iSeries to UNIX: First Contact

In this three-part series, Thibault Dambrine describes the basics of the UNIX Operating System and first impressions, from the perspective of an iSeries programmer.

- *Part 1 describes the basics of UNIX, how it came to be and how the UNIX architecture differs from that of the iSeries.*
- *Part 2 deals with some of the basic UNIX commands, directories, security and the scheduling system.*
- *Part 3 is all about shell scripting, the UNIX equivalent of the iSeries' CL language.*

Thibault Dambrine

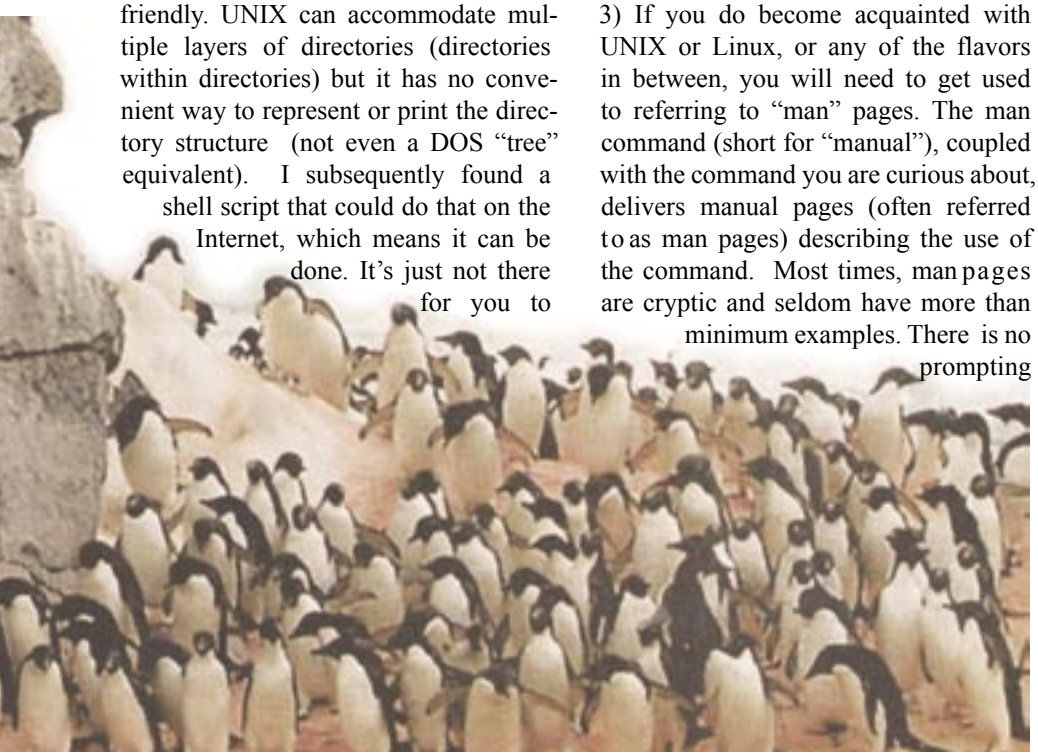
(Part II of III)

UNIX First Impressions

By Thibault Dambrine

In any situation where you try a new car, visit a new house, you get a few "first impressions" that will likely remain in your mind for a while. In the next few paragraphs, I will describe my first UNIX impressions. My aim is to convey what I saw, as a typical iSeries programmer, when first crossing the line. Here are 10 items that struck me in my first UNIX experience:

1) Upon first glance, my number one impression was that UNIX could be probably well described as "MS DOS on steroids". The command line behaves the same way and it seems to be just as friendly. UNIX can accommodate multiple layers of directories (directories within directories) but it has no convenient way to represent or print the directory structure (not even a DOS "tree" equivalent). I subsequently found a shell script that could do that on the Internet, which means it can be done. It's just not there for you to start with. One other common point with DOS, the Shell Script language (Korn Shell, Bourne Shell etc..) is interpreted, like a .bat type of DOS file.



2) UNIX commands have evolved from many sources and many authors. They do NOT follow any type of naming convention. This, for the OS/400 programmer, is something to get used to. It's hard to compare commands like "CRTCLPGM" – named after "**CR**ea**T**e **C**ontrol **L**anguage **P**ro**G**ra**M**" with "awk" named after "**A**ho, **W**einberg, and **K**ernighan" – the three authors of the command. "awk", by the way, is a text manipulation command.

3) If you do become acquainted with UNIX or Linux, or any of the flavors in between, you will need to get used to referring to "man" pages. The man command (short for "manual"), coupled with the command you are curious about, delivers manual pages (often referred to as man pages) describing the use of the command. Most times, man pages are cryptic and seldom have more than minimum examples. There is no prompting with the F4 key or any equivalent to OS/400 command prompting in UNIX. In a strange twist of fate, QSHELL, the UNIX style shell implemented on OS/400 has no F4 prompt key support and no "man" command either. Again, there will be more on this topic later in the article.

