COMP 206 – Introduction to Software Systems

Lecture 5 – Final Linux & Shell Ideas

Job Control

- The shell allows you to manage *jobs*
  - place *jobs* in the *background*
  - move a job to the foreground
  - suspend a job
  - kill a job

Background jobs

- If you follow a command line with "&", the shell will run the *job* in the background.
  - you don't need to wait for the job to complete, you can type in a new command right away.
  - you can have a bunch of jobs running at once.
  - you can do all this with a single terminal (window).

```
ls -lR > saved_ls &
```

Listing jobs

- The command *jobs* will list all background jobs:

```
> jobs
[1] Running      ls -lR > saved_ls &
>
```

- The shell assigns a number to each job (this one is job number 1).

Suspending and Killing the Foreground Job

- You can suspend the foreground job by pressing ^Z (Ctrl-Z).
  - Suspend means the job is stopped, but not dead.
  - The job will show up in the **`jobs`** output.

- You can *kill* the forground job by pressing ^C (Ctrl-C).
  - It's gone...

Moving a job back to the foreground

- The **`fg`** command will move a job to the foreground.
  - You give **`fg`** a job number (as reported by the **`jobs`** command) preceeded by a %.

```
> jobs
[1] Stopped                 ls -lR > saved_ls &
> fg %1
ls -lR > saved_ls
```

Important Linux paths

- “/” is the root of the file system. Every other file falls below “/” in the directory tree:
  - E.g., $ ls /
- “~” is the current users home directory
  - E.g., $ ls ~/
- “.” is means right here when it starts a path, and nothing if it occurs within a path (2nd case just a convenience for programming):
  - E.g., $ ls .
  - E.g. $ ls /usr/./bin
- “..” means the parent directory
  - E.g. $ cd ..

Wildcards (metacharacters) for filename abbreviation

- When you type in a command line the shell treats some characters as special.
- These special characters make it easy to specify filenames.
- The shell processes what you give it, using the special characters to replace your command line with one that includes a bunch of file names.

The special character *

- * matches anything.
- If you give the shell * by itself (as a command line argument) the shell will remove the * and replace it with all the filenames in the current directory.
- "**a*b**" matches all files in the current directory that start with **a** and end with **b**.

Understanding *

- The **`echo`** command prints out whatever you give it:

```
> echo hi
hi
```

- Try this:

```
> echo *
```

* and **`ls`**

- Things to try:

```
ls *
ls -al *
ls a*
ls *b
```

Other metacharacters

? Matches any single character

```
ls Test?.doc
```

[abc…] matches any of the enclosed characters

```
ls T[eE][sS][tT].doc
```

[a-z] matches any character in a range

```
ls [a-zA-Z]*
```

[!abc…] matches any character except those listed.

```
ls [!0-9]*
```

Quoting - the problem

- We've already seen that some characters mean something special when typed on the command line: **`* ? []`**
- What if we don't want the shell to treat these as special - we really mean *, not all the files in the current directory:

```
echo here is a star *
```

Quoting - the solution

- To turn off special meaning - surround a string with double quotes:

```
echo here is a star "*"

echo "here is a star"
```

Careful!

- You have to be a little careful. Double quotes around a string turn the string in to a single command line *parameter*.

```
> ls
fee file? foo
> ls "foo fee file?"
ls: foo fee file?: No such file or directory
```

Quoting Exceptions

- Some *special* characters are **not** ignored even if inside double quotes:
- $ (prefix for variable names)
- " the quote character itself
- \ slash is always something special (\n)
  - you can use \$ to mean $ or \" to mean "

```
echo "This is a quote \" "
```

Single quotes

- You can use single quotes just like double quotes.
  - Nothing (except `'`) is treated special.

```
> echo 'This is a quote \" '
This is a quote \"
>
```