**Chapter 4**

**Wildcards in UNIX**

**1. How to use UNIX Wildcards**

Many computer operating systems provide ways to select certain files without typing complete filenames. For example, we may wish to remove all files whose names end with "old". Unix allows us to use wildcards (more formally known as metacharacters ) to stand for one or more characters in a filename.

The two basic wildcard characters are ? and \*. The wildcard ? matches any one character. The wildcard \* matches any grouping of zero or more characters. Some examples may help to clarify this. (Remember that Unix is case-sensitive). Assume that your directory contains the following files:

**Chap bite bin**

**bit Chap6 it**

**test.new abc**

**Lit site test.old**

**Big snit bin.old**

**1.1 The ? wildcard**

The command ls will list all the files. The command

**$ ls ?bit**

**Lit bit**

lists only the files Lit and bit. The file snit was not listed because it has two characters before "it". The file it was not listed because it has no characters before "it".

The ? wildcard may be used more than once in a command. For example,

**$ ls ?i?**

**Lit big bin bit**

Finds any files with "i" in the middle, one character before and one character after.

**1.2 The \* wildcard**

The \* wildcard is more general. It matches zero or any number of characters, except that it will not match a period that is the first character of a name.

**$ ls \*t**

**Lit bit it snit**

Using this wildcard finds all the files with "it" as the last two characters of the name (although it would not have found a file called .bit).

We could use this wildcard to remove all files in the directory whose names begin with "test". The command to do this is

**$rm test\***

Be careful when using the \* wildcard, especially with the rm command. If we had mistyped this command by adding a space between test and \*, Unix would look first for a file called test, remove it if found, and then proceed to remove all the files in the directory!

**1.3 Matching a range of characters with [ ]**

The ? wildcard matches any one character. To restrict the matching to a particular character or range of characters, use square brackets [ ] to include a list. For example, to list files ending in "ite", and beginning with only "a", "b", "c", or "d" we would use the command:

**$ ls [abcd]ite**

This would list the file bite, but not the file site. Note that the sequence [ ] matches only one character. If we had a file called delite, the above command would not have matched it.

We can also specify a range of characters using [ ]. For instance, [1-3] will match the digits 1, 2 and 3, while[A-Z] matches all capital letters.

**ls [A-Z]it**

Will find any file ending in "it" and beginning with a capital letter (in this case, the file Lit).

Wildcards can also be combined with [ ] sequences. To list any file beginning with a capital letter, we would use:

**$ ls [A-Z]\***

**Chap1 Chap6 Lit**

**1.4 Matching a string of characters with { }**

The method described in the previous section matches a single character or range of characters. It is also possible to match a particular string by enclosing the string in { } (braces). For example, to list only the files ending in the string "old", we would use

**$ ls \*{old}**

**bin.old test.old**

To list all files ending in either "old" or "new", use

**$ ls \*{old,new}**

**bin.old test.new test.old**

**2. I/O Redirection**

**2.1 Standard File Descriptors**

The Unix environment allows for each process to have access to three standard file descriptors by default. They are

* 0 standard input
* 1 standard output
* 2 standard error

It is the responsibility of the shell when executing a command to provide appropriate file descriptors to the process for each of these standard files. Most Unix tools are developed to take their input from the standard input file and write their output to the standard output file. Error messages that do not make up part of the expected output are usually written to the standard error file.

Unless otherwise specified, the shell will usually pass it's own standard file descriptors down to the process that it executes, allowing the output from any called tools to be included with the output of the script.

Through using I/O redirection, the developer can modify how the shell handles the file descriptors and usually either replace one of the standard interactive file descriptors with a file on disk, or create a pipe to connect the output file descriptor of one process to the input file descriptor of another process.

Redirection can also be used to perform redirection on file descriptors for a group of commands.

**2.2 Basic File Redirection**

* Disk file redirection is done using the < and > characters.
* > redirects the standard output of the command to write to a file
* >> redirects the standard output of the command to append to a file
* < redirects the standard input of the command to read from a file

**Example:**

**$ ls -al>dirlist.txt**

**$ ls -al>> longlist.txt**

**$ cat a>f1**

For the redirections, the standard file description redirection can be modified by placing the file descriptor identifier in front of the redirection symbol.

For example 2> redirects standard error instead of the default standard input. 1> redirects standard output, which is the default setting.

**Example:**

**$ command > output.log 2>error.log**

**2.3 Advanced File Redirection**

>& is used to redirect one file descriptor to another.

**Example:**

**$ command > common.log 2>&1**

This redirects standard output to common.log, and then redirects standard error to the same place as standard output. The order of these redirections is important, if the 2>&1 is placed before the >common.log, then standard error will be redirected to the standard output (console) then standard output will be redirected to common.log.

<< redirects the standard input of the command to read from what is called a "here document". Here documents are convenient ways of placing several lines of text within the script itself, and using them as input to the command. The

<< Characters are followed by a single word that is used to indicate the end of file word for the Here Document. Any word can be used, however there is a common convention of using EOF (unless we need to include that word within your here document).

**Example:**

**$ sort << EOF**

**bravo**

**delta**

**alpha**

**chrlie**

**EOF**

This would be the same as having a text file with the lines "bravo", "delta", "alpha" and "charlie" in it and redirecting it using sort <input.txt but is simpler and cleaner (no problems with accidentally forgetting to include the input.txt file when we distribute your script)