which Unix command line program it is setup to use. There are several, and each one uses a different name for its startup file, and requires a different line of text added to change the prompt. To see which one you have, under the Terminal menu option, choose "Preferences…", and note the line of text in the top text field.

Program name	Startup file name	Line to add
`csh`	`.cshrc`	`set prompt="%c$ "`
`tcsh`	`.cshrc`	`set prompt="%c$ "`
`sh`	`.profile`	`PS1='\W $ '`
`zsh`	`.profile`	`PS1='\W $ '`
`bash`	`.bash_profile`	`PS1='\W $ '`

Notes: the quotes need to be included, and must be either both single or both double quotes. For `sh`, `zsh`, and `bash`, there must be no spaces on either side of the equals sign. You can use any character string you want for the final prompt characters instead of `"$ "`. I like to have a space at the end of the prompt.

Look in your home directory. Unfortunately, files that begin with a period are hidden in the *Finder*. To make them visible, open a *Terminal* window and type the following command:

```
defaults write com.apple.finder AppleShowAllFiles TRUE
```

Close all *Finder* windows, switch to another program, and then switch back to the *Finder*, and you will be able to see the hidden files. To hide the files once you are done, enter the command line:

```
defaults write com.apple.finder AppleShowAllFiles FALSE
```

If there is no file with the name you need, use a text editor to create it. Add the appropriate line (right column), and save it in your home directory. Close the *Terminal* window and open a new one. Now the prompt should just show your current directory and a dollar sign, rather than the full path to the current directory.

Essential Command line Info

Commands which operate on files require that you specify the directory path of the file. Example:

```
open /Users/rroberts/.bash_profile      # Mac
start C:\adobe\FDK\FDKReleaseNotes.txt  # Windows
autohint -a MyFont.pfa                  # any system
```

The easiest way to get a file path into the command line is to drag its icon from a *Finder/Explorer* window onto the *Terminal/Command Prompt* window. When you do this, the absolute path — the complete path from the computer's root directory — is copied. However, a command window always has a "current" directory. If the file is in the current directory, then you only need to type the file's name, as in the third example above. To see the absolute path of the current directory, type the following command:

```
pwd   # Mac
dir   # Windows (this also shows a list of files in the current directory)
```

To change the current directory, type `cd new_dir_path` on both Mac and Windows. Example:

```
cd /Users/rroberts/Documents   # Mac path
cd C:\adobe\FDK                # Windows path
```

If you leave off the initial slash (`/`), then the path is assumed to be relative to the current directory. If the current directory is `/Users/rroberts`, and you type the path `Documents/ToDo.txt`, then the absolute path is assumed to be `/Users/rroberts/Documents/ToDo.txt`.

Some commands produce a lot of text output, so much that it would be more convenient to look at the output in a text editor with good search functions. To send the output of a command to a file, add a greater-than sign (`>`) followed by a file path, to the command line. For example:

```
autohint -a MyFont.pfa
```

is likely to produce several hundred lines of output. To browse this more easily, enter:

```
autohint -a MyFont.pfa  >  MyFont_autohint.txt
```

This will "re-direct" the output of the `autohint` command into the file `MyFont_autohint.txt`. You can then open this file in your favorite text editor, and search for interesting notes.

Favorite AFDKO (Adobe® Font Development Kit for OpenType®) commands

Run **compareFamily** QA tool on all the fonts in the current directory, and send the output to the file `cf_output.txt`:

```
compareFamily -rn -rm -rp > cf_output.txt
```

Note: All the fonts in the current directory should belong to the same family; compareFamily can handle fonts from multiple font families found in the current directory, but the output file is easier to read if all fonts belong to the same family. Also, it may issue incorrect error messages if the directory contains fonts from different families.

