3 Lecture: Getting started with Unix

Outline:

Announcements
Thoughts on the Assignment: detab
From last time: A C Program's Environment
From last time: A C Program's layout
Pointer Types
Arrays and Strings in C
Coping in the UNIX environment
Unix Family tree
Unix Stuff
    RTFM
The Shell
User-level UNIX
    Useful Commands
Foreshadowing
From last time: A C Program's Environment
A C Program's layout
Aliasing `rm` to move to Trash
The snapshots directory in the CSL

3.1 Announcements

- Coming attractions:

Event	Subject	Due Date	Notes

  Use your own discretion with respect to timing/due dates.

- Asgn1 test script `~pn-cs357/bin/tryAsgn1`

```
  The UNIX philosophy basically involves giving you enough rope to
hang yourself.  And then a couple of feet more, just to be sure.
```

- "`vim -u NONE`" to ignore .vimrc
- Paths and finding gcc (example)
- Line buffering
- Characters are *signed*
- binary files and `cat -v` and/or `od/xxd`
- Program that prints its own arguments
- You can find the test cases (The test script prints 'em)
- tryAsgn1 is online as ~pn-cs357/demos/tryAsgn1. Remember, error messages (usage messages, too) should go to stderr. (see `fprintf(3)`)
- Trivia (wrt)Asgn1

  - std{in,out,err}
  - redirection
  - tr trick
  - tilde expansion
- Be reading K&R. It'll help. (really):

Basic Language Concepts	Now	Chapters 2, 3, 4
Pointers and Memory	Next time	Chapters 5, 6
IO	Whenever	Chapters 7

- Lab 02 will be out shortly.
  1. a few quick written exercises
  2. Picture exercise
  3. A GDB exercise
  4. A Make exercise
  5. A simple utility, `uniq`.
- Piazza forum is live. I'll send invites as soon as enrollment has stabilized
- The syllabus has been updated with a calendar including exams

3.2 Thoughts on the Assignment: detab

- Think about what you know (and think like a typewriter)
- Think about what you don't know (line length?)
- The `tr(1)` trick.
- What would you do if this were a Java program? (it only differs by a few words).

3.3 From last time: A C Program's Environment

- It's *called*
- (parameters to main)
- Three open FILE *s: stdin/stdout/stderr
- Exit status (what good is it.)

3.4 From last time: A C Program's layout

The OS divides the running program (a *process*) into segments:

- Text
- Data
- Stack
- Heap

3.5 Pointer Types

- A pointer is a variable that stores a location
- Read pointer (really all) declarations inside→out and backwards. e.g.,

```
int *x
```

  as "x is a thing that points to an integer"

3.6 Arrays and Strings in C

Just like Java, except:

- Syntax is a little different
  - Type and shape.
  - (to come later) Arrays as parameters: The size of the first dimension can be left empty.
- Creation is the same as declaration.
- Not-resizable (There are techniques for dynamic allocation, but we'll talk about them later.)
- A string is an array of `char` with a zero byte at the end
- **No bounds checking**

No dynamic allocation is necessary for asgn1.

3.7 Coping in the UNIX environment

3.8 Unix Family tree

An overview of the family.

3.9 Unix Stuff

Connecting to the machine: ssh
A quick overview of the users' view of the system: (the *file* and the *process*) (The static and dynamic entities in the system.)

filesystem No matter how many disks and other devices exist on the machine, the system makes it look like one large filesystem starting at a "root" called /

On our systems, the home directory filesystem is shared everywhere.

users Individual user accounts each have their own identity and ownership. (uid, gid)

processes Every program execution creates a new process on the system. ps(1). A *process* is an active entity in the system, whereas a program is simply a file with execute permission.

Processes have identity:

- User ID

- Group ID
- Process ID

Processes have resources:

- memory
- time
- IO Streams (stdin,stdout,stderr)

commands Most user commands reside in one of three places: `/bin`, `/usr/bin`, or `/usr/local/bin`. When you try to run a program it looks in these locations.

shell The shell is the command interpreter you are talking to. The shell executes commands and keeps track of your environment. The shell can read commands interactively or from a file. (.login, .cshrc, .logout or .bashrc (interactive) and .bash_profile (login))

conventions:

- `ls(1)` means `ls` in section 1 of the manual
- files or directories are referred to by paths
  - A path starting at / is **absolute**
  - Any other path is **relative** (to the current working directory. Every process (and the shell is one) has a concept of a current working directory.
  - t csh and bash expand ~ to be your own home directory and *~user* to be *user's* home directory.

3.9.1 RTFM

Read the manual!

- On the system you're using
- How to know which headers you need?

3.10 The Shell

Shell families (/etc/shells)

The shell does everything necessary to run programs for you and sets up proper IO for them:

- Path searches (how to see your path)
- IO Redirection: "into" and "from" (> and <)
- IO Redirection: stderr

  {t,}csh: `a.out >& item`

  {ba,}sh: `a.out > outfile 2>& 1`

- The uses of bang (!)

- IO Redirection: pipes
- Different quotes
- Some shell behavior can be controlled by variables
  - Two kinds: shell and environment (don't worry about the difference now)
  - e.g. PATH, noclobber
- Globbing

3.11 User-level UNIX

Philosophy

- the system is designed for the expert (contrast Windoze)
- most commands do simple things. Many are designed as filters that can be combined into more complex systems.

```
cat Roster* | grep SO | sort | uniq | wc -l
```

- e.g. how many unique students are on the waitlist.
- tr example for `wc(1)` (or `expand(1)`) for seeing spaces.

3.11.1 Useful Commands

Subject areas:

Subject	Related commands/concepts
Connectivity	**ssh(1)**, scp(1)
Navigation	pwd(1), mkdir(1), rmdir(1), cd(1), (pushd(1), popd(1), dirs(1))
File manipulation	ls(1), cat(1), more(1), rm(1), mv(1), IO redirection
Process manipulation	ps(1), kill(1)
File permissions	ls -l, chmod(1)
Shell tricks	alias, .login, .cshrc, .bashrc, paths, redirection, running programs, shell and environment variables
Utilities	diff(1), grep(1), more(1), sort(1), uniq(1), lpr(1), ...
The Joy of X(7)	Life at the console is a good thing.
Other Stuff	Useful information

3.12 Foreshadowing

For the next few lectures will continue our crash course in C with some more advanced C topics. In particular:

1. Memory management (pointers, malloc(), free())
2. Compound data types (structs, arrays, unions?, typedef)
3. Development tools and techniques (headers, cpp, make, gdb)

We will see the first example of the difference between OS services (`sbrk(2)`) and C library services (`malloc(3)`).

3.13 From last time: A C Program's Environment

- It's *called*
- (parameters to main)
- stdin/stdout/stderr
- Exit status (what good is it?)

3.14 A C Program's layout

The OS divides the running program (a *process*) into segments:

- Text
- Data
- Stack
- Heap

3.15 Aliasing `rm` to move to Trash

This is in response to a question asked in class where I said I'd look up the answer. It's really straightforward in `csh` and derivatives, but a little more complicated in `bash`

- First, create your `Trash` directory:

```
mkdir ~/Trash
```

- Now, if you're running `csh` you create the alias thusly

```
% alias rm /bin/mv \!\* ~/Trash
```

- Now, if you're running `bash` you create a function, then alias it:

```
$ saferm () { /bin/mv $@ ~/Trash; }
$ alias rm=saferm
```

Either way, once you've worked out what you want it to be, you'll want to install it in your `.bashrc` or `.cshrc`.

3.16 The snapshots directory in the CSL

- `.shapshot`
- it'll save you
- even if you're not using source code control, back your stuff up.