Today

- Bash Configuration
- Aliases
- Shell/Bash scripting

1

Review: Unix Commands

- What is a link? What is the difference between a "regular" link and a symbolic link?
- How can we view the contents of files?
- What is a process?
  - What are commands related to processes?
- What are environment variables?
  - What are examples of environment variables?
  - How is the PATH environment variable used?
- How many lines are in your ~/.bash_history file?

2

Mystery Solved-ish

- It appears that you may not have the default config files in your accounts
  - Supposed to get copies of those files when account is created
- Confirm: Go to your home directory and run: `ls -a`
  - Check if you have the files: .bashrc, .bash_logout, and .profile
- If not, copy the files: `cp /etc/skel/.* ~`
  - Run `ls -a`
  - Source/load the .bashrc file:
    - `. .bashrc` **OR** `source ~/.bashrc`
  - Check the course's handouts directory again

Bash's Configuration Files

File Name	Purpose
`.profile`	Read by all shells
`.bashrc`	Read and executed by Bash every time you start an interactive non-login shell/subshell
`.bash_logout`	Read and executed every time a login shell exits
`.bash_profile`	Read and executed by Bash for an interactive login shell

Not provided by default in Ubuntu

Open your .bash* files in your favorite text editor
Notice what each file contains

Environmental Variables

Name	Meaning
$HOME	Absolute pathname of your home directory
$PATH	A list of directories to search for
$MAIL	Absolute pathname to mailbox
$USER	Your user name
$SHELL	Absolute pathname of login shell
$TERM	Type of terminal
$PS1	Prompt

To view ***all*** shell variables, `set`

The PATH environment variable

- Colon-separated list of directories
- Executables (without a specified path) are executed if found in a directory in the list
  - Searched left to right
- Example:

```
$ example.sh
-bash: example.sh not found
$ PATH=$PATH:.
$ example.sh
hello!
```

Examples of Impact of PATH

Not having . in your path:

```
$ ls
foo
$ foo
sh: foo: not found
```

```
$ ./foo
Hello, foo.
```


8

Examples of Impact of PATH

Not having . in your path:

```
$ ls
foo
$ foo
sh: foo: not found
```

```
$ ./foo
Hello, foo.
```

- What **not** to do:

```
$ PATH=.:$PATH
$ ls
foo
$ cd /tmp/
$ ls
Congratulations! Your files have been removed
and you have just sent email to Professor Watson
challenging him to a fight.
```


9

Examples of Impact of PATH

From Professor Watson:

Not having . in your path:

```
$ ls
foo
$ foo
sh: foo: not found
```

- What **not** to do:

```
$ PATH=.:$PATH
$ ls
foo
$ cd /tmp/
$ ls
Congratulations! Your files have been removed
and you have just sent email to Professor Watson
challenging him to a fight.
```


ALIAS

- Allow you to rename commands or type something simple instead of a list of options
- Can be defined on the command line, in .bashrc, .bash_profile, .bash_aliases

Default .bashrc says

```
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
```

Why might that be a good idea?

ALIAS

- Allow you to rename commands or type something simple instead of a list of options
- Can be defined on the command line, in .bashrc, .bash_profile, .bash_aliases
- To see all defined aliases
  - alias
- To see the definition for an alias
  - alias name
- To create an alias
  - alias name='command'

Deleting an ALIAS

- unalias name

- Just for the current shell/session

CONTROL-COMMANDS

14

Control-Commands

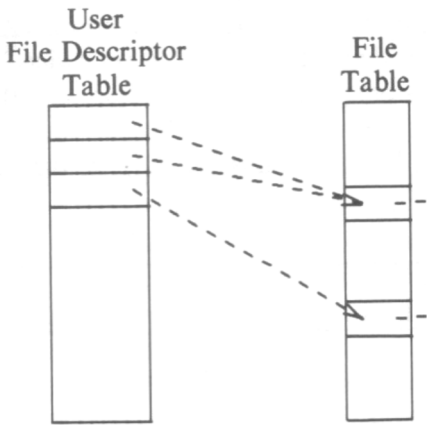
Control +	Function
c	Interrupt or break job; stops printing and returns to UNIX
z	Suspend current job bg to run in background
h	Erase or backspace character
s	Freezes screen
q	Unfreezes screen
u	Erase everything on line before this
w	Erase previous word
k	Erase remainder of line

15

FILE SYSTEM INTERNALS

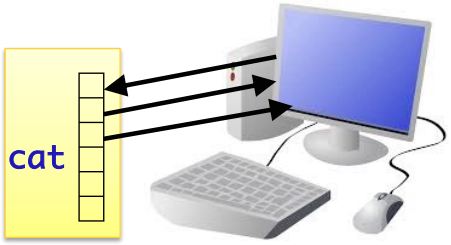
The File Descriptor Table

- Each process contains a table of files it has opened
- Inherits open files from parent
- Each open file is associated with a **number** or **handle**, called a **file descriptor** (fd)
- Each entry of this table points to an entry in the *open file table*
- Always starts at 0



