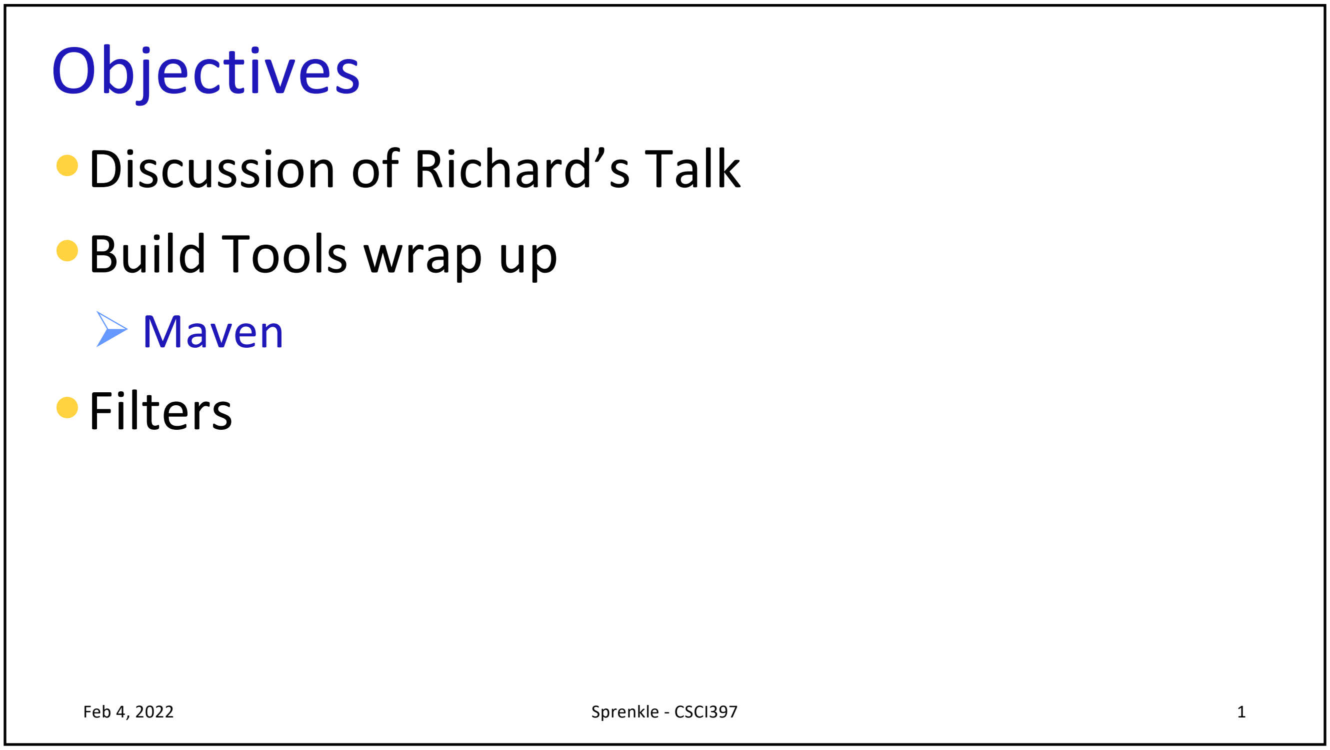
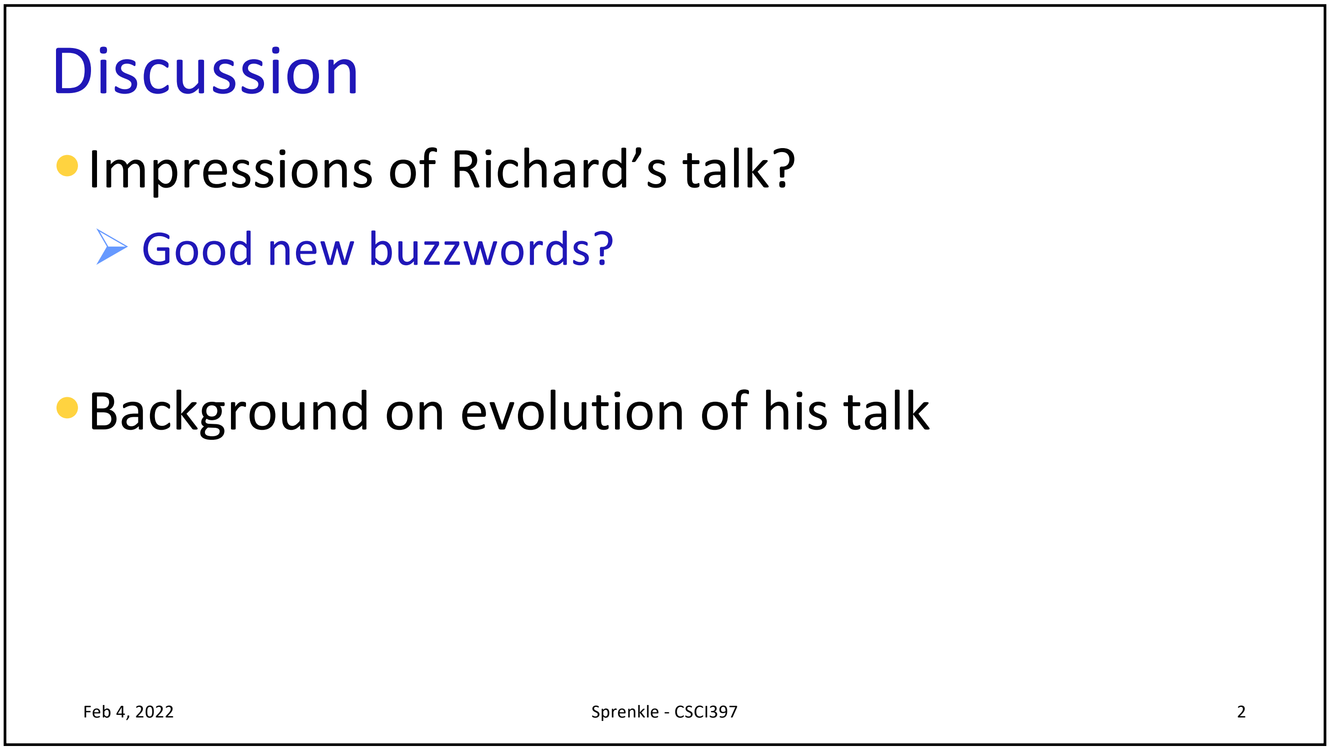


