Advanced Unix Tools — **Lecture 1**
CS214 Spring 2002 — **Monday, February 18, 2002**

I will assume you are familiar with the 114 material:

- filesystem notions
- permissions
- text editing
- regular expressions
- job control
- I/O redirection
- shell ideas: variables, scripts

Recall that on a Unix system, the shell is the basic interface between the user and the operating system. More specifically, the shell has the following “responsabilities”:

1. read and parse command line,
2. evaluate special characters,
3. setup pipes, redirections, and background processing,
4. find and setup programs for execution.

Alternatively, there are three main uses of the shell:

1. as an interactive interface,
2. as a customization tool (to customize applications via environment variables, for example),
3. as a programming language for scripts.

The history of shells is tied with that of Unix itself. Originally (I’m talking in the early 70s), there were two main “strains” of Unix: one from AT&T, and one from Berkeley. AT&T Unix came with the Bourne shell (or *sh*, due to Stephen Bourne), a minimalistic shell inspired by Algol, and much more aimed at scripting than at easing the interactive experience. At Berkeley, Bill

Joy and others developed the C shell (or *csh*), inspired by the C programming language, and with interactivity in mind: it supported history mechanisms and job control. In 1986, David Korn developed the Korn shell (or *ksh*), an extension of *sh* that included much of the *csh* features, and more beyond. Finally, the Bourne Again shell (or *bash*) was developed, an extension of *sh*, inspired by *ksh*. It sported history, command-line edition, filename completion, aliases, arrays, and many programming features. It remains *sh*-compatible. Alternatively, in the *csh* world, the TC shell (or *tcsh*) is a recent shell extending *csh* and supporting much of the same features as *bash*, albeit with a different syntax.

In this course, we will focus on *bash*, the Bourne Again shell. Why? It is a superset of *sh*, the minimalist shell that is quite popular among sysadmins, is has a slightly easier syntax, it is the default shell on Linux systems, and finally, why not?

When a shell is started, it can be started in one of three modes:

1. As a login shell. This is the instance of the shell that is first started when you login to your Unix account. Typically, it sets the basic environment, such as the terminal. When you exit this shell, you logout.
2. As an interactive shell. This shell gets created when you invoke *bash* from another shell, for instance. This kind of shell has a prompt, and allows you to enter command for it to interpret. Note that the login shell is a special kind of interactive shell.
3. As a noninteractive shell. This is a shell that gets created to execute a script. It never gives you a prompt, and disappears once the script has finished executing.

There are configuration files that are read by a shell when it is started, different configuration files depending on the mode of the shell. A login shell will attempt to read the file *.bash_profile* in your home directory. If it does not find it, it tries to read *.bash_login* and *.profile*, if they exist. Interactive and noninteractive shells will attempt to read *.bashrc* in your home directory. (Since a login shell is also interactive, a login shell will also try to read *.bashrc*). These configuration files are just shell scripts that are automatically executed.