Standard in/out/err

- The first three entries in the *file descriptor table* are special by convention:

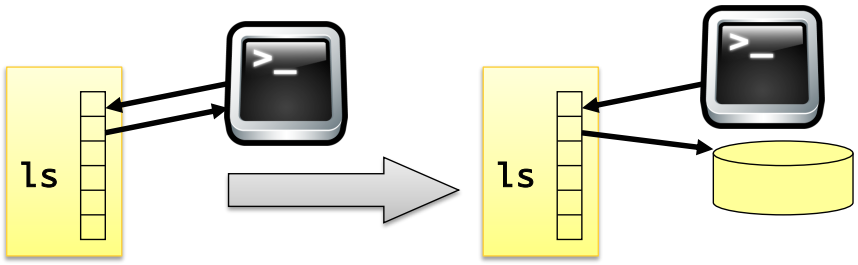


- Entry 0 is for *input*
- Entry 1 is for *output*
- Entry 2 is for *error* messages

18

Redirection

- Before a command is executed, the input and output can be changed from the default (terminal) to a file
  - Shell modifies file descriptors in child process
  - The child program knows nothing about this



19

Redirection of input/ouput

- Redirection of output: >
  - Example:`$ ls > my_files`
  - Can save output from one of your programs
- Redirection of input: <
  - Example: `$ wc < input.data`
- Append output: >>
  - Example: `$ date >> logfile`
- Bourne Shell derivatives: `fd>`
  - Example: `$ ls 2> error_log`

Redirecting Output

- Save output from a program
  - `java MyProgram > file.out`
  - Redirected `stdout` to `file.out`
  - `stderr` would still go to terminal
- To redirect `stderr` to file as well
  - `java MyProgram >& all.out`

Reminder of our goal (to be more like Jim):

SHELL SCRIPTING

Shell Scripts

- ***Script***: a shell program
- Tool for building applications by "gluing together" system calls, tools, utilities, and compiled binaries
- Just about everything we've done so far is available for use in a script
  - Adds even more
- Good for repetitive tasks that don't require a more structured programming language

Shell Scripting vs. [C/Python/Java] Programming

Advantages
Easy to work with/use other programs
Easy to work with directories, files
Easy to work with strings (easier than C, at least)
Good for prototyping

Shell Scripting vs. [C/Python/Java] Programming

Advantages	Disadvantages
Easy to work with/use other programs	Slower
Easy to work with directories, files	Not well-suited for algorithms and data structures
Easy to work with strings (easier than C, at least)	Syntax differences from what we're used to
Good for prototyping	

Scripts tend not to be long.
In some ways, we'll love it.

in some ways, we'll hate it.

Shell Scripts

- A shell script is a **text** file that contains shell or UNIX commands
- Kernel uses the *first line* of script to determine which shell script to use
  - `#!pathname-of-shell`
    - Kernel invokes pathname and sends the script as an argument to be interpreted
  - If `#!` is not specified, the current shell assumes it is a script in its own language
    - Can lead to problems – shells have different syntax
    - Best practice: specify the shell

Simple Example

```
#!/bin/bash

echo "Hello World"
```

Which shell to use

Command to execute
echo – like a print statement

See the available shells by executing:

```
ls -l /bin/*sh
```

Invoking a Script

- A script can be invoked as:
  - `sh scr_name [ arg … ]`
  - `sh < scr_name [ args … ]`
  - `path/scr_name [ arg …]`
    - Before running it, it must have execute permission:
      - `chmod +x scr_name`

Where `sh` is the shell you want

We'll typically use either the 1st or 3rd execution option and we'll use the `bash` shell

Your First Script

- Write a script called `first.sh`
  - Displays the files in your home directory
- Use your favorite text editor
- Recall good development practices
  - Build in pieces
  - Execute and test your script

Classifications of Shell Commands

Recall: A shell script is a **text** file that contains shell or UNIX *commands*

- Programs/Executables
  - Most programs that are part of the OS in **`/bin`, `/usr/bin`**
- Built-in commands
- Functions
- Aliases

```
$ type cat
cat is /usr/bin/cat
$ type ls
ls is aliased to `ls --color=auto'
$ type cd
cd is a shell builtin
$ type if
if is a shell keyword
```


Classifications of Shell Commands

All work the same in taking parameters and exit status

- Programs/Executables
  - Most programs that are part of the OS in **`/bin`, `/usr/bin`**
- Built-in commands
- Functions
- Aliases