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

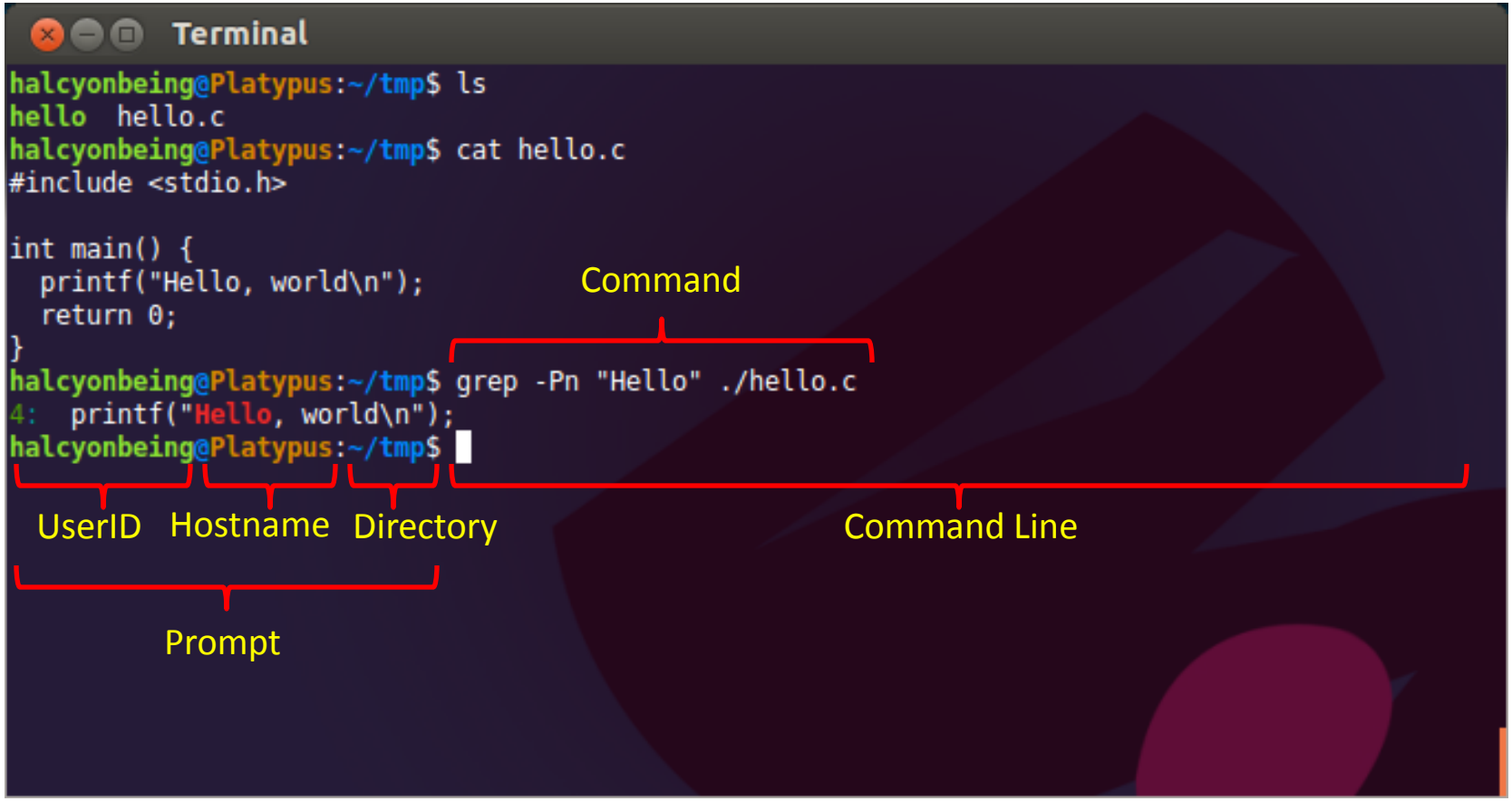
- Command Prompt
  - AKA **prompt** or **shell prompt**
  - String before the **command line** that:
    1. Prompts the user "for the next **command**, data element or other input".
    2. Helps "the user plan and execute subsequent operations."

- 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

Moving Around		Manipulating Files	
ls	List directory contents	mv	Move (rename) files
cd	Change working directory	cp	Copy files (and directories with “-r”)
pwd	Display present working directory	rm	Remove files (or directories with “-r”)
ln	Make links between files/directories	cat	Concatenate and print files
mkdir	Make directories	chmod	Change file permission bits
Working Remotely		**Looking Up Commands**	
ssh	Secure remote login program	man	Interface to online reference manuals
sftp	Secure remote file transfer program	which	Shows the full path of shell commands
scp	Secure remote file copy program	locate	Find files by name
Managing Processes		**Other Important Commands**	
ps	Report current processes status	echo	Display a line of text
kill	Terminate a process	exit	Cause the shell to exit
jobs	Report current shell’s job status	history	Display the command history list
fg(bg)	Run jobs in foreground (background)	who	Show who is logged on the system

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

hello.sh	hello.sh with variables

`#!/bin/sh`
 	
`# Prints "Hello, world." to STDOUT`
`echo "Hello, world."`
	`#!/bin/sh`
	
	`str="Hello, world."`
	`echo $str # Also prints "Hello, world."`

- "#!/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'.
    - ThereCANNOT be any spaces to the left and right of the "=".
  - Evaluating a variable takes the form "$varName".

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

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

* – 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.

- `sudorm -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(intargc, 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 itsarguments and prints them back out with the missing spaces and trailing newline.

Should compile using the following flags:

```
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 –
ansiremove_duplicates.clinkedlist.c –oremove_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

C – Compiling: Command Line

gcc	GNU project C and C++ compiler

- 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	GNU project C and C++ compiler

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

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

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

Make	GNU make utility to maintain groups of programs

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

Makefile Rule Format	Makefile for "gccfoo.cbar.cbaz.c –omyapp"
target : source(s) [TAB]command [TAB]command	myapp: foo.obar.obaz.o gccfoo.obar.obaz.o –omyapp foo.o: foo.cfoo.h gcc –cfoo.c bar.o: bar.cbar.h gcc –cbar.c baz.o: baz.cbaz.h gcc –cbaz.c

C – Compiling: Makefiles

Make	GNU make utility to maintain groups of programs

- 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. `makedependfoo.cbar.cbaz.c`)
  - `gcc -MM` does the same thing but outputs to console.

C – Compiling: Makefiles

Make	GNU make utility to maintain groups of programs

- Macros – similar to shell variables

Makefile Rule Format

```
CC = gcc
CCOPT = -g –DDEBUG –DPRINT
#CCOPT = -02

foo.o: foo.cfoo.h
        $(CC) $(CCOPT) –cfoo.c
```

- 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

valgrind	A suite of tools for debugging and profiling programs

- Great tool for finding memory problems in C programs.

- Examples of what valgrind'smemcheck 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.