Get the in-line help for compareFamily:

```
compareFamily -h
```

Run **checkOutlines** QA tool on the font `MinionPro-Bold.otf` present in the `Bold` subdirectory of the current directory:

```
checkOutlines Bold/MinionPro-Bold.otf
```

Autohint only unhinted glyphs in a font: (This allows you to manually hint some glyphs in FontLab without overwriting that work when using the autohint program)

```
autohint font.pfa
```

Autohint all glyphs in a font: (This will remove any hints that existed before)

```
autohint -a font.pfa
```

Build and OpenType CFF font in release mode, assuming that all the input files (`font.pfa`, `features`, `fontinfo`, `FontMenuNameDB` and `GlyphOrderAndAliasDB`) have default names, and default locations relative to the current directory: (The resulting OpenType font file, will be named according to the font's PostScript® name)

```
makeotf -r
```

Build and OpenType CFF font in release mode (assuming that all the input files have default names and locations relative to the current directory), and write the OpenType font file with the name `test.otf`:

```
makeotf -r -o test.otf
```

Display a crude representation, at `24` points per em, of the glyphs `A` and `dollar.taboldstyle` from the font `MinionPro-Regular.otf`: (This command does not work with fonts containing TrueType® outlines)

```
tx -bc -z 24 -g A,dollar.taboldstyle MinionPro-Regular.otf
```

Get a PDF file (`MinionPro-Regular_glyphset.pdf`) displaying all the glyphs in the font:

```
tx -pdf -1 MinionPro-Regular.otf > MinionPro-Regular_glyphset.pdf
```

Get the glyph name, encoding value, advance width and the glyph's bounding box (left,bottom,right,top), for all glyphs in the font:

```
tx -mtx MinionPro-Regular.otf
```

Get all the switches that can be used with the `-mtx` option of the **tx** program:

```
tx -mtx -h
```

Dump the `name` table of an OpenType font, in a fairly readable way:

```
spot -t name=3 MinionPro-Regular.otf
```

Dump the `GSUB` table of a font (in AFDKO's feature-file syntax), and re-direct the output to a text file (`gsub.txt`):

```
spot -t GSUB=7 MinionPro-Regular.otf > gsub.txt
```

Dump the `GPOS` table of a font, and re-direct the output to a text file (`gpos.txt`):

```
spot -t GPOS=7 MinionPro-Regular.otf > gpos.txt
```

Recommendations for text editing applications

Large log files and the urge to build lists of command lines require a text editor with excellent search-and-replace abilities. For example, it is useful to be able to search the *compareFamily* report for all lines containing the word "Error", and show them all in a window. For Windows, the clear low-cost leader is UltraEdit®; for Mac, an excellent choice is BBedit®. Get one, and read the program help about *Regular Expressions* (sometimes referenced as *grep* or *regexp*). Regular Expressions is a programmer term for wide-spread standard for wild cards on steroids. These let you do things like "find all lines beginning with 'error', containing MinionPro-Bold, not containing 'width' after MinionPro-Bold, and not containing 'ligature'". The very complex forms are mind-bending, but you can do very useful things with the simple wild-cards. MS Word, WordPad and TextEdit are woefully insufficient for finding useful information and for building command line batch files.

A real example of the power of command line tools plus good editors with regular expression support is the following story on Mac OS X:

1) Discover that for a library of hundreds of fonts, the copyright string needs to have ", 2007" added to the list of copyright years.

In the directory structure used by Adobe, this would be specified in the "features.tables" file, which can be either at the face directory level or, if it is the same for all faces, at the family directory level.

2) Open all the files with a single command line:

```
# change the current directory to the library root directory
cd /adobe/AdobeTypeLibrary/fonts/
```

```
# open all the "features.tables" files in the all sub-directories, up to three levels down
bbedit */features.tables */*/features.tables */*/*/features.tables
```

3) The BBEdit text editor is now open, with several hundred windows. Open the Find window, and do a multi-file Grep replace of all occurrences of `(Copyright\s*((19..|20..),*\s*)+)` with `\1, 2007`

4) Quit BBEdit.

This took five minutes, instead of several hours of opening each of several hundred files and making a minor editing change. Most of the time was spent iterating steps 2 through 4 until I got the search-and-replace command right.