Unix Programming Environment

• Objective: To introduce students to the basic features of Unix and the Unix Philosophy (collection of combinable tools and environment that supports their use)
  – Basic commands
  – File system
  – Shell
  – Filters (wc, grep, sort, awk)

Many of the examples for this lecture come from the UNIX Prog. Env. and AWK books shown (see lecture outline for full references)
Getting Started

• Getting a CS account
  – Lab in Univ. Crossings 151
  – tux.cs.drexel.edu, queen.cs.drexel.edu
  – lab machines and tux running linux, queen running solaris

• ssh (part of Drexel CD)
  http://www.drexel.edu/IRT/services/software/

• cygwin (www.cygwin.com)

• loggin on and out
Command Line Interface

[jjohnson@ws44 jjohnson]$ echo hello
hello

[jjohnson@ws44 jjohnson]$ date
Tue Nov 30 05:24:34 EST 2004

[jjohnson@ws44 jjohnson]$ uptime
05:24:40 up 8 days,  5:19,  6 users,  load average: 1.22, 1.26, 1.63

[jjohnson@ws44 jjohnson]$ who
ummaycoc pts/0 Nov 23 09:56 (node4.uphs.upenn.edu)
jmn27 pts/1 Nov 30 01:06 (mst.cs.drexel.edu)
kn42 pts/2 Nov 30 02:09 (n2-202-96.resnet.drexel.edu)
jjohnson pts/4 Nov 30 05:23 (n2-19-88.dhcp.drexel.edu)
ks347 pts/6 Nov 30 02:59 (pcp04354303pcs.glstrt01.nj.comcast.net)
jmn27 pts/3 Nov 30 01:33 (mst.cs.drexel.edu)
Command Line Interface

[jjohnson@ws44 jjjohnson]$ finger jmn27
Login: jmn27  Name: John Novatnack
Directory: /home/jmn27  Shell: /usr/local/bin/tcsh
On since Tue Nov 30 01:06 (EST) on pts/1 from mst.cs.drexel.edu
  3 hours 38 minutes idle
On since Tue Nov 30 01:33 (EST) on pts/3 from mst.cs.drexel.edu
  3 hours 38 minutes idle
Mail last read Tue Jan  4 15:53 2005 (EST)
Plan:
hey i'm john
Command Line Interface

• options (usually designated with -)

• who -q
Getting Help

- manual
  $man who
- info
  $info who
- internet

The linux documentation project
  (http://www.tldp.org/)
- safari
- online
- friends and others
$ man who

WHO(1)          User Commands       WHO(1)

NAME
   who - show who is logged on

SYNOPSIS
   who [OPTION]... [ FILE | ARG1 ARG2 ]

DESCRIPTION
   -a, --all
       same as -b -d --login -p -r -t -T --u
   ...
   -q, --count
       all login names and number of users logged on
   ...

SEE ALSO
   The full documentation for who is maintained as a Texinfo manual. If
   the info and who programs are properly installed at your site, the
   command
       info coreutils who
   should give you access to the complete manual.
$ info who


`who': Print who is currently logged in

`who' prints information about users who are currently logged on.
Synopsis:

`who' [OPTION] [FILE] [am i]

If given no non-option arguments, `who' prints the following information for each user currently logged on: login name, terminal line, login time, and remote hostname or X display.

If given one non-option argument, `who' uses that instead of `/etc/utmp' as the name of the file containing the record of users logged on. `/etc/wtmp' is commonly given as an argument to `who' to look at who has previously logged on.
File System

- Organized into a tree of directories starting at the root

```
/    \
|     |
bin dev etc usr tmp
|     |
me you them
|     |
junk stuff
```
File System

- absolute and relative paths
- `/usr/me/stuff`
- `. and ..`
- Commands for traversing file system
  - `pwd, cd, ls`
- Commands for viewing files
  - `cat, more, less`
File System

• absolute and relative paths
• /usr/me/stuff
• . and ..
• Commands for traversing file system
  – pwd, cd, ls
• Commands for viewing files
  – cat, more, less, od
File System

- Commands for copying, removing and linking files
  - `cp`, `mv`, `rm`, `ln`
- Commands for creating and removing directories
  - `mkdir`, `rmdir`
- Archiving directory structure
  - `tar`, `gzip`, `gunzip`
File System

