

# A Look at Command Line Utilities

by Danny R. Faught

When I think about which tool I use most often, what comes to mind is not one tool, but a whole collection of small tools, often called “utilities,” that have one thing in common—an old-fashioned command line user interface. Because I switch between Mac OS, Microsoft Windows, Linux, and Unix variants, I can count on these basic tools being there. Keep in mind that the command line is useful for testing any kind of application interface. I usually test GUI or Web applications, and I’m constantly using the command line to facilitate my testing.

## The Shell

The core of a command line interface is the shell—the program that launches other programs and manages their input and output. Common shells include bash and tcsh, which evolved from Unix, and the MS-DOS shell, a more limited shell available on Microsoft Windows. Shells include some niceties like keeping a history of previous commands that you’ve entered as well as some rudimentary programming support like variables, conditional logic, and looping. However, the bulk of the work is done by external programs, which we’ll talk about later.

A text-only terminal running a shell used to be the primary interface for a computer, but now shells are most often run in terminal emulators running as a window in a graphical user interface. I usually have several terminal emulator windows running on each computer I’m using. Sometimes I’ll remotely access a shell on another computer using ssh or telnet, which uses much less network bandwidth than controlling a GUI remotely.

## Running a Shell

On most operating systems I use, shells and a nice set of Unix-style programs are installed by default. On Mac OS X, I run the Terminal application to get to a command line. On Linux and Unix, there are several terminal emulators

```

dfaught@qa:~ -- bash -- 72x18
~ $ uname -srp
Darwin 7.9.0 powerpc
~ $ ssh qa
[dfaught@qa dfaught]$ uname -srp
Linux 2.4.21-20.ELsmp i686
[dfaught@qa dfaught]$ head -3 /usr/local/tomcat/logs/catalina.out
Dec 22, 2005 5:08:41 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 1028 ms
Dec 22, 2005 5:08:41 PM org.apache.catalina.core.StandardService start
[dfaught@qa dfaught]$ grep XML !$ | sort | uniq
INFO: XML validation disabled
[dfaught@qa dfaught]$ grep XML !!:2 | wc -l
53
[dfaught@qa dfaught]$ logout

```

A terminal emulator on Mac OSX logging in to a Linux Web Server

available, including xterm and rxvt, which are launched in a variety of ways, depending on which desktop environment you’re using.

Microsoft Windows systems are an extra challenge. For any Windows system that I’m going to use for more than a few hours, I install the free Cygwin environment (see the StickyNotes for a link). Cygwin provides a wide variety of Unix shells and utilities on Windows.

The standard set of Cygwin tools is a rather large download, and I usually need to add a few optional utilities as well. In cases where I need a specific tool quickly, I’ll download a native Windows build of the tool from the GnuWin32 project (see the StickyNotes for a link) and run it from an MS-DOS shell.

Though MS-DOS is rather simplistic, you can familiarize yourself with the MS-DOS commands that you have at your disposal on any Windows system without having to install anything extra. I’m going to focus on Unix-style shells and utilities here because they’re so much more powerful than MS-DOS and they’re readily available to install on Windows.

## Plumbing

The Unix philosophy is that programs should be simple, doing one thing and

doing it well. Options usually go on the command line, making the utilities as easy to run from a shell script as when you run them manually from the shell. If a utility needs input, it usually reads from the “standard input” stream, which you can type manually, or use the shell to redirect the input from a file instead. Similarly, output usually goes to the “standard output,” which you’ll see in your terminal by default, or you can redirect it to a file.