Objectives

- Discussion of Richard’s Talk
- Build Tools wrap up
  - Maven
- Filters

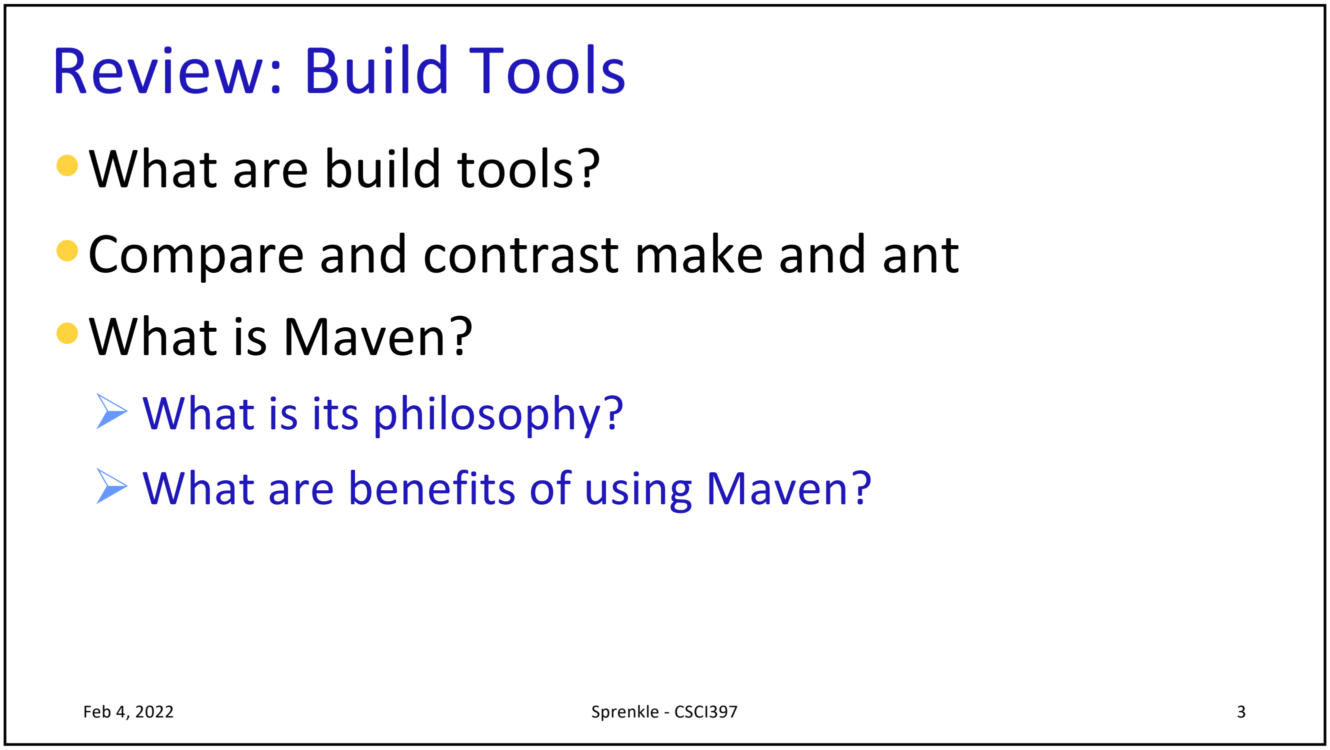
1



Discussion

- Impressions of Richard’s talk?
  - Good new buzzwords?
- Background on evolution of his talk

2



Review: Build Tools

- What are build tools?
- Compare and contrast make and ant
- What is Maven?
  - What is its philosophy?
  - What are benefits of using Maven?

3

Maven Repository

`https://mvnrepository.com/`

- How to use it
- Typically: looking for a stable release
  - rc = release candidate

4

Benefits of Using Maven

- Provides a common structure for organizing code
- Provides common plugins for various stages in the build process
- Makes it easier to build your code
  - Just need to organize in the required structure
- Makes it easier to understand others' code

Summary: Build Tools

- Automate process of building various artifacts from your source code
  - Examples: compile, distribute (jars), documentation, commercial_version, …
- Why is there more than one build tool?
- What are the similarities and differences between make, ant, and maven?

