# *Bash: how bash works*

Glenn Bruns
CSUMB

# Lecture Objectives

When this lecture is finally over, you should be able to:

- explain what a command-line interpreter is
- use basic command-line features of bash
- explain how bash finds commands

# Why the command line?

How to do this in the Windows GUI?

- for each file containing ‘operating system’:
  - see if the file contains ‘334’
  - if so copy the file to folder ‘OS 334’

It's hard to program a GUI!

GUIs are not designed for power users

# Command-line interpreters

Bash is a **command-line interpreter** (CLI)

Basic operation of a command-line interpreter:

- display a prompt
- accept user input
- "parse" input to get command and parameters
- run the command
- repeat

Example:

```
$ cat temp.txt          # command is 'cat'
$ mv temp.txt foo.txt   # parameters are temp.txt,
                          foo.txt
```

# bash is a powerful CLI

- the command line can be edited
- command history, including search
- filename completion
- full programmability

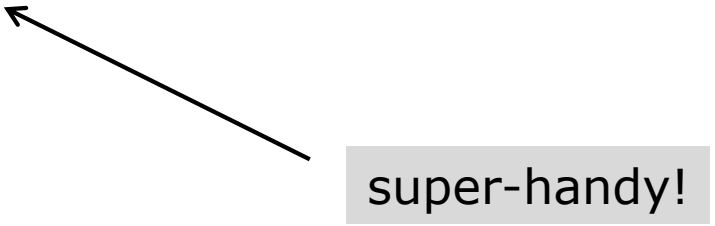
and much, much more!

# Command-line editing

- right/left arrow keys  (newbish)
- keyboard shortcuts  (pro tier)
  - ctrl-a  beginning of line
  - ctrl-e  end of line
  - ctrl-f  move forward one character
  - ctrl-b  move backward one character

# Command history

- up/down arrow keys
- keyboard shortcuts
  - ctrl-p go back one line in command history
  - ctrl-n go forward one line in command history
  - ctrl-r search history



# Filename expansion

- use tab key to complete
- if multiple files match, tab again to see them all

# Getting help

## Use ‘man’ to get info on a command

```
$ man cat
CAT(1)                          User Commands                          CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s), or standard input, to standard output.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonempty output lines
...
```

# Question

Are the commands you enter at the bash prompt implemented as part of bash?

Hint: if you wrote a C program, could you run it from the command line?

# Can bash find my code?

```
$ ls bin/hello
bin/hello
$ hello
hello!
$
$ cat my_test.c
#include <stdio.h>
int main() {
   printf("this is only a test\n");
   return 0;
}
$ gcc -o my_test my_test.c
$ my_test
-bash: my_test: command not found
$
$ echo $PATH
/usr/lib/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:
/home/CLASSES/brunsglenn/bin:/home/CLASSES/brunsglenn/msh/bin
$
```

# How does bash find commands?

- the process bash uses to find a command:
  - there is an **environment variable** named PATH
  - PATH gives a list of directories
  - bash looks for the command in each of the directories, starting with the first one
- to enable bash to find a command:
  - put it in one of the directories in PATH, **or**
  - add a new directory to PATH (don't put current directory in PATH)
  - if command in current directory, invoke the command with './' at the beginning

# The ‘which’ command

```
$ echo $PATH

/usr/lib/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbi
n:/sbin:/home/CLASSES/brunsglenn/bin:/home/CLASSES/brunsglenn
/msh/bin

$ which my_test

/usr/bin/which: no my_test in (/usr/lib/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbi
n:/sbin:/home/CLASSES/brunsglenn/bin:/home/CLASSES/brunsglenn
/msh/bin)

$
```

echo: displays a string

which: finds the location of a command

# Adding new commands

```
$ gcc hello.c –o hello    # compile ‘hello world’ program
$ hello                   # try to invoke it from bash
$ -bash: hello: command not found
$ ./hello           # tell bash to look in working dir.
hello!
$ mv hello /home/CLASSES/brunsglenn/bin
                    # or move it to a directory in PATH
$ hello
hello!
```

We have made our own program runnable from bash

# How to see if a command succeeds?

Commands have an exit status of 0 if they succeed

```
$ touch foo
$ rm foo
$ echo $?              # exit status stored in variable $?
0
$ rm foo
rm: cannot remove `foo': No such file or directory
$ echo $?
1
$
```

# Bash builtins

Some commands that bash runs are implemented by bash itself.

Examples: cd, pwd, echo, help

Question: why?

Answer: usually because they concern things within the shell.

Question: how to tell whether a command is a bash builtin?

Answer: 'man cd' shows bash man page; 'man ls' shows ls man page.  Or use builtin command 'type'

# Summary

- command-line interpreters are flexible, programmable, extendable
- the bash shell is a command-line interpreter
  - it has powerful features like command editing, command history, and filename completion
- bash searches for commands you enter by looking at the directories in the PATH variable
- Commands introduced in this lecture:
  - `man, echo, which, type`