Programming in C & Living in Unix

15-213: Introduction to Computer Systems
Recitation 6: Monday, Sept. 30, 2013
Arthur Chang
Section G
Weekly Update

- Buffer Lab is due Tuesday (tomorrow), 11:59PM
  - This is another lab you don’t want to waste your late days on.

- Cache Lab is out Tuesday (tomorrow), 11:59 PM
  - Let the coding in C begin!
  - Due Thursday October 10th
Agenda

- **Living in Unix (w/ Demo)**
  - Beginner
    - Command Line Interface
    - Basic Commands
  - Intermediate
    - Shell Scripting
    - More Commands

- **Programming in C (w/ Demo)**
  - Refresher
  - Compiling
  - Hunting Memory Bugs
Unix – Beginner: Command Line Interface

- Command Line Interface
  - “Provides a means of communication between a user and a computer that is based solely on textual input and output.”
  - In UNIX, the **shell** presents the user with a **command prompt** when it is ready to receive a new **command** on the **command line**.

- Shell
  - The program responsible for reading and executing the **commands** entered on the **command line**.
  - **sh** is the original UNIX shell.
  - Many other versions exist. (e.g. **bash**, **csh** and **zsh**)

The Linux Information Project: Command Line Definition
Unix – Beginner: Command Line Interface

- Command Prompt
  - AKA prompt or shell prompt
  - String before the command line that:
    1. Prompts the user “for the next command, data element or other input”.
    2. Helps “the user plan and execute subsequent operations.”

- Command Line
  - “The space to the right of the command prompt... in which a user enters commands and data.”

- Command
  - “An instruction given by a human to tell a computer to do something.”

The Linux Information Project: Command Line Definition
Unix – Beginner: Command Line Interface
# Unix – Beginner: Basic Commands

<table>
<thead>
<tr>
<th>Moving Around</th>
<th>Manipulating Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls</td>
<td>List directory contents</td>
</tr>
<tr>
<td>cd</td>
<td>Change working directory</td>
</tr>
<tr>
<td>pwd</td>
<td>Display present working directory</td>
</tr>
<tr>
<td>ln</td>
<td>Make links between files/directories</td>
</tr>
<tr>
<td>mkdir</td>
<td>Make directories</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Working Remotely</th>
<th>Looking Up Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssh</td>
<td>Secure remote login program</td>
</tr>
<tr>
<td>sftp</td>
<td>Secure remote file transfer program</td>
</tr>
<tr>
<td>scp</td>
<td>Secure remote file copy program</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Managing Processes</th>
<th>Other Important Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps</td>
<td>Report current processes status</td>
</tr>
<tr>
<td>kill</td>
<td>Terminate a process</td>
</tr>
<tr>
<td>jobs</td>
<td>Report current shell’s job status</td>
</tr>
<tr>
<td>fg(bg)</td>
<td>Run jobs in foreground (background)</td>
</tr>
</tbody>
</table>
Quick Aside: Man Page Sections

From man-db, the on-line manual database:

“Each page argument given to man is normally the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A section, if provided, will direct man to look only in that section of the manual. The default action is to search in all of the available sections, following a pre-defined order and to show only the first page found, even if page exists in several sections.”
Quick Aside: Man Page Sections

- Some programs/utilities/functions have the same name, this will require you to specify the section you want to search. (e.g. `man 3 printf`)

- Find the section with `whatis` or `man -f`, which will display the names, sections and short descriptions for all the matching pages.
Agenda

- **Living in Unix (w/ Demo)**
  - Beginner
    - Command Line Interface
    - Basic Commands
  - Intermediate
    - Shell Scripting
    - More Commands

- **Programming in C (w/ Demo)**
  - Refresher
  - Compiling
  - Hunting Memory Bugs
Unix – Intermediate: Shell Scripting

Why do you care about shell scripting in 15213?
- It will make your life easier. (e.g. customizing Bash with shortcuts)
- You might save time by writing a script to automate repetitive actions.
- Might as well start learning now; many of you will be working in a UNIX environment for many years to come.

What do we plan to teach you?
- In this recitation, Hello World with variables.
- Afterwards, only what you ask for help with.
Unix – Intermediate: Shell Scripting

- Our goal is to arm you with the basic knowledge and tools you’ll need to make your life easier during and after this class.

