Today’s Lecture

- Review Lab 0’s info on the shell
- Discuss pattern matching
- Discuss regular expressions

Unix Architecture (simplified)

Processes (programs)

- Shell
- Is
- Netscape

The Kernel
(Device Drivers, Virtual Memory, Files, etc.)

- Disk
- Printer
- Keyboard

The Unix Shell

- A shell is a program that presents the user with an interpreted programming environment; there are many many shells!
- It provides
  - Variables and built-in commands
  - The ability to execute external commands (e.g. programs)
  - The ability to redirect input and output
  - Shortcuts such as aliases and wildcards
- In this class we are going to be using tcsh
  - Its an extended version of csh (The C Shell); the tcsh author added the “t” after adding features from the TENEX and TOP-10s operating systems to the vanilla C shell
  - The purpose behind csh was to emulate the syntax and operators of the C programming language
Variables

- $set x = ken$
- $echo x$
- $echo $x

- $set y = (bananas apples kiwi)$
  - This creates a 1-based array (first element indexed with 1)
  - An array is separated by spaces (which can cause problems) and surrounded by parentheses
  - What happens if you leave the parentheses out?

- Certain constructs can take advantage of an array
  - $echo $y[2]$
  - apples
  - $foreach fruit ($y)$
    - bananas
    - apples
    - kiwi
  - $set y[2] = oranges$
  - $echo $y$
  - bananas apples kiwi

More on Spaces

- Arrays are often used to iterate over the contents of directories in the file system
- Since the space character is used as a delimiter for arrays, you need to watch out for spaces that appear in file and directory names
  - See example next slide

Example Directory Structure

- Tenure Review/
  - Tenure Talk
  - Tenure Demo

- $set z = ("find Tenure/ Review -type d -print")$
- $echo $z$

Tenure Review Tenure Review/Tenure Talk ...

- $echo $z[1]$
  - Tenure
- $echo $z[2]$
  - Review

- This is not what we want!

How to fix?

- Use the shell “quoting” mechanism
- Already saw one example when I used the string “Tenure/ Review” in the find command
  - The backslash “escapes” the space and allows the two words to be treated as a single directory name
- You will learn more about the quoting mechanism in your labs; In addition, there is a lot of information about the quoting mechanism in your reference text book
**Example Revisited**

- Tenure Review/
  - Tenure Talk
  - Tenure Demo
  
```bash
%set z = ('"find Tenure\ Review -type d -print"')
%echo $z
```

Tenure Review Tenure Review/Tenure Talk ...

- Much better!

**Math**

- "set" treats the value as a string
  
```bash
% set x = (2 + 3)
% echo $x
2 + 3
% echo $x[2]
+

% Use "@" to do math;
  - Note: the space between the "@" and the variable is REQUIRED
  
```bash
% @ x = (2 + 3)
% echo $x
5
```

- tcsh supports most of C’s expression operators (such as plus, minus, multiply, divide, less than, greater than, equal, etc.)
  
```bash
+ - * / < <= >= && || ! + + = -= *= /=
```

**Input/Output Redirection**

- tcsh can redirect input and output
  - it can also redirect error output (not shown)
  
```bash
% date
Sun Aug 20 11:11:10 MDT 2000
% date > today
% more today
  Sun Aug 20 11:11:14 MDT 2000
% rev < today
  0002 TDM 41:11:11 02 guA nuS
```

**Control Flow Constructs**

- Conditional
  