4) Speaking of no iSeries-style F4 prompting, UNIX usually executes, without warning, exactly what you tell it to do. Expect no warnings if you are about to do something completely silly. Case in point: If you log-on as root, the UNIX equivalent to the OS/400 QSECOFR profile, you can comfortably type the following instruction: `rm -rf /*` and hit enter.

With this order, using that profile, UNIX will proceed to wipe out your entire system, including the operating system without even as much as a burp or a thank you. For this reason, to protect yourself against dangerous experiments, it is best to use the root user profile sparingly. This of course, is one man's opinion on command prompting. There are developers who find the iSeries operating system too talkative for its own good, and that less verbose is better.

5) vi is the most well known editor on UNIX. If you know vi, you can be sure you know at least one editor on just about any Unix machine you will ever get to work on, save your trusty iSeries, who has chosen not to implement it in QSHELL. Pronounced "vee-eye", vi is short for **v**isual **i**nterface.

Working with three modes of operation: "command", "insert" and "line editor", vi has over 100 commands. Entire books have been written about vi. My experience is that it is rather confusing until you figure out which mode you are in. This is the type of editor you have to use regularly to get proficient. UNIX pros swear by it. They often say it is the most powerful text-based editor available. If you think there is power in simplicity, you may be tempted to argue differently. My number one vi tip: stay away from the arrow keys on the right side of the keyboard - they won't move your cursor!

6) Object authority or "file permission" in UNIX lingo, is something to both be aware of and get used to. I will go into some detail here because this is a key concept. Understanding how permissions work in UNIX is a necessity for every-day operations and it also gives a good indication of the style in which UNIX works.

Octal Value	Binary Value	Permission	Directory Listing
1	001	no read, no write, execute	--x
2	010	no read, write, no execute	-w-
3	011	no read, write, execute	-wx
4	100	read, no write, no execute	r--
5	101	read, no write, execute	r-x
6	110	read, write, no execute	rw-
7	111	read, write, execute	rwx

Table 1

Each file has 3 sets of "read/write/execute" flags attached to it, one for the owner, one for the group and one for the public. These 3 flags can be set to 0 or 1. To handle all this in the most economical way, each of the three authority groups are represented by the binary value of an octal number. This Octal number, (0 to 7) translated into binary, gives a combination of 3 binary digits.

Clear as mud? You need to understand that clearly when you work with UNIX, as this is one concept you will deal with every day. If for example, you write a small test shell script to see what you can do, you will never be able to execute it unless you change the permissions (using the obviously named `chmod` command) to give yourself permission to execute the file.

A file, in UNIX can be anything from a flat file to a UNIX script. A UNIX script with an "x" in the "execute" permission flag position is deemed executable, even if not compiled (shell script is interpreted). Each file has 3 types of permissions (read, write or execute), for each of the three categories of users (individual users, groups of users or all users).

The 0 to 7 values are actually used as shortcuts, referring to their binary values. For example, the binary value of Octal 5 is "101". This translates into a permission value of "r-x" or "read, no write, execute". Octal 6 translates into binary "110" and thus the permission value of "rw-" or "read, write, no execute". **Table 1** tells the whole story:

If a file for example has permission setting 777, it really has permission setting where the owner has rwx (read/write/execute), the group has rwx (read/write/execute) and the public has rwx (read/write/execute) permission. The permission flags look like rwxrwxrwx, listed at the right side of the file in a directory listing.

If a file has permission setting 764, the individual has rwx (read/write/execute), the group has rw- (read/write/no execute), and the public has r-- (read/no write/no execute). The permission flags look like rwxrw-r--, listed at the right side of the file in a directory listing.
You can see the permissions on the files in a directory by doing a `ls -l` command.

There is more to read on UNIX permissions, but this is the base functionality. Coming from the iSeries environment, this method of determining authority or permissions, looks like a step backwards. It is however widely used and accepted by UNIX lovers the world over.
There is no equivalent in UNIX to the notion of "object security" in the OS/400 sense. This may help you to understand the somewhat simple UNIX permission system.

Directory	Description
/	Root directory of the filesystem.
/bin/	User utilities fundamental to both single-user and multi-user environments.
/boot/	Programs and configuration files used during operating system bootstrap.
/boot/defaults/	Default bootstrapping configuration files.
/dev/	Device nodes.
/etc/	System configuration files and scripts.
/etc/defaults/	Default system configuration files.
/etc/mail/	Configuration files for mail transport agents such as sendmail.
/etc/namedb/	Named configuration files.
/etc/periodic/	Scripts that are run daily, weekly, and monthly, via cron.
/etc/ppp/	ppp (point-to-point) configuration .
/mnt/	Empty directory commonly used by system administrators as a temporary mount point.
/proc/	Process file system
/root/	Home directory for the root account.
/sbin/	System programs and administration utilities fundamental to both single-user and multi-user environments.
/stand/	Programs used in a standalone environment.
/tmp/	Temporary files, usually a memory-based filesystem (the contents of /tmp are usually NOT preserved across a system reboot).
/usr/	The majority of user utilities and applications.
/var/	Multi-purpose log, temporary, transient, and spool files.

Table 2

7) One of the things you may have heard already about UNIX is that there is more than one standard for this operating system. You heard the truth. The main flavors of UNIX currently in the market place are Berkeley UNIX, AT&T Systems V Release 4 , SunOS/Solaris, XENIX, SCO and Linux.

