Objectives

- Customizing your environment
  - Aliases
- Filtering commands
- Combining Commands
  - Semicolon
  - Pipes

Review

- How do we find out how much space some files/ directories are taking up?
- What command do we use to find a set of files that meet various criteria?
- How do we see the processes that are currently running?
- How can we see all our environment's variables?
- What are two different ways to set environment variables?
  - How long will the variables be set in each case?

Follow-up

- Ctl-S seems to freeze the terminal for one keystroke

Tar: Tape Archive

- File format
  - Used to archive or distribute files
    - Useful when we're downloading/installing new tools
  - Similar to Zip
- Utility:
  - Collects many files into one larger tar file OR
  - Extracts many files from one larger tar file

Tar: Tape Archive

- Syntax: `tar <options> <arguments>`
- Typical use:
  - Extracting file:
    - `tar xf archive.tar`
    - `tar xfz archive.tar.gz`
  - Creating file:
    - `tar cf archive.tar myDir`
    - `tar cfz archive.tar.gz files*`

Option	Meaning
c	Create archive file
x	Extract archive file
f	File
z	Compress (gzip)
v	Verbose

Archive file to create

Files to put into archive

CONFIGURING YOUR ENVIRONMENT

Environment of a Process

- A set of key-value pairs associated with a process
- Keys and values are strings
- Passed to children processes
- Cannot be passed back up
  - Meaning, what you do in the child doesn't affect parent
- Common examples:
  - **PATH**: Where to search for programs
  - **TERM**: Terminal type

The PATH environment variable

- Colon-separated list of directories
- Non-absolute pathnames of executables are only executed if found in the list
  - Searched left to right
- Example:

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


My PATH Variable

- In my `.bash_profile`:
  - `PATH=$PATH:$HOME/bin`
- What does the above line mean?
- What is the result?

Shell Variables

- Shells have several mechanisms for creating variables. A variable is a name representing a string value. Example: PATH
  - Shell variables can save time and reduce typing errors
- Allow you to store and manipulate information
  - Ex: `ls $DIR > $FILE`
- Two types: local and environmental
  - Local are set by the user or by the shell itself
  - Environmental come from the operating system and are passed to children

Shell Variables

- Syntax varies by shell
  - `varname=value # sh, ksh, bash`
  - `set varname = value # csh`
- To access the value: **`$varname`**
- Turn local variable into environment:
  - All child processes from this terminal
  - **`export varname # sh, ksh, bash`**
  - **`setenv varname value # csh`**

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`

Setting Environment Variables

- You can set environment variables in your `~/.bash_profile` file
- Open `~/.bash_profile` using jedit or emacs
- Create a new variable:
  - `CS297=/home/courses/cs297`
- Export the variable
  - `export CS297`
- In terminal, run the `source` command to load your new profile
  - `source ~/.bash_profile`
- Check that your new variable was created:
  - `echo $CS297`
- Use the variable
  - `cd $CS297`

Bash's Configuration Files

File Name	Purpose
`.bash_profile`	Read and executed by Bash every time you log into the system
`.bashrc`	Read and executed by Bash every time you start a subshell
`.bash_logout`	Read and executed every time a login shell exits

Open your .bash* files in jedit
Notice what each file contains

Customizing Your Prompt

- Open your `.bashrc` file
- Comment out your PS1 definition, if it exists
  - "Comment out" → #
- Default prompt is \u@\h>

Customizing Your Prompt

Code	Meaning
\d	the date in "Weekday Month Date" format (e.g., "Tue May 26")
\D{format}	the format is passed to strftime(3) and the result is inserted into the prompt string; an empty format results in a locale-specific time representation. The braces are required
\h	the hostname up to the first .
\H	the full hostname
\t	the current time in 24-hour HH:MM:SS format
\T	the current time in 12-hour HH:MM:SS format
\u	the username of the current user
\w	the current working directory, with $HOME abbreviated with a tilde
\W	the basename of the current working directory, with $HOME abbreviated with a tilde
\!	the history number of this command
\#	the command number of this command

Bash Color Excape Codes

- Use any of these escape codes between `\e[` and `m` to colorize text

Color	Code
Brown	0;33
Yellow	1;33
Light Gray	0;37
White	1;37

Trends in codes?