- For more information about shell scripting, check out Kesden’s old 15123 lectures 3, 4 and 5.
Unix – Intermediate: Shell Scripting

- Language can be very powerful
  - Functions, conditionals, loops

- Language can also be very weak
  - Completely un-typed (everything is a string)
  - Strict, unintuitive syntax (not very user friendly)

- Remains popular for its real power
  - Extensive library (can call any program)

- Relatively quick and easy to integrate command line tools to solve complex problems.
Unix – Intermediate: Shell Scripting

<table>
<thead>
<tr>
<th>hello.sh</th>
<th>hello.sh with variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/bin/sh</code></td>
<td><code>#!/bin/sh</code></td>
</tr>
<tr>
<td><code># Prints “Hello, world.” to STDOUT</code></td>
<td><code>str=&quot;Hello, world.&quot;</code></td>
</tr>
<tr>
<td><code>echo “Hello, world.”</code></td>
<td><code>echo $str</code></td>
</tr>
</tbody>
</table>

- “#!/bin/sh” tells the shell to run the script using /bin/sh.
  - Required to guarantee consistency.
  - People use different shells and each shell has a slightly different syntax and set of features.

- Anything after a ‘#’ is a comment.

- Variables
  - Setting a variable takes the form ‘varName=VALUE’.
    - There CANNOT be any spaces to the left and right of the “=”.
  - Evaluating a variable takes the form “$varName”.

<table>
<thead>
<tr>
<th>#!/bin/sh</th>
<th>#!/bin/sh</th>
</tr>
</thead>
<tbody>
<tr>
<td><code># Prints “Hello, world.” to STDOUT</code></td>
<td><code>str=&quot;Hello, world.&quot;</code></td>
</tr>
<tr>
<td><code>echo “Hello, world.”</code></td>
<td><code>echo $str</code></td>
</tr>
</tbody>
</table>
Quick Aside: Script Permissions

- When you first create a script, it is treated as if it were any other file, without the execute permission.
  - There are ways to circumvent this: `man umask`.

- You will need to use `chmod +x` to give yourself permission to execute your script.

- This only needs to happen once per script, unless you some how remove the permission bits again.
Quick Aside: Three Types of Quotes

- There are three different types of quotes, and they all have different meanings to the shell.
  - Unquoted strings are normally interpreted
  - “Quoted strings are basically literals, but $variables are evaluated.”
  - ‘Quoted strings are absolutely literally interpreted.’
  - `Commands in quotes like this are executed, their output is then inserted as if it were assigned to a variable and then that variable was evaluated.`
## Unix – Intermediate: More Commands

<table>
<thead>
<tr>
<th>Transforming Text</th>
<th>Useful with Other Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>cut</td>
<td>Remove sections from each line of files (or redirected text)</td>
</tr>
<tr>
<td></td>
<td>screen</td>
</tr>
<tr>
<td></td>
<td>Screen manager with terminal emulation</td>
</tr>
<tr>
<td>sed</td>
<td>Stream editor for filtering and transforming text</td>
</tr>
<tr>
<td></td>
<td>sudo</td>
</tr>
<tr>
<td></td>
<td>Execute a command as another user (typically root)</td>
</tr>
<tr>
<td>tr</td>
<td>Translate or delete characters</td>
</tr>
<tr>
<td></td>
<td>sleep</td>
</tr>
<tr>
<td></td>
<td>Delay for a specified amount of time</td>
</tr>
<tr>
<td>Archiving</td>
<td>Looking Up Commands</td>
</tr>
<tr>
<td>zip</td>
<td>Package and compress files</td>
</tr>
<tr>
<td></td>
<td>alias</td>
</tr>
<tr>
<td></td>
<td>Define or display aliases</td>
</tr>
<tr>
<td>tar</td>
<td>Tar file creation, extraction and manipulation</td>
</tr>
<tr>
<td></td>
<td>export (setenv)*</td>
</tr>
<tr>
<td></td>
<td>Exposes variables to the shell environment and its following</td>
</tr>
<tr>
<td></td>
<td>commands</td>
</tr>
<tr>
<td>Manipulating File Attributes</td>
<td>Searching Files and File Content</td>
</tr>
<tr>
<td>touch</td>
<td>Change file timestamps (creates empty file, if nonexistent)</td>
</tr>
<tr>
<td></td>
<td>find</td>
</tr>
<tr>
<td></td>
<td>Search for files in a directory hierarchy</td>
</tr>
<tr>
<td>umask</td>
<td>Set file mode creation mask</td>
</tr>
<tr>
<td></td>
<td>grep</td>
</tr>
<tr>
<td></td>
<td>Print lines matching a pattern</td>
</tr>
</tbody>
</table>

* – Bash uses `export`. Csh uses `setenv`. 
Quick Aside: Environment Variables

- Defined before the shell begins.

- Reflect an aspect of the shell environment.

- Changing environment variables affects the environment programs are executed in.

- Set and evaluated just like normal variables.