The closest thing to a governing body that says, "This is UNIX" is the POSIX standard. It is now an accepted fact that to be commercially viable, a UNIX system, of any flavor, has to be POSIX compliant. That said, as long as this minimum is met, any company is free to invent or come up with a newer, better and stronger version. This is good and bad at the same time, for obvious reasons. It is good because there is freedom to innovate and change, it is bad because there is no strict standard, and thus incompatibilities do creep in.

Perhaps the closest thing to a standard in UNIX available right now is the Linux kernel, which is standard across all distributions (Red Hat, SuSe etc.). It is also evolving faster than all other UNIX flavors and provides a lot more user friendly utilities than the traditional UNIX vendors.

8) You may have heard of the term UNIX "shell" before. "Shell" is a UNIX term for the interface between the user and the operating system. The shell is the layer of programming that interprets and executes the commands a user enters. As the outer layer of an operating system, a shell can be contrasted with the kernel, the operating system's inner most layer or core of services.

There are several UNIX shells. The main shells are Korn shell, C-shell and Bourne shell. Why so many?

The shell, in UNIX, is more or less equivalent to the iSeries CL language. Just as there are many UNIX operating system manufacturers, the UNIX Shell has also evolved over time. S.R. Bourne wrote the original UNIX shell around 1975. His version, one of the most commonly available, is the "Bourne shell". Bill Joy, from the University of California in Berkeley, created the "C shell". David Korn, at AT&T, created the "Korn shell". Linux has something called the "Bourne Again shell", also known as "bash". Consistently, each new shell creator (or creators) have tried to build upon established bases to create a better shell. Note that the Linux bash (Bourne Again shell) is free, as opposed to the other shells.

9) Despite differences between UNIX releases, some things are similar. Upon delivery, most UNIX systems have a number of standard directories, which contain well-known utilities or contain operating system programs that will perform certain functions or services. **Table 2** gives a quick idea of what these are. Note that the slash (/) is the symbol for the root directory. If you do a "cd /", you are doing a "change directory to root". This directory, in UNIX, is the OS/400 equivalent to QSYS, where all libraries ultimately belong. Note also that the example below is a typical map, but not necessarily an exact one for all UNIX flavors. On the iSeries, up until recently, we had only one level of library or director: There was QSYS, and all the libraries within it. This has now changed with the addition of the IFS, the Integrated File System, which allows multiple levels of directories.

10) In the same league as knowing what is going on with the authorities or permissions, there is a function in UNIX called the "cron", or "crontab file", which is equivalent to what we would call the "job scheduler" utility in OS/400. The crontab, you guessed it, is UNIX's version of a task scheduler. At first glance, it is very cryptic, but it can be interpreted with relative ease if you know what it is talking about. Be prepared – in UNIX, "Relatively cryptic but can be interpreted" is practically a way of life.

This is how the entire operating system is built. Many UNIX programmers will tell you that this is where the power is. Like beer, this may be an acquired taste, popular but acquired.

Back to the topic. A crontab file can have any number of commands, although it can only process one command per line. One more thing to remember: after the last command, you must have a blank line or the crontab will not run.

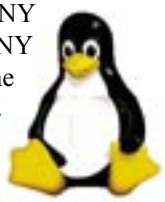
What does a crontab line look like? The way the schedule looks is very cryptic but it's really very simple. There are five fields to the schedule. They are noted in bold below:

MINUTE(0-59) **HOUR**(0-23) **DAY-OF-MONTH**(1-31) **MONTH-OF-YEAR**(1-12) **DAY-OF-WEEK**(0-6) Note: 0 = Sunday. Also note that the ASTERISK (*) stands for a WILDCARD meaning it will match any value.

Example 1:

```
0 * * * * /etc/dothis.abc
```

This means literally "execute the script located at /etc/dothis.abc" whenever the clock is equal to 0 minutes o n ANYDAY, ANY HOUR, ANY DAY-OF-MONTH, ANY DAY-OF-WEEK. So the script is set to run ONCE PER HOUR EXACTLY ON THE HOUR regardless of what day it is or what hour.



Example 2:

```
0 0 * * * /etc/dothat.now
```

This is a little more picky. This crontab runs again whenever the internal clock hits ZERO (0) Minutes, but instead of running once per minute it will only run once per day. Why? Because we also set the HOUR to zero so both the MINUTE and the HOUR must be equal to zero before crontab will execute /etc/dothat.now. So this example runs once per day at midnight, server time.

Example 3:

```
21 14 * * 2 /yellow/brick/road/wizard.exe
```

This example sets a crontab entry to run only on Tuesday at 2:21 PM. Note that this is written in military style time, i.e. 14:21.

In the next issue (Part III), I will explore more details regarding UNIX Shell Scripting, and in particular the remarkable iSeries QSHELL environment.

TUG

Thibault Dambrine *works as an independent contractor in Calgary, Alberta. You can visit the iSeries / AS/400 related website he manages at http://www.tylogix.com or e-mail him directly at thibault.dambrine@tylogix.com.*