Color	Code
Black	0;30
Dark Gray	1;30
Blue	0;34
Light Blue	1;34
Green	0;32
Light Green	1;32
Cyan	0;36
Light Cyan	1;36
Red	0;31
Light Red	1;31
Purple	0;35
Light Purple	1;35

Bash Color Excape Codes

- What does this prompt look like?
  - `export PS1="\e[1;31m\h:\W \u\$ \e[00m"`

Bash Color Excape Codes

Escape the color codes so that it doesn't mess up the line-wrapping

- Lets make the prompt red:

```
export PS1="\[\e[1;31m\]\h:\W \u\$ \[\e[00m\]"
```

- Break it down:
  - `\e[` is the Escape Character sequence
  - `1;31m` Sets the color to light (01) red (31)
  - `\h:\W \u\$` Prints hostname:(current directory) username$
  - `\e[` Escape Character sequence
  - `00m` Clears all color (no color after this point)
- Mine looks like:
  - `hopper:~ sprenkle$`

Practice

- In .bashrc
  - `export PS1="\[\e[1;31m\]\h:\W \u\$ \ [\e[00m\]"`
- source your .bash_profile
- Make terminal small and use tab completion

ALIAS

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

Create a new ALIAS

- Open `~/.bashrc` and `.bash_profile`
- Move your definition of `CS297` and its export from `.bash_profile` to `.bashrc`
- Add an alias called `cd297` (or something easy to remember) that cds to the CS297 directory

Deleting an ALIAS

- `unalias name`
- Just for the current shell/session

PROCESS COMMUNICATION

Inter-process Communication

Ways in which processes communicate:

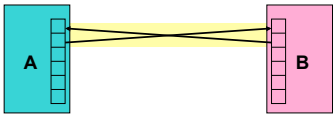
- Passing arguments, environment
- Read/write regular files
- Exit values
- Signals
- **Pipes**

One of the cornerstones of UNIX

PIPES

Pipes

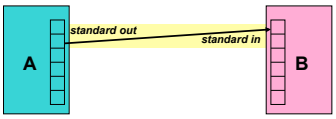
- General idea: The input of one program is the output of the other, and vice versa



- Both programs run at the same time

Pipes

- Often, only one end of the pipe is used



- Could this be done with files/redirection?

Redirection/File Approach

Run first program, save output into file

Run second program, using file as input

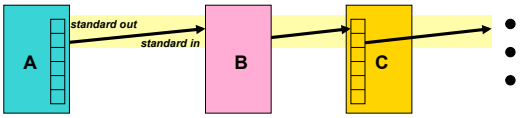
- Unnecessary use of the disk
  - Slower
  - Can take up a lot of space
- Doesn't take advantage of multi-tasking

Coordinating Pipes

- What if a process tries to read data but nothing is available?
  - UNIX puts the reader to sleep until data available
- What if a process can't keep up reading from the process that's writing?
  - UNIX keeps a buffer of unread data
    - This is referred to as the pipe size
  - If the pipe fills up, UNIX puts the writer to sleep until the reader frees up space (by doing a read)
- Multiple readers and writers possible with pipes

Filters

- Pipes are often chained together
  - Called *filters*



Pipelines

- Output of one program becomes input to another
  - Uses concept of UNIX *pipes*
- Example: `$ who | wc -l`
  - counts the number of users logged in
- Example: `$ find . –mtime 0 | wc`
  - Counts number of files created in last day
- Pipelines can be long:

```
who | awk '{print $1}' | sort | uniq
```


What's the difference?