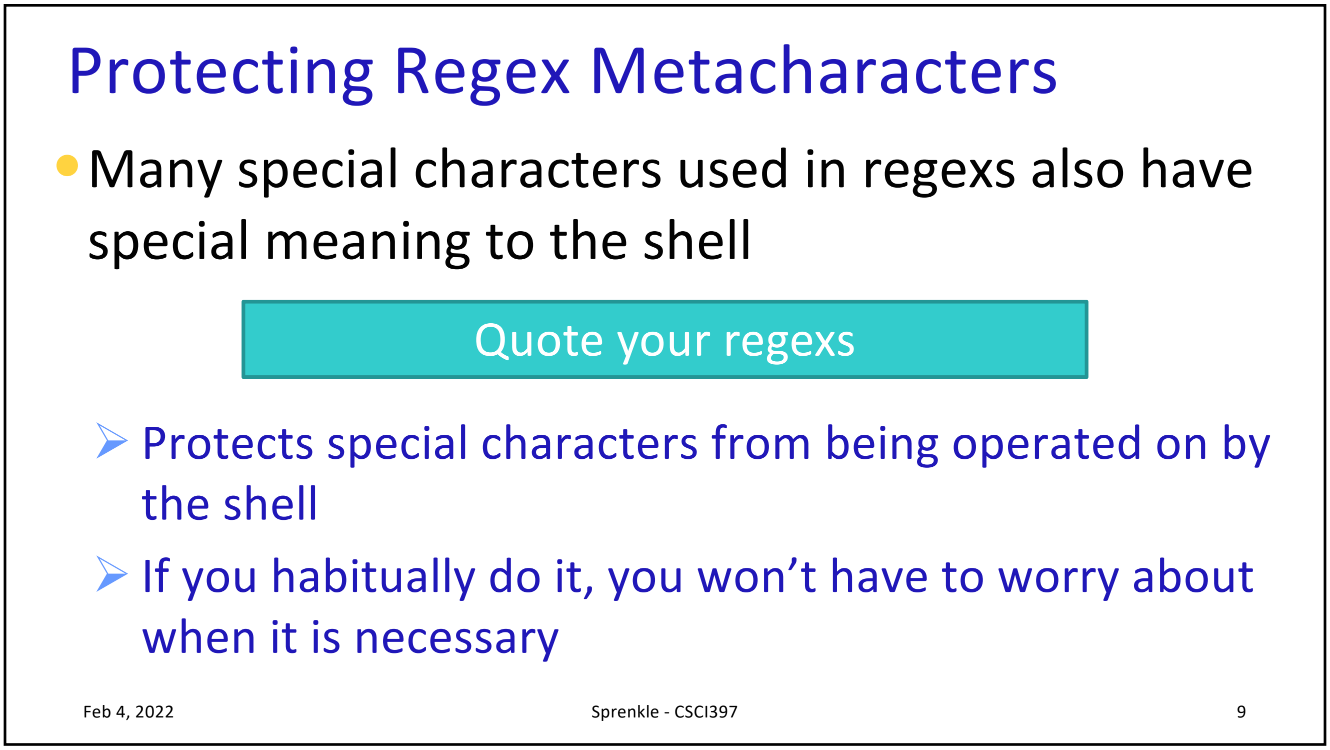
Running Discussion Questions

- Why does the tool exist? What is its purpose?
- What can the tool do?
- What can't the tool do?
  - Because it hasn't been done? Because of current technology limitations? Or some other limitations?
  - If because it hasn't been done, what can do to change that?

Review

- What are regular expressions?
- What is grep?
  - How do we use it?