• File permissions
  – owner, group, world (everyone else)
  – chgrp, chown, ls –l, chmod
File System

```
[jjohnson@ws44 winter]$ ls -l
total 24
drwxr-xr-x 7 jjohnson users 80 Jan 3 2005 cs265/
-rw------- 1 jjohnson users 8258 Jan 3 2005 cs265.html
-rw-r--r-- 1 jjohnson users 8261 Jan 3 2005 cs265.html~
[jjohnson@ws44 winter]$ chmod 644 cs
 cs265 cs265.html cs265.html~
[jjohnson@ws44 winter]$ chmod 644 cs265.
 cs265.html cs265.html~
[jjohnson@ws44 winter]$ chmod 644 cs265.html
[jjohnson@ws44 winter]$ ls -l
total 24
drwxr-xr-x 7 jjohnson users 80 Jan 3 2005 cs265/
-rw-r--r-- 1 jjohnson users 8258 Jan 3 2005 cs265.html
-rw-r--r-- 1 jjohnson users 8261 Jan 3 2005 cs265.html~
```
shell

- command interpreter (bash, sh, csh,...)
- .bashrc, .profile
- PATH and shell variables
- metacharacters
- history and command completion
- file redirection
- pipes
- process management
editor

- A text editor is used to create and modify files.
- The most commonly used editors in the Unix community are vi (improved vi – vim) and emacs.
- You must learn at least one of these editors (you can get started quickly – use info and go through a tutorial – and learn more as you start using it).
- Tutorial for vim
  - $ vimtutor
filters

- Programs that read some input, perform a simple transformation on it, and write some output.
- Examples
  - wc
  - tr
  - grep, egrep
  - sort
  - cut
  - uniq
  - head, tail
grep

- search for lines matching pattern in specified files.
  - In the simplest case, search for given string (file and matching line are shown)

```
$ grep main *.cpp
assign31.cpp: *     The main program queries the user to provide assignments of
          truth values to the
assign31.cpp:int main()
bestval.cpp:int main()
bestval.cpp:     string remainder; /* read remainder of line */
bestval.cpp:     getline(cin, remainder);
max.cpp:int main()
set.cpp:int main()
tstr.cpp:int main()
```

- More generally regular expressions are used for patterns
Regular Expressions

• There are three operators used to build regular expressions. Let R and S be regular expressions and L(R) the set of strings that match R.

  – Union
    * R|S
      L(R|S) = L(R) ∪ L(S)

  – Concatenation
    * RS
      L(RS) = {rs, r ∈ R and s ∈ S}

  – Closure
    * R*
      L(R*) = {ε,R,RR,RRR,…}
Regular Expressions

- a|(ab)
- (a|(ab))|(c|(bc))
- a*
- a*b*
- (ab)*
- a|bc*d
- letter = a|b|c|…|z|A|B|C|…|Z|_
- digit = 0|1|2|3|4|5|6|7|8|9
- letter(letter|digit)*
Unix Syntax for Regular Expressions

- Many Unix commands (grep, egrep, awk, editors) use regular expressions for denoting patterns. The notation is similar amongst commands, though there are a few differences (see man pages)
- It pays to get comfortable using regular expressions (see examples at the end)
grep and egrep Regular Expressions
(decreasing order of precedence)

c any non-special character matches itself
\c turn off any special meaning of character c
^ beginning of line
$ end of line
. any single character
[...] any one of the characters in …; ranges like a-z are legal
[^...] any single character not in …; ranges are legal
\n what the nth character \((\ldots)\) matched (grep only)
r* zero or more occurrences of regular expression r
r+ one or more occurrences of regular expression r
r1r2 regular expressions r1 followed by r2
r1|r2 regular expressions r1 or r2 (egrep only)
\(r) tagged regular expression r (grep only); can be nested
( r) regular expression r (egrep only); can be nested
No regular expression matches a new line
pipes and combining filters

• Connect the output of one command to the input of another command to obtain a composition of filters

  • who | wc –l
  • ls | sort –f
  • ls –s | sort –n
  • ls –l | sort +3nr
  • ls –l | grep ‘^d’
Awk

- Awk is a convenient and expressive programming language that can be applied to a wide variety of computing and data manipulation tasks.