- Both of these commands send input to ***command*** from a file instead of the terminal:

```
$ cat file | command
```

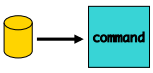
vs.

```
$ command < file
```


Difference: An Extra Process

```
$ cat file | command
```


```
$ command < file
```



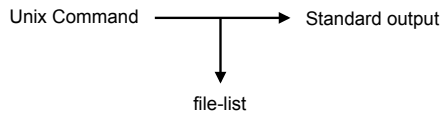
Introduction to Filters

- A class of Unix tools called ***filters***
  - Utilities that read from standard input, transform the input, and write to standard out
- Using filters can be thought of as *data oriented programming*
  - Each step of the computation transforms data *stream*

Examples of Filters

- **`sort`**
  - Input: lines from a file
  - Output: lines from the file sorted
- **`grep`**
  - Input: lines from a file
  - Output: lines that match the argument
- **`awk`**
  - Programmable filter

tee



- Read from standard input and write to standard output and one or more files
  - Captures intermediate results from a filter in the pipeline

tee

- Syntax: `tee [ -ai ] file-list`
  - `-a` append to output file rather than overwrite, default is to overwrite (replace) the output file
  - `-i` ignore interrupts
  - `file-list` one or more file names for capturing output
- Examples

```
ls | head –10 | tee first_10 | tail –5
who | tee user_list | wc
```

What is the end result of each command?

Unix Text Files: Delimited Data

Tab Separated

```
John    99
Anne    75
Andrew  50
Tim     95
Arun    33
Sowmya  76
```

Lots of other delimiters, e.g., commas or pipes

Why do we use delimiters?

Colon-separated

```
root:x:0:0:root:/root:/bin/bash
sgoryl:x:513:504:Steve Goryl:/home/faculty/sgoryl:/bin/bash
pinkhamd:x:514:504:Pinkham Derek:/home/faculty/pinkhamd:/bin/bash
sprenkle:x:205:500:Sara Sprenkle:/home/faculty/sprenkle:/bin/bash
```


cut: select columns

- `cut` prints selected parts of input lines
  - Can select columns (assumes tab-separated input)
  - Can select a range of character positions
- Some options:
  - `-f listOfCols` print only specified columns (tab-separated) on output
  - `-c listOfPos` print only chars in specified positions
  - `-d c` use character c as the column separator
- Lists are specified as ranges (e.g. 1-5) or comma-separated (e.g. 2,4,5).

cut examples

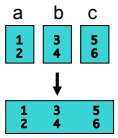
```
cut -f 1 data
cut -f 1-3 data
cut –f 4,2 data
cut -f 4- data
cut -d'|' -f 1-3 data
cut -c 1-4 data
```

Note how output is formatted
→ Columns joined by delimiter

Unfortunately, there's no way to refer to "last column" without counting the columns.

paste: join columns

- `paste` displays several text files "in parallel" on output
- If the inputs are files a, b, c
  - the first line of output is composed of the first lines of a, b, c
  - the second line of output is composed of the second lines of a, b, c
- Lines from each file are separated by a tab character
- If files are different lengths, output has all lines from longest file, with empty strings for missing lines



paste example

```
cut -f 1 data > data1
cut -f 2 data > data2
cut -f 3 data > data3
paste data1 data3 data2 > newdata
```

What is each command doing?
What is the final result?

sort: Sort lines of a file

- `sort` copies input to output but ensures that output is arranged in ascending order of lines.
  - By default, sorting is based on ASCII comparisons of the whole line
- Other features of sort:
  - Understands text data that occurs in columns. (can also sort on a column other than the first)
  - Can distinguish numbers and sort appropriately
  - Can sort files "in place" as well as behaving like a filter
  - Capable of sorting very large files

sort: Options

- `sort [-dftnr] [-o filename] [filename(s)]`

Option	Meaning
-d	Dictionary order, only letters, digits, and whitespace are significant in determining sort order
-f	Ignore case (fold into lower case)
-t	Specify delimiter
-n	Numeric order, sort by arithmetic value instead of first digit
-r	Sort in reverse order
-o	Filename – write output to filename, filename can be the same as one of the input files

