The Linux Programming Environment (I)

File Name Expansion

- Most file system commands accept multiple file or directory names as arguments

```
# list some files
$ ls –l a.txt b.txt c.txt

# copy some files
$ cp a.txt b.txt c.txt backup

# remove some files
$ rm a.txt b.txt c.txt
```

- Typing long lists of file names can be tedious
- To make this easier, the shell supports file name expansion (or "globbing")

File Name Expansion

- * match any string of zero or more characters

```
$ cp *.txt backup
```

- ? match any single character

```
$ cp ?.txt backup
```

- [abc...] match any of the enclosed characters

```
$ cp [abc].txt backup
```

- [!abc...] match anything but the enclosed characters

```
$ cp [!abc].txt backup
```

File Name Expansion

- [a-z] match any character in the range
  ```
  $ cp [a-c].txt backup
  ```
- [!a-z] match any character not in the range
  ```
  $ cp [!a-c].txt backup
  ```
- {str1,str2,...} match any of the enclosed strings
  ```
  $ cp {dog,cat,duck}.txt backup
  ```
- ~ substitute user's home directory
  ```
  $ cp ~/cs240/*.txt backup
  ```
- ~name substitute some other user's home directory
  ```
  $ cp ~george/cs240/*.txt backup
  ```

File Name Expansion

- The shell processes each command-line argument that contains file expansion operators as if it were a pattern
- It automatically replaces the pattern with the names of all files and directories that match the pattern
- If nothing matches the pattern, the argument is not modified at all
- File name expansion works for all commands, not just file system commands

```
# myprog is a program that I wrote
$ myprog ~/[A-Z]*
```

Quoting

- What if we need to pass arguments that contain meta-characters?

```
$ echo * is an asterisk
```

- Quoting disables a meta-character's special meaning and allows it to be used literally
- \ the character following is taken literally

```
$ echo \* is an asterisk
```

- ' everything between ' and ' is taken literally

```
$ echo '~/[A-Z]*'
$ echo 'It\'s time to go'
```

Quoting

- Quotes can also be used to pass command-line arguments that contain whitespace

```
# cd to a dir with spaces in its name
$ cd ’my cs240 files’

# list some files with strange names
$ ls –l ’hw 15.txt’ ’hw 16.txt’
```

Standard Input and Output

- C++ programs can read input from the keyboard. This is called "standard input"
- C++ programs can write output to the screen. This is called "standard output"
- Many programs that take file names as command-line arguments will read their input from standard input if no file name is provided on the command-line

```
# count the number of lines typed,
# hit CTRL-D to end
$ wc -l
```

- Demo - reverse.cpp

Standard I/O Redirection

- The shell lets you redirect a program's standard input so that it comes from a file instead of the keyboard

```
$ myprog < file.input
```

- The shell lets you redirect a program's standard output so that it goes to a file instead of the screen

```
# overwrite the output file
$ myprog > file.output
# append to the output file
$ myprog >> file.output
```

- You can redirect both standard input and standard output at the same time
- `$ myprog < file.input > file.output`
- Demo - reverse.cpp

Pipelines

- The shell lets you set up a pipeline of commands so that the standard output of one command is used as the standard input of another command

```
$ grep 'B *Y *U' file.txt | wc -l
$ cat file.txt | grep 'B *Y *U' | wc -l
```

- This works because programs like wc and grep read their input from standard input if no file name is specified on the command line