Login to lab computer
Go to handouts directory

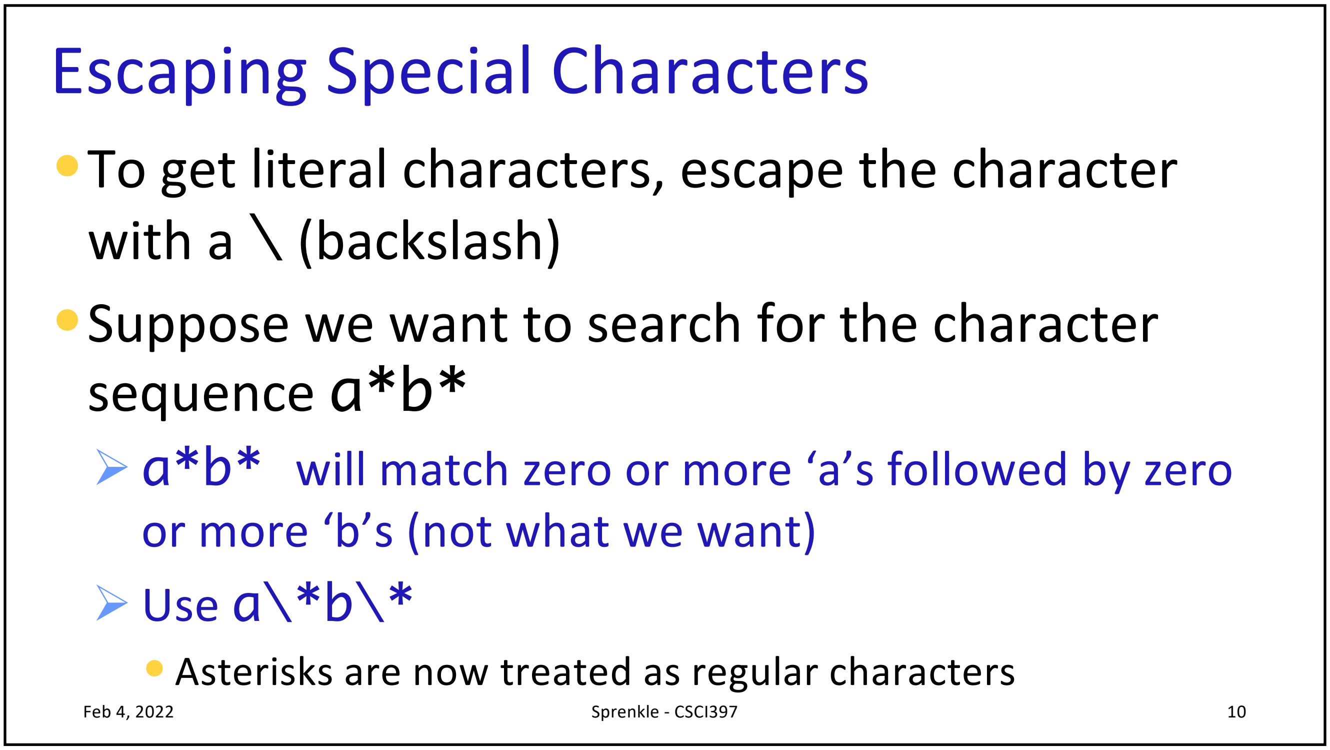


Protecting Regex Metacharacters

- Many special characters used in regexs also have special meaning to the shell