- It can be used as a filter.

```
pattern {action}
pattern {action}
...
```
Awk Features

- Patterns can be regular expressions or C like conditions.
- Each line of the input is matched against the patterns, one after the next. If a match occurs the corresponding action is performed.
- Input lines are parsed and split into fields, which are accessed by $1,\ldots,\text{NF}$, where NF is a variable set to the number of fields. The variable $0$ contains the entire line, and by default lines are split by white space (blanks, tabs)
**Example**

```bash
$ cat emp.data
Beth   4.00    0
Dan    3.75    0
Kathy  4.00    10
Mark   5.00    20
Mary   5.50    22
Susie  4.25    18
```

```bash
awk '$3 > 0 { print $1, $2 * $3 }' emp.data
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Hours</th>
<th>Overtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kathy</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Mark</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Mary</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>Susie</td>
<td>76.5</td>
<td></td>
</tr>
</tbody>
</table>
Associative Arrays

• Awk supports arrays that can be indexed by arbitrary strings. They are implemented using hash tables.
  – Total[“Sue”] = 100;

• It is possible to loop over all indices that have currently been assigned values.
  – for (name in Total) print name, Total[name];
Example using Arrays

$ cat scores
Fred 90
Sue 100
Fred 85
Sam 70
Sue 98
Sam 50
Fred 70

$ awk -f total.awk scores
Sue 198
Sam 120
Fred 245

$ cat total.awk
{ Total[$1] += $2}
END { for (i in Total)
    print i, Total[i];}
Problem 1

• Find all words that contain all of the vowels in alphabetical order.

  • ab•ste•mi•ous adj : sparing in use of food or drink : temperate — ab•ste•mi•ous•ly adv — ab•ste•mi•ous•ness n

  • (c)2000 Zane Publishing, Inc. and Merriam-Webster, Incorporated. All rights reserved.
Problem 1 Solution

A file containing words in a dictionary is usually available in different Unix systems (e.g. look in /usr/dict/words or /usr/share/dict/words)

$ grep '.*a.*e.*i.*o.*u.*' < /usr/dict/words

adventitious
facetious
sacrilegious

• What if you only wanted one occurrence of each vowel?
Problem 2

- Partial Anagram: Find all words that can be made from the letters in Washington/

- a, ago, ah, an, angst, …
Approach

• Instead of generating all possibilities and checking the result to see if it is a word, check each word to see if it is a partial anagram.

• To check a word
  – see if it has the right letters
  – make sure each letter occurs an allowable number of times
Problem 2 Solution

```bash
$tr A-Z a-z </usr/dict/words | \\
egrep '^[aghinostw]*$' | \\
egrep -v \\
'a.*a|g.*g|h.*h|i.*i|n.*n.*n|o.*o|s.*s|t.*t|w.*w'
```

a
ago
ah
an
angst
Problem 3

opts post pots spot stop tops

• Two words are anagrams if one can be obtained from the other by permuting the letters. For example, the words stop and pots are anagrams since the letters "s", "t", "o", and "p" in stop can rearranged to obtain the word pots.

• Given a list of words find all anagrams
Approach

• Two strings are anagrams if the sorted characters in the strings are equal.

• For each word in the list create a key by sorting the characters.

• Sort the list by their keys

• Anagrams are now adjacent

• Fold them into anagram classes
Problem 3 Solution

• Use the composition of three filters
  – sign (C program that reads a list of words, one per line, and for each words sorts the letters and prints the sorted letters and original word – it uses sort function, qsort, from the standard C library)
  – sort (Unix command to sort the output produced by sign)
  – squash.awk (awk program to produce anagram classes – it combines and prints words with the same key, which must be adjacent when sorted by key)
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define WORDMAX 101

int main()
{
    char thisword[WORDMAX], sig[WORDMAX];
    int compchar();
    while (scanf("%s",thisword) != EOF) {
        strcpy(sig, thisword);
        qsort(sig, strlen(sig), sizeof(char), compchar);
        printf("%s %s
", sig, thisword);
    }
    return 0;
}
squash.awk

$1 != prev { prev = $1; if (NR > 1) printf "\n" }
             { printf "%s ", $2 }
END        { printf "\n" }
Anagram Filter

$ sign < /usr/share/dict/words| sort | awk -f squash.awk > out

This produces a file, out, with each line containing an anagram class (i.e. list of words that are anagrams)

To find the largest anagram classes, we need to count the number of words in each class and sort by the counts. The command tail is used to select the 10 largest classes

$ awk '{ print NF " " $0}' < out | sort -n | tail
Anagram Output

5 mate meat meta tame team
5 mates meats steam tames teams
5 palest pastel petals plates staple
5 pores poser prose ropes spore
5 reins resin rinse risen siren
5 restrain retrains strainer terrains trainers
6 caret cater crate react recta trace
6 caster caters crates reacts recast traces
6 opts post pots spot stop tops
6 pares parse pears rapes reaps spare