Review

- What is Maven?
  - How do you define a Maven project?
  - Why is it helpful to use Maven?
- What is a filter (in Unix)?
  - What are examples of filters? How do we use them?

Copy $HANDOUTS/pipes_practice directory into your cs397 directory

1

Fun with the Dictionary

- How many words have 3 a's one letter apart?
  - `grep "a.a.a" /usr/share/dict/words | wc -l`
    - 275
- How many words have 3 u's one letter apart?
  - `grep "u.u.u" /usr/share/dict/words | wc -l`
    - 4
- How many words violate the "i before e except after c" rule?
  - `grep "cie" /usr/share/dict/words | wc -l`
    - 182

2

Unix Text Files: Delimited Data

Tab Separated

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

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

Why do we use delimiters?

`/etc/passwd` *Colon-separated*

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```


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 student_info.csv
cut -f 1-3 student_info.csv
cut –f 2,1 student_info.csv
cut –f 2- student_info.csv
cut –d'@' -f 1 student_info.csv
cut -c 1-5 student_info.csv
```

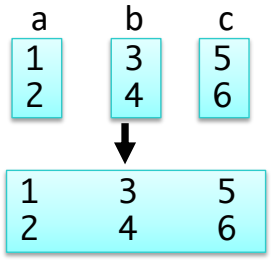
Note how output is formatted
→ Columns joined by delimiter

No way to refer to "last column" without counting columns.

Get me the list of usernames from the email addresses

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 student_info.csv > data1
cut -f 2 student_info.csv > data2
cut -f 3 student_info.csv > data3
paste data1 data3 data2 > newdata
```

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

sort: Sort lines of input

- 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`

Get the count of each last name
Find all the unique last names or first names

Using Unique

- View the contents of villains.txt
- What happens when you run:
  - `uniq villains.txt`
  - `uniq –c villains.txt`
  - `sort villains.txt | uniq`
  - `sort villains.txt | uniq -c`
  - `sort villains.txt | uniq –d`
  - `sort villains.txt | uniq -u`

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

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 | cut –f 1 -d" " | 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
```


FINDING FILES

Tree Walking

- How can do we find a set of files?
- One possibility:
  - `ls -lR /`
- What about
  - All files below a given directory in the hierarchy?
  - All files since Jan 1, 2022?
  - All files larger than 10K?

`find` utility

- **`find`** *<pathlist> <expression>*
- **`find`** recursively descends through *pathlist* and applies *expression* to every file
- *expression* can be:
  - `-name` *pattern*
    - *true* if file name matches pattern. Pattern may include shell patterns such as *, must be in quotes to suppress shell interpretation
      - `find / -name '*.java'`
      - `find ~ -name "*.py"`
  - …

What do these commands do?

find utility (continued)

- **-perm** [+-]*mode*
  - Find files with given access mode, mode must be in octal.
  - Eg: **find . –perm 755**
- **-user** *userid/username*
  - Find by owner *userid* or *username*
- **-atime n**
  - File was last accessed n*24 hours ago. When find figures out how many 24-hour periods ago the file was last accessed, any fractional part is ignored
  - To match `-atime +1`, a file has to have been accessed at least two days ago.
- **-size** *size*
  - File size is at least *size*
- *many more…*

find: actions

- **-print** prints out the name of the current file (default)
- **-exec** *cmd*
  - Executes *cmd*, where *cmd* must be terminated by an escaped semicolon (`\;` or `';'`)
  - If you specify `{}` as a command line argument, it is replaced by the name of the current file just found
  - **exec** executes *cmd* once per file
  - Example: `find . –name "*~" -exec ls -l {} \;`

What does this command do?

find Examples

- Find all files beneath home directory beginning with .b
  - `find ~ -name '.b*'`
- Find all files beneath home directory modified within last 24 hours
  - `find ~ -mtime 0`
- Find all files beneath home directory larger than 10K
  - `find ~ -size 10k`
- Count words in files under home directory
  - `find ~ -exec wc -w {} \;`
- Remove core files
  - `find / -name core –exec rm {} \;`

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; then list the files in exec and verify want to remove those files

diff: comparing two files

- **diff**: compares two files and outputs a description of their differences
  - Usage: **diff** [*options*] *file1 file2*
  - **-i** : ignore case
  - **-u** : human readable
  - **-bB** : ignore white space

```
apples
oranges
walnuts
```

```
apples
oranges
grapes
```

```
$ diff list1 list2
3c3
< walnuts
---
> grapes
```


Other file comparison utilities

- cmp
  - Tests two files for equality
  - If equal, nothing returned. If different, location of first differing byte returned
  - Faster than **diff** for checking equality
- comm
  - Reads two files and outputs three columns:
    - Lines in first file only
    - Lines in second file only
    - Lines in both files
  - Must be sorted
  - Options: fields to suppress ([-123])

Looking Ahead

- Install Docker on your machine before Friday's class
- Assignment 1: out soon!