- `export` (bash) and `setenv` (csh) are used in scripts to export changes to environment variables to the scope of the shell.
Quick Aside: PATH

- How does the shell know which `ls` to execute?
  - The environment variable PATH.

- PATH is a : delimited list of directories to search for executables.
  - Can be set to include your shell scripts and C binaries.
Quick Aside: Customizing your Shell

- Shells can be configured by setting environment variables, adding aliases, running scripts and more.

- Most shells are setup by running a file to a series of files before the first command prompt is given.
  - Typically these files are hidden in the $HOME directory, but in the case of AFS they are not always set to run.
    - When using AFS, use $HOME/.login, which is run every login.
    - Bash typically uses .bashrc and csh typically uses .cshrc for example.

- Adding alias commands for commonly used commands is a useful and easy way to customize your shell.
Quick Aside: Using the `rm` Command

- `rm ./filename` – deletes file `filename`.

- `rm ./*name` – deletes all files in the current directory that end in `name`.

- `rm -r ./directory` – deletes all files inside the given directory and the directory itself.
Quick Aside: Using the rm Command

- `rm -r ./*` – deletes all files and directories inside the current directory.

- `sudo rm -rvf /*` – deletes the entire hard drive.
  - **DO NOT DO THIS!!!**
  - `sudo` will run the command as root, allowing you to delete anything.
  - `-v` (verbose) flag will list all the files being deleted.
  - `-f` (force) flag will delete files whose permissions would have normally asked for confirmation before deleting.
Agenda

- Living in Unix (w/ Demo)
  - Beginner
    - Command Line Interface
    - Basic Commands
  - Intermediate
    - Shell Scripting
    - More Commands

- Programming in C (w/ Demo)
  - Refresher
  - Compiling
  - Hunting Memory Bugs
C – Refresher: Things to Remember

- If you allocate it, you free it.
- If you use Standard C Library functions that involve pointers, make sure you know if you need to free it.
- Don’t pass structs into functions by value. Always use a pointer.
  - You should now be able to answer the question, “why is this bad?”
C – Refresher: Things to Remember

- There is no String type. Strings are just NULL terminated char arrays.

- Setting pointers to NULL after freeing them is a good habit, so is checking if they are equal to NULL.

- Global variables are evil, but if you must use make sure you use `extern` where appropriate.

- Define functions with prototypes for simplicity and clarity.
C – Refresher: Command Line Arguments

- If you want to pass arguments on the command line to your C functions, your main function’s parameters must be `main(int argc, char **argv)`

- `argv` is the command line string, parsed on space, in an array of char *’s (strings). `argv[0]` is the name of your compiled C binary.

- `argc` is the number of arguments and is always at least 1 because the binary’s name is always present.
C – Refresher: Echo Demo

Write a basic echo.c file that takes its arguments and prints them back out with the missing spaces and trailing newline.

Should compile using the following flags:
```bash
gcc -Wall -Wextra -Werror -pedantic -ansiecho.c -o echo
```
C – Refresher: Libraries

- Headers are used to expose interfaces through function and struct prototypes, defines and externing global variables.

- Aim to put implementation in *.c files and definition in *.h files.
C – Refresher: Libraries

- `#include `<* .h>` - Used for including header files found in the C include path: standard C libraries.
  - Specifying `-I DIR` on the `gcc` command line requests `gcc` to search `DIR` for headers before searching the rest of the include path.

- `#include “* .h”` – Used for including local header files.
C – Refresher: Remove Duplicates Demo

Write a basic remove_duplicates.c file that takes arguments from the command line and constructs a linked list of all the arguments, with duplicates removed and prints out how many different strings were given.

Example: “cat cat dog” has 2 items in the list, cat and dog.

Should compile using the following flags:

gcc -Wall -Wextra -Werror -pedantic -
anisiremove_duplicates.c linkedlist.c -oremove_duplicates_duplicates
Agenda

- Living in Unix (w/ Demo)
  - Beginner
    - Command Line Interface
    - Basic Commands
  - Intermediate
    - Shell Scripting
    - More Commands

- Programming in C (w/ Demo)
  - Refresher
  - Compiling
  - Hunting Memory Bugs
When compiling C code, all dependencies must be specified.

This will not compile because the dependency linkedlist.c is missing: `gcc -Wall -Wextra -Werror -pedantic -ansiremove_duplicates.c -oremove_duplicates`
C – Compiling: Command Line

gcc

<table>
<thead>
<tr>
<th>gcc</th>
<th>GNU project C and C++ compiler</th>
</tr>
</thead>
</table>

- gcc does not require these flags, but they encourage people to write better C code.