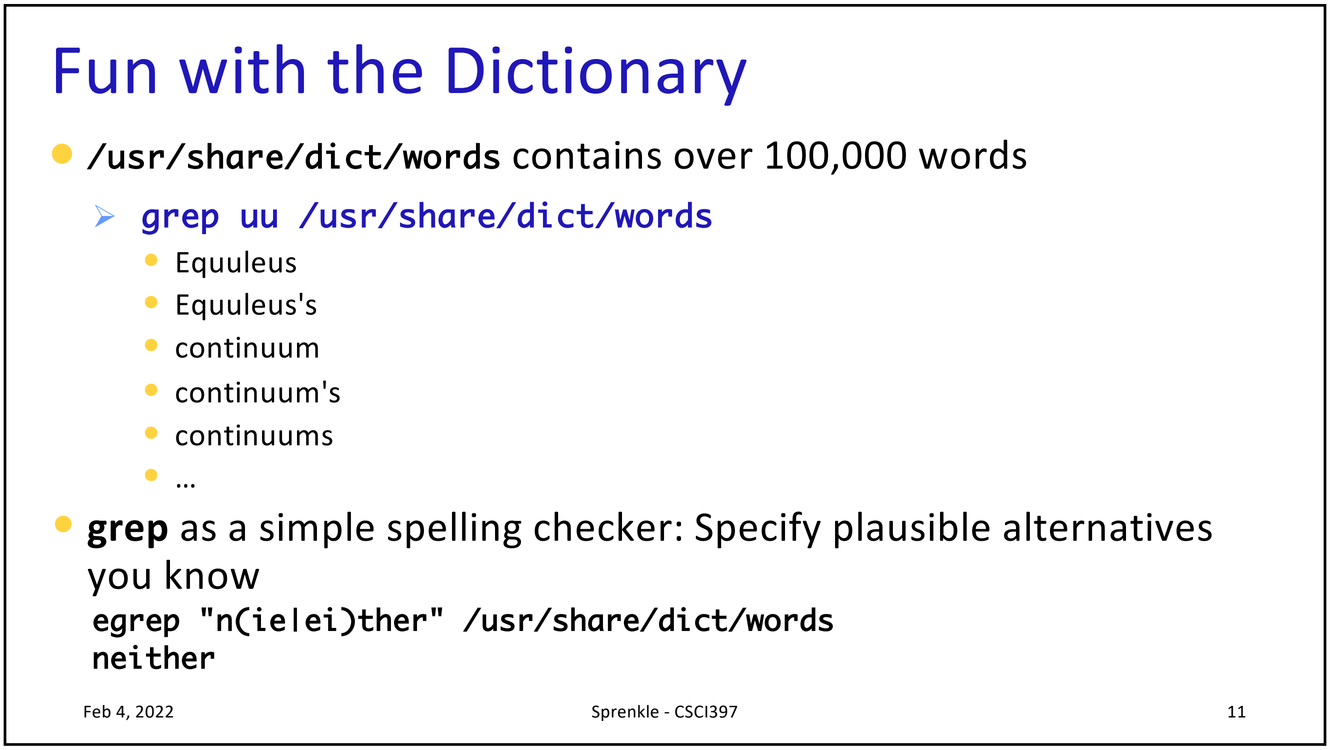
Quote your regexs

  - ➢ Protects special characters from being operated on by the shell
  - ➢ If you habitually do it, you won't have to worry about when it is necessary



