Advanced Unix Tools	**Lecture 2**
CS214 Spring 2004	**Wednesday, February 25, 2004**

Shell expansions

The Unix system (the core system, if you will) can only execute commands when presented in the form of a sequence of tokens. (Each token can be thought of as a string of characters). The first token represent the command to executed, the subsequent tokens are the arguments to the command. Each command is free to interpret its arguments whatever way it wants.

The main job of the shell is translate a string of characters input by the user (or given by a line of a shell script) into a sequence of tokens for Unix to execute. A rule of thumb is that every part of the input string separated by at least a space is a token. However, this can be somewhat controlled. Understanding exactly what happens in this process of translating a string into a sequence of token is central to being able to write (and especially understand) shell scripts. For the sake of discussion, we will call a sequence of characters separated by spaces a word.

The translation process happens in the following order, and I will describe most of them in the remainder of the lecture.

1. brace expansion
2. tilde expansion
3. parameter expansion
4. variable substitution
5. command substitution
6. arithmetic substitution
7. word splitting
8. pathname expansion

Brace expansion is the process of expanding every word containing a brace expression, of the form {*w1*,*w2*,*w2*}, where each *w1*, *w2*, *w3* are words (without any space), into a sequence of words where the brace expression is replaced by *w1*, *w2*, *w3*, respectively. For example, `abc{de,fg,hi}` expands into the sequence of words `abcde abcfg abchi`.

Tilde expansion expands every ~ into the path to your home directory. The form ~*name* expands into the path to the home directory of user *name*.

Variable substitution and parameter expansion substitute the value of variables. Variable substitution replaces every expression `$`*var* by the value of variable *var*. Parameter expansion is a kind of conditional substitution. There are many variations of parameter expansion. The most common one is to replace every expression of the form `${`*var*`:-`*word*`}` either by the value of variable *var* if *var* is set and non-null, or by *word* if *var* does not exist, or is null-valued.

Command substitution replaces every expression of the form `$(`*cmd*`)`, where *cmd* is a command, by the output of the execution of *cmd*. Hence, `$(pwd)` is replaced by the output of `pwd`, that is, the current working directory.

Arithmetic substitution allows you to perform numerical computations in the shell, instead of using a program such as `bc`. An example of an arithmetic substitution is `$(( 3 + 4 ))`, which gets replaced by `7`. We will return to arithmetic expressions in later lectures.

Word splitting is the process of actually splitting the command line into tokens, at the spaces. Hence, if a substitution occuring earlier in the process substitutes a string with spaces, for example, `$FOO` where variable `FOO` has value `some word`, then that value will be split into two tokens at this step.

Finally, pathname expansion is the process of replacing paths containing wildcard characters (such as `*` and `?`) by the sequence of paths that match the pattern. Recall that `*` matches one of more character, `?` matches exactly one character, `[abc]` matches any character between the brackets (here, `a`, `b`, `c`), and `[!abc]` matches any character not in the brackets. (This process is also known as globbing.) Each filename in an expansion is taken as a token. Note that this step happens after word splitting, so that if there are spaces in filenames, each filename is still considered a token.

Sometimes, we want to disable some aspects of this automatic expansion by the shell. By and large, this process is known as quoting. Quoting can take many forms. The simplest form is simply to escape the special characters that have meaning to the shell. For instance, you may want to use the `$` character without having it interpreted as variable, parameter or command substitution. Similarly for the `&`, `;`, `?`, `*`, `[`, `]`, {, } characters. To use such a special character as a literal character, you precede it with a backslash. (This is called escaping a character.) Since a backslash is itself a special character, if you want a literal backslash, you need to escape it as well.

Two alternate forms of quoting exist. The form `'text'` disables any form of expansion between the single quotes, including word splitting. Hence, a line `cmd 'word1 word2'` will split the line into `cmd` and `word1 word2`, where the latter is a single token. The form `"text"` disables expansion between the double quotes, except for variable and command substitution. But again, word splitting does not happen between the double quotes. Hence, `"*$foo"` will expand to `*bar` if variable `foo` has value `bar`. The quotes (single or double) are removed from the token when word splitting happens, and so never appear to what eventually gets invoked.