# 3) Why command-line tools?

I use makefiles and simple tools because they give me insight into how the problem is being solved. It is hard to create a better solution if I let an integrated development environment (IDE) hide the details from me. While the IDE may give me a result faster, it may not be the best solution to the problem. I can better control each part of the compilation and linking processes by using a makefile. It is easy to add a resource rule to the makefile when you create a graphic solution. Adding a required library to a makefile is also simple, which is not always the case when using an IDE.

Simple command-line tools allow me to see exactly what is going on. Each simple command is very powerful because it does only one obvious thing. An IDE follows the same paths as the command-line tools but it hides much of its functionality under cover. Ease of use does not mean you learn more. I can easily switch from compiling for a Linux computer to a Windows computer by using the makefile as my construction recipe. An IDE prevents this in the worst case, while making it more difficult in most cases. I have more control over what I create by using the command-line tools. I learn more, and more rapidly, because everything is right in front of me, easy to read.

By learning command-line tools you can use any computer, anywhere, to work on your ideas. As long as you have an editor and a compiler, with access to the file system, you can create applications. As long as you have permission to modify a computer, you can add the missing tools quickly and for free. The simplest, and the least intrusive, modification is to install MinGW on a Windows box, or gcc on your Linux box. You can expand the MinGW environment a great deal by adding more tools, but the base set is all you need for these lessons. Installation adds a folder with the tools and libraries, and a modification to the environment variable PATH, so the MinGW tools are available for use from any command prompt.

If you cannot modify the computer you're working on hopefully, you have access to its file system. That way you can use the native editing tools of the OS (Notepad on Windows for instance, or gedit on Ubuntu for another) to write and edit code, comments, or notes about programming ideas. You won't be able to compile your code, but that actually helps by making you read your code like prose. Adding more comments is always a good idea. Fixing any formatting problems is a good way to check your code line by line for logic errors. Not having access to a compiler makes you think about your code more. Be your own compiler. Imagine how variable tables are built, or string data is accessed, and rewrite your code to make things more efficient. Are your variable and function names descriptive and accurate? Do they help your code read correctly as a sentence or a story? Do they help you remember what is happening at any given line?

If you cannot access a computer but have paper and pencil you can also create code. Write out your ideas in pseudo code, describing what you want done, and how you propose to do it. Then start filling out each module with more pseudo code. Once you get your idea roughed, out start writing modules in part pseudo code, part real code. Repeat until you have enough code to start typing.

I like the idea of simple tools. They don't take much space. They are available either immediately or within a minute or two from the web. Most of all they don't own me in any way (free as in unencumbered). I am not forced to use the same editor each time I write or change a file. In fact, I don't even need to use the same computer. My code is just a file which lives on a file system. My editor can be as simple as Notepad, or nano, if I really don't care much about formatting. But, they are available everywhere, ready to start typing as soon as the computer is running. No waiting for a bloated IDE to load just to write a few lines of code. IDEs tend to own you. They force you to create folders and file systems just for a short bit of code. When I use a simple editor I can keep my code anywhere I want. This forces me to know where I put things and why I put them where I did. It forces me to be organized, but I am also free to move everything with one folder copy command.

Once you have learned how to use command-line tools moving to an IDE will feel strange. Everything seems to be done for you, but at a price. One button click to compile your code (but you did have to spend an hour to get the flags set correctly the first time). Everything is encapsulated in the IDE's structured way of things and your working application is buried levels deep in the cruft. At this point the IDE owns you. You have paid the price of ease.

Simple tools allow you freedom. They allow you to create your own makefiles and file structures to your own needs, following your own thinking. I am free to work, able to create code in a fluid manner while maintaining my form of order instead of constantly learning tools and tweaking environments. The worst part of an IDE is losing my files, never quite knowing what the compiler has been asked to include in the final application. My hand-crafted makefiles use file structure habits I've learned and are easy to read. Make has a lot of cool macros which I don't use very often. I would rather type everything out longhand than not understand who is doing what to whom when I type **make**. It is all about freedom and control.

**Unix tools I use on a regular basis:**

```
ls, ls -l, ls -lt, rm, cp, df -h, rsync, cat, >, |, *, mkdir
```

How do I navigate to my lesson folders?

```
cd /srv/samba/share/*ion
```

Here I used the asterisk character to represent BD_2nd_Edition the name of the folder. By finding the unique portions of each folder's name I was able to use the wild card character * to save a lot of typing. When you are typing a long folder string it really helps if you can use the * key.