- Lots more options…

uniq: list UNIQue items

- Remove or report adjacent duplicate lines
- `uniq [ -cdu] [input-file] [ output-file]`
  - `-c` Supersede the `-u` and `-d` options and generate an output report with each line preceded by an occurrence count
  - `-d` Write only the duplicated lines
  - `-u` Write only those lines which are not duplicated
  - The default output is the union (combination) of `-d` and `-u`

wc: Counting results

- The *word count* utility, **wc**, counts the number of lines, characters or words
- Options:
  - `-l` Count lines
  - `-w` Count words
  - `-c` Count characters
- Default: count lines, words and chars

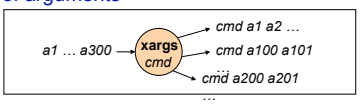
wc and uniq Examples

```
who | sort | uniq –d
wc my_essay
who | wc
sort file | uniq | wc –l
sort file | uniq –d | wc –l
sort file | uniq –u | wc -l
```

Why do we have to do sort before uniq?
Can't we just use uniq?
(How do you think uniq is implemented?)

xargs

- Unix limits the size of arguments and environment that can be passed down to child
- What happens when we have a list of 10,000 files to send to a command?
- xargs solves this problem
  - Reads arguments as standard input
  - Sends them to commands that take file lists
  - May invoke program several times depending on size of arguments



find utility and xargs

- `find . -type f -print | xargs wc -l`
  - `-type f` for files
  - `-print` to print them out
  - `xargs` invokes `wc` 1 or more times
- `wc -l a b c d e f g`
  `wc -l h i j k l m n o`
  `…`
- Compare to: `find . -type f –exec wc -l {} \;`

yes

- What does this command do?

yes

- Syntax: `yes [STRING]`
- Output a string repeatedly *until killed*

Danger: Deleting a Set of Files

- One solution:

```
find . -name "*~" -exec rm "{}" ";"
```

  - Seems to do forced `rm`, no interaction with user required
  - **LESSON**: Do `find` part first and verify want to do remove
- Alternative (not quite equivalent) solution:

```
yes | rm *~
```

Just in current directory

Example Execution

Answers y to each removal question

```
[sprenkle@hopper personal]$ yes | rm */*~
rm: remove regular file `craw-hobby/index.html~'? rm:
remove regular file `England2008/England2008.html~'? rm:
remove regular file `England2008/Page1.html~'? rm:
remove regular file `England2008/Page2.html~'? rm:
remove regular file `England2008/Page3.html~'? rm:
remove regular file `England2008/Page4.html~'? rm:
remove regular file `England2008/Page5.html~'? rm:
remove regular file `England2008/Page6.html~'? rm:
remove regular file `England2008/Page7.html~'?
[sprenkle@hopper personal]$
```

- Best practice: Do an `ls` using the regular expression to see what files you're going to delete, e.g., `ls */*~`
- Try the `rm` command, when it prompts you, say yes a few times.
- If it seems to be working, kill it and do the command with the `yes |`

Another Useful Shortcut

- On-the-fly modification of a previous command to create a new command
- the Bash shell uses the caret (^) character to perform substitutions:

```
[sprenkle@hopper day3]$ ls -l villains.txt
-rw-r--r-- 1 sprenkle cs297 0 2009-04-22 13:29 villains.txt
[sprenkle@hopper day3]$ ^villains^heroes
ls -l heroes.txt
-rw-r--r-- 1 sprenkle cs297 0 2009-04-22 13:29 heroes.txt
```

New command

Execute Multiple Commands: ;

- Can execute multiple commands on one line
- Example:

```
[sprenkle@pascal cs297]$ mkdir assigns; cd assigns
[sprenkle@pascal assigns]$
```


BY POPULAR DEMAND

Using the `talk` command

- talk service is disabled

Using the `mail` command

- Requirement: must be on a machine where the mail daemon is activated
  - Best bet: pascal
  - Message is from your `name@pascal.cs.wlu.edu`
- Examples:
  - `mail -s "Subject" <e_addr>`
    - Type message, ending message with a . on a line by itself or control-D
  - `cat message | mail -s "CSCI211: Grade" <e_addr>`

Using the `mail` command

- You can check your messages on pascal using the mail command
  - Not a pretty interface!

Assignment 3

- Due Monday