<table>
<thead>
<tr>
<th>Useful Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Wall Enables all construction warnings</td>
</tr>
<tr>
<td>-Wextra Enables even more warnings not enabled by Wall</td>
</tr>
<tr>
<td>-Werror Treat all warnings as Errors</td>
</tr>
<tr>
<td>-pedantic Issue all mandatory diagnostics listed in C standard</td>
</tr>
<tr>
<td>-ansi Compiles code according to 1989 C standards</td>
</tr>
<tr>
<td>-g Produces debug information (GDB uses this information)</td>
</tr>
<tr>
<td>-O1 Optimize</td>
</tr>
<tr>
<td>-O2 Optimize even more</td>
</tr>
<tr>
<td>-o filename Names output binary file “filename”</td>
</tr>
</tbody>
</table>
C – Compiling: Makefiles

| Make               | GNU make utility to maintain groups of programs |

- Projects can get very complicated very fast and it can take very long to have GCC recompile the whole project for a small change.

- Makefiles are designed to solve this problem by compiling only the necessary parts of a project and linking them to those unaltered.
C – Compiling: Makefiles

- Makefiles consist of one or more rules in the following form.

<table>
<thead>
<tr>
<th>Makefile Rule Format</th>
<th>Makefile for “gccfoo.cbar.cbaz.c –omyapp”</th>
</tr>
</thead>
<tbody>
<tr>
<td>target : source(s)</td>
<td>myapp: foo.obar.obaz.o</td>
</tr>
<tr>
<td>[TAB]command</td>
<td>gccfoo.obar.obaz.o –omyapp</td>
</tr>
<tr>
<td>[TAB]command</td>
<td>foo.o: foo.cfoo.h</td>
</tr>
<tr>
<td></td>
<td>gcc –cfoo.c</td>
</tr>
<tr>
<td></td>
<td>bar.o: bar.cbar.h</td>
</tr>
<tr>
<td></td>
<td>gcc –cbar.c</td>
</tr>
<tr>
<td></td>
<td>baz.o: baz.cbaz.h</td>
</tr>
<tr>
<td></td>
<td>gcc –cbaz.c</td>
</tr>
</tbody>
</table>
C – Compiling: Makefiles

<table>
<thead>
<tr>
<th>Make</th>
<th>GNU make utility to maintain groups of programs</th>
</tr>
</thead>
</table>

- Comments are any line beginning with ‘#’

- The first line of each command must be a TAB.

- Makedepend – tool for identifying dependencies.
  - Run on all your source files to add the correct dependencies to ‘Makefile’. (e.g. `makedepend foo.c bar.c baz.c`)
  - `gcc -MM` does the same thing but outputs to console.
C – Compiling: Makefiles

- Macros – similar to shell variables

<table>
<thead>
<tr>
<th>Makefile Rule Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC = gcc</td>
</tr>
<tr>
<td>CCOPT = -g –DDEBUG –DPRINT</td>
</tr>
<tr>
<td>#CCOPT = -02</td>
</tr>
<tr>
<td>foo.o: foo.cfoo.h</td>
</tr>
<tr>
<td>$(CC) $(CCOPT) –cfoo.c</td>
</tr>
</tbody>
</table>

- For more information on Makefiles, checkout Kesden’s old 15123 lecture 16.
Agenda

- Living in Unix (w/ Demo)
  - Beginner
    - Command Line Interface
    - Basic Commands
  - Intermediate
    - Shell Scripting
    - More Commands

- Programming in C (w/ Demo)
  - Refresher
  - Compiling
  - Hunting Memory Bugs
C – Hunting Memory Bugs: GDB

- Useful for debugging the occasional easy segfault.

- Run until segfault evaluate the situation using:
  - `where` – prints function stack and lines.
    - `up/down` – traverse the function stack.
  - `list` – prints source code for where you are in the function stack.
  - `display / print` – analyze the variables in use and see who is incorrectly using memory and why
C – Hunting Memory Bugs: Valgrind

- Great tool for finding memory problems in C programs.

- Examples of what valgrind’s memcheck tool can do are:
  - Track memory leaks
  - Track possibly lost blocks
  - Track origin for uninitialized values
  - Report definitely lost (and possibly reachable) blocks

- The verbose (-v) flag is recommended.
Sources and Useful Links

- The Linux Information Project: Command Line Definition
- Introduction to Linux: A Hands-On Guide (Garrels)
  - You should be comfortable with chapters 2, 3, 4 and 5.
- The On-line Manual Database
- Kesden’s 15213: Effective Programming in C and Unix
  - Lectures 3, 4 and 5 cover the basics of Shell Scripting.
  - Lecture 16 covers Makefiles and lecture 15 covers Valgrind.