csci-e113 class 1
19 sep 2004

1. Welcome

Welcome to csci-e113, Intro to C/Unix/CGI programming. Tonight's class will introduce you to the topics, goals, and methods of the course. The outline is:

a. brief summary of course
b. demonstration of Unix programming
c. detailed description of course
d. demonstration of Unix as an operating system
e. demonstration of CGI Unix programming
f. summary and wrapup - The big idea

A. Brief Summary of Course

This course is an introduction to C and Unix programming with web programming as an example of how to use C/Unix. The time will be roughly 60% C programming, 25% Unix/Shell programming, and 15% HTML/CGI programming. These numbers are not exact because I sometimes teach two topics in combination at once.

This is an intro to C/Unix, not an intro to programming. If you do not know how to write computer programs, you are not prepared for the course. An intense desire to learn the material or friends who know how to program won't help. Please take an intro to programming course first, then come back. I have been teaching this course since 1987, I am likely to be here next year.

Rather than talk in generalities about what C/Unix programming are, we shall start with a real-world example.

B. Demonstration of Unix Programming

How many of you folks ride the commuter rail to work or to get here? Say you wanted to find the trains you could take after class. We can look at the MBTA web site to see.

Notice how you can select a line, a direction, and a day to see a schedule. You can examine the schedule to figure out which train to take. The site also has a trip planner. There, you can type in the staring point and ending point and when you want to arrive or leave and the computer tells you which trains and connections to take. You've probably seen that sort of thing for air travel and for driving directions. In fact, a lot of web usage is travel related. How does that work? Could you build a travel planning web site? By the end of this course you should be able to.

Imagine it was several years ago, and the MBTA was just starting on this website travel planning, schedule reporting project. Imagine you work for the MBTA already or were hired as a software consultant and they tell you what they've done and what they want. They tell you that someone already began working on some sort of database. That person used a spreadsheet program to record data from the printed paper schedules. So, here's the project:

1. The dataset

The dataset is a transcription of all the MBTA rail schedules as a flat file with one line per event- train, station, day, time, direction, line. Compare these records to the schedule we see on the web page. Each time a train stops is an event, each event is one line in the file.

A trip is a set of these events. A single station has several trains per day, that list is a different set of events.

So if we want to see all the incoming trains stopping in Brockton, we look for all rows where the station is brockton, and if we want to see all the stops on train 017, we look for all rows where the train# is 017.

2. Tasks

There are various sthings we can do with a dataset like this:

- clean it (find typos, missing data, format errors..)
- analyze it: get stats and info
- report: generate schedules
- search : trip planning
- webify : build remote access to reports, stats, and searches

3. How would you do it?

There are lots of data management and web access tools out there. How would you folks do this, and how long would it take to get clean, analyze, and report, and webify some of those?

4. Unix as a programming language

Here are 12 questions one can ask about the train system:

1. When do trains leave from Braintree going to Boston?
2. What is the time of the earliest train from Ashland to Boston?
3. How many trains stop at West Medford on a weekday?
4. List the stations on the Fitchburg line.
5. List all the lines in the system.
6. List the train numbers of all trains passing through Beverly Depot.
7. What station has the most trains on Sunday?
8. Which line has the most stations?
9. During what hour does the greatest number of trains arrive at South Station?
10. When does the last train to Worcester leave Boston?
11. What is the most common train stop time?
12. What is the longest train trip (time, not distance) on the system?

Here are six programs you can use. All are available on any Unix system:

grep - find and print (line oriented)
cut - extract fields from lines
sort - sorts based on any selection of fields
uniq - removes (and/or counts) duplicate lines
head - selects first n lines
wc - counts chars, words, lines

Can we combine these six operations to answer the twelve questions?

Answer: pipelines -- combine tools to process data

Unix was originally built to do text processing and has always been based on text data and text processing. It is a univeral, portable format.

Unix Programming: combine tools into programs. To learn Unix programming, you need to learn to

a) use tools - shell scripts and pipelines
b) write tools - C

C. Detailed description of course

We go through the handout

course structure, homeworks, help, exams, academic honesty
workload - can be heavy
focus: C programming to build tools, scripts/pipelines to use them
HTML, CGI to run them over the Net

D. Unix for Users

Unix is an operating system that manages files, directories, programs, and users. The basic operations are:

login/logout
directories and files
  ls, mkdir, rmdir, cd, mv
files
  cat, more, rm, mv, cp
programs
i/o redir
IDE - vi/emacs, cc, gdb
Unix at home; Linux, BSD, knoppix

E. Scripts and Web Interfaces

Unix progrmming is based on the idea of combining software tools. We saw earlier how to combine software tools into a pipeline. Now we see how to write shell scripts to combine tools into a program.

first, consider train-times for Users:

```
#!/bin/sh
#
# train-times
#   purpose: list train times for a station
#     usage: train-times
#    action: script prompts for station name
#
     echo "Which station? "
     read STATION
     echo "inbound or outbound (i/o)? "
     read DIR
     echo "Trains passing through $STATION"
     grep "stn=$STATION" sched | grep "dir=$DIR"
```

We could instead, code it to accept the parameters as command line args, the way one passes args to a method.

Here is train-time-args for Unix programmers:

```
#!/bin/sh
#
# train-times-args
#    purpose: list train times for a station
#      usage: train-times-args stationname direction
#      where: direction is "i" or "o"
#
    STATION=$1
    DIR=$2
    grep "stn=$STATION" sched | grep "dir=$DIR"
```

Now we can write:

```
train-times-args salem o | cut -d";" -f4 | cut -d= -f2
```

Note, the use of command line args makes the script into a method woth args passes in as string values. There are two advantages:

a. the script can be used as a tool in a pipeline
b. the user interface can be a script OR a webpage

Putting a Webpage interface on this script:

We need two things:

1. A webpage - prompt the user and input values
2. Connector script - receive the strings and run the tools

Remarks: The system works, but the report format is crude. We need to process the output to make it look nicer. Therefore we need to find or write some tools to manage the output format.

F. The big picture

In this class we've seen what Unix/C/CGI programming is about

a. there are tools that perform specific operations
b. tools are often written in C
c. pipelines and scripts can combine those tools to solve problems
d. HTML forms and CGI connector scripts allow remote access
e. The skills to learn

C programming to build tools to process text
shell scripts/pipelines to combine tools into programs
HTML pages to accept user input
shell scripts to pass user input to Unix programs