```bash
if ($x > 3) then
  echo true
else
  echo false
endif
```

- Iteration
  
```bash
while !(isdone)
  ...
end
```

```bash
foreach directory (bin build lib)
  mkdir $directory
end
```
Control Flow Constructs, cont.

- multi-branch
  switch ($char)
    case a:
      echo character is “a”
      breaksw
    default:
      echo character is not “a”
      breaksw
  endsw

- tcsh does not have a for loop construct...e.g.,
  for (x = 0; x < 5; x++)
    ...
  end
- ...use a while loop instead
  @ x = 0
  while ($x < 5)
    ...
    @ x++
  end

File Inquiry operations

- tcsh has file inquiry operators. They can be used in expressions.
  % set filename =~/.cshrc
  % echo $filename
    /home/ken/.cshrc
  % if (-e $filename) echo true
     true
  % @ x = (5 + -e $filename)
  % echo x = $x
     x = 6

Job control

- When you invoke a program, you are executing a “job”
- You can find out what jobs are running with the “jobs” command
- Typically, a job “suspends” the shell until it has finished running, e.g. when you invoke the “ls” program, the shell waits until ls generates its output
  - You can run a job in the background with an ampersand “&”, e.g. “% emacs &”
- You can suspend a running job using C-z (control-z)
- You can interrupt a running job using C-c
- A job can be brought into the foreground with “fg” and placed into the background with “bg”

Wildcards

- The shell, and some other UNIX programs (such as find) make use of wildcard characters. The name wildcard comes from card games where a “wild” card can stand for any other card. We will also call wildcard characters, metacharacters.
  % ls
    graph.c graph.h main.c stack.c stack.h
  %ls “.c"
    graph.c main.c stack.c
- Note: Is does not do the wildcard search, the shell does. If you do not want the shell to perform a wildcard search, then you need to quote all metacharacters
  %ls “*.c”
    Is: *.c: No such file or directory
Wildcards and their meanings

- * - match 0 or more instances of any character
- ? - match a single instance of any character
- [123ab] - match a single instance of any character within the brackets
- [0-9] - Shorthand for [0123456789]
  - The range is based on the ASCII character set. So, [a-Z] does not capture lowercase and uppercase letters. Use [a-zA-Z] instead
- [*0-9] - Match a single instance of any character except those specified in the brackets
- {pattern1,pattern2, …} - Match one of the listed patterns

Wildcard Examples

- Consider a directory with the following files:
  - aa aba a123aa baa Abbba3ab

\[
\text{% is a* } \\
\text{aa aba a123aa a3ab}
\]

\[
\text{% is a? } \\
\text{aa}
\]

\[
\text{% is [ab][123][ab][ab] } \\
\text{a3ab}
\]

\[
\text{% is *b[a-z] } \\
\text{aba Abbba3ab}
\]

\[
\text{% is \{a?,*b\} } \\
\text{Abba3ab aa}
\]

Pattern Matching

- Wildcards are one form of pattern matching. Another form of pattern matching is based on a formalism known as “regular expressions.”
- We need to make this distinction since some programs, such as grep and awk, use regular expression pattern matching rather than wildcard pattern matching.
- Unfortunately, the syntax for each uses the same characters but in different ways!
  - Actually, the situation is worse (especially for newcomers). Some metacharacters remain the same, but some are different in rather significant ways!

Regular Expression Syntax

- . - match a single instance of any character except newline
- [123ab] - Match a single instance of any character within the brackets
- [0-9] - Shorthand for [0123456789]
- [*0-9] - Match a single instance of any character except those specified in the brackets
- pattern1 | pattern2 | … - Match one of the listed patterns
- Question: if [a-z] means match a single instance of “a, b, c, d, …, z”, how do I match a “-” in a range expression?
Regular Expression Syntax, cont.

- ^ - Match the beginning of the line
- $ - Match the end of the line
- * - Match zero or more repetitions of the previous regular expression
- + - Match one or more repetitions of the previous regular expression (requires egrep)
- ? - Match zero or one repetitions of the previous regular expression (requires egrep)
- \ - Remove the special meaning of the next character

Wildcard and RE Differences

- In regular expressions “*” means something very different than it does in wildcards
- In wildcards a* matches a, aa, abc, a52b, ...
- In regular expressions a* matches only a, aa, aaa, aaaa, ...
- This also applies to “+” and “?”. They do not stand for any characters themselves, but rather modify the previous regular expression.
  - So, how do you search for an “*”?
    - a* matches only “a*”

Regular Expression Strategies

- When using a regular expression to find a desired search string, be aware of the following three quantities
  - Hits
    - Lines you wanted to match
  - Omissions
    - Lines you didn’t match but wanted to match
  - False Alarms
    - Lines you matched but didn’t want to match

Looking for the word “book”

%cat example
This file tests for book in various places, such as book at the beginning of a line or at the end of a line book as well as the plural books and handbooks
%grep “ book ” example - matches only line 1
%grep “book” example - matches all lines

How would we match lines 1, 2, and 3?