How can simple tools solve a complex problem, though? The shell’s plumbing can make amazing things happen. Here’s an example: Let’s say you’re testing a Windows application that writes debugging information to a plain text log file named “c:\temp\debug.log”. You can view the most recent lines in the log by using the tail utility. And you can view new lines as they’re written to the file if you use tail’s “-f” option:

```
tail -f "c:\temp\debug.log"
```

This runs the tail utility, giving it the “-f” option followed by the filename. I use tail frequently, watching the debug output while I test an application. But sometimes the output is so verbose that

it's hard to spot the information that's most interesting. So we can use some plumbing and the grep utility to help:

```
tail -f "c:\temp\debug.log" |
  grep -i ERROR
```

This uses the same tail command, but instead of sending the output directly to my terminal, I used the "|" pipe symbol to tell the shell to connect the standard output of the tail program to the standard input of the grep program. Grep's output then goes to my terminal. The grep utility is used to search text for a pattern that we specify. In this case, the pattern is "ERROR", and I added the "-i" option to do a case-insensitive search, so it will find "error", "Error", etc. I filter the debug information on the fly, only showing the lines that match my pattern. If I prefer to filter out a particular pattern and show all the rest, I can also use grep's "-v" option.

What if the application writes debugging information to its standard output

rather than to a log file? Sometimes when I run a GUI application from a shell, I'm surprised to find that it's printing useful information in the shell that I otherwise would not have seen. When I launch an application through the GUI, Mac OS X logs the output in a system log that I don't often think to check, and as far as I know, Windows throws the output away. Several times I found that a GUI application was aborting right after I started it, and when I ran the application from a shell, I saw errors that explained what was going wrong.

Let's extend our example, assuming we have a program that is writing a lot of debugging information to standard output. Sometimes I'll encounter this situation with one of my test tools that's configured to be verbose with its logging. One such tool that I wrote is a GUI test tool named "monkey." Below is a three-stage pipe I use with this tool. This is a more advanced pipe example, so don't worry if it doesn't make sense without some background reading on

how shells work.

```
monkey | tee monkey.log |
  grep -i ERROR
```

First, I send monkey's output to the tee utility. Here, tee writes all of monkey's output to a file named monkey.log so I can refer to it later. Tee also copies its input to its standard output. I pipe this extra copy of the output to grep like I did before, so I can watch my terminal for any problems that crop up without slogging through all the verbose information about where the monkey is clicking on the screen. So the file gets all of the information without the filtering, and what I see on my terminal is filtered. If I reverse the order of the tee and grep commands, then the contents of monkey.log are also filtered.

### Learning the Command Line

Many people have said "Unix is a great place to live, but I wouldn't want to visit there." Unix utilities are very powerful once you learn how to use them, but casual visitors may get frustrated by the steep learning curve. I learned Unix through a combination of books, online manual pages, and looking over people's shoulders and asking, "How'd you do that?" I highly recommend you find a shoulder to look over. I've included in the StickyNotes some references that may help you learn how to live with Unix. **{end}**

*Danny R. Faught is proprietor of Tejas Software Consulting, focusing on practical approaches to exploratory testing and test automation and a regular contributor to Better Software magazine and StickyMinds.com. See [tejasconsulting.com](http://tejasconsulting.com) for more information or contact him at [faught@tejasconsulting.com](mailto:faught@tejasconsulting.com). Danny thanks Grig Gheorghiu and Walter Kruse for their input on this article.*

### Sticky Notes

For more on the following topics, go to [www.StickyMinds.com/bettersoftware](http://www.StickyMinds.com/bettersoftware)

- Link to Cygwin
- Link to GnuWin32
- References

## Can Agile Development Really Scale Beyond a Small Team?

Rally has helped thousands of developers, testers, analysts and their managers improve their **responsiveness** and velocity, gain real-time project **visibility**, and increase team **collaboration** for distributed, multi-team projects.



### Scale Software Agility from Single Team Projects to Multi-Team Programs

Agile Software Development Management On Demand

- Program and Project Management
- Agile Requirements Management
- Test Management and Defect Tracking

Test drive the **New Rally Team Edition**  
at [www.rallydev.com/bsm](http://www.rallydev.com/bsm)



scaling software agility



Visit our  
Booth!

[rallydev.com/bsm](http://rallydev.com/bsm)