Escaping Special Characters

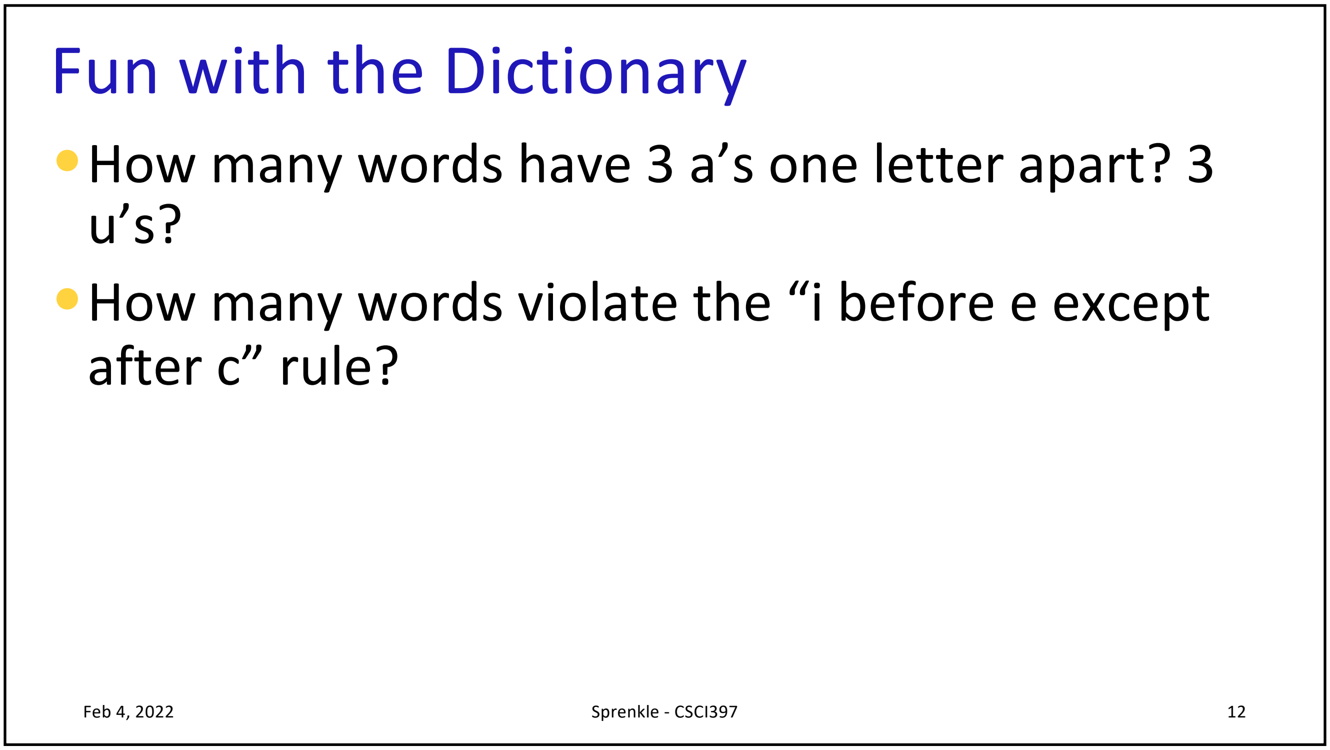
- To get literal characters, escape the character with a \ (backslash)
- Suppose we want to search for the character sequence a*b*
  - ➢ a*b* will match zero or more 'a's followed by zero or more 'b's (not what we want)
  - ➢ Use a*b*
    - Asterisks are now treated as regular characters



Fun with the Dictionary

- `/usr/share/dict/words` contains over 100,000 words
  - ➢ `grep uu /usr/share/dict/words`
    - Equuleus
    - Equuleus's
    - continuum
    - continuum's
    - continuums
    - …
- **grep** as a simple spelling checker: Specify plausible alternatives you know

```
egrep "n(ie|ei)ther" /usr/share/dict/words
neither
```



Fun with the Dictionary

- How many words have 3 a’s one letter apart? 3 u’s?
- How many words violate the “i before e except after c” rule?

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

FILTERS

Go to $HANDOUTS/pipes_examples

Getting Input: What’s the difference?

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

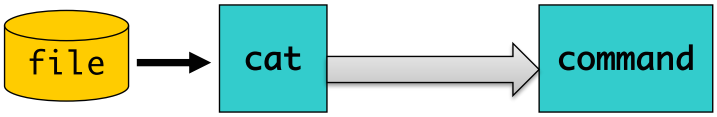
```
$ cat file | command
```

vs.

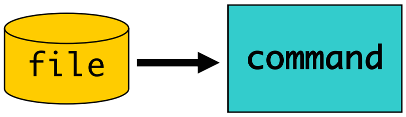
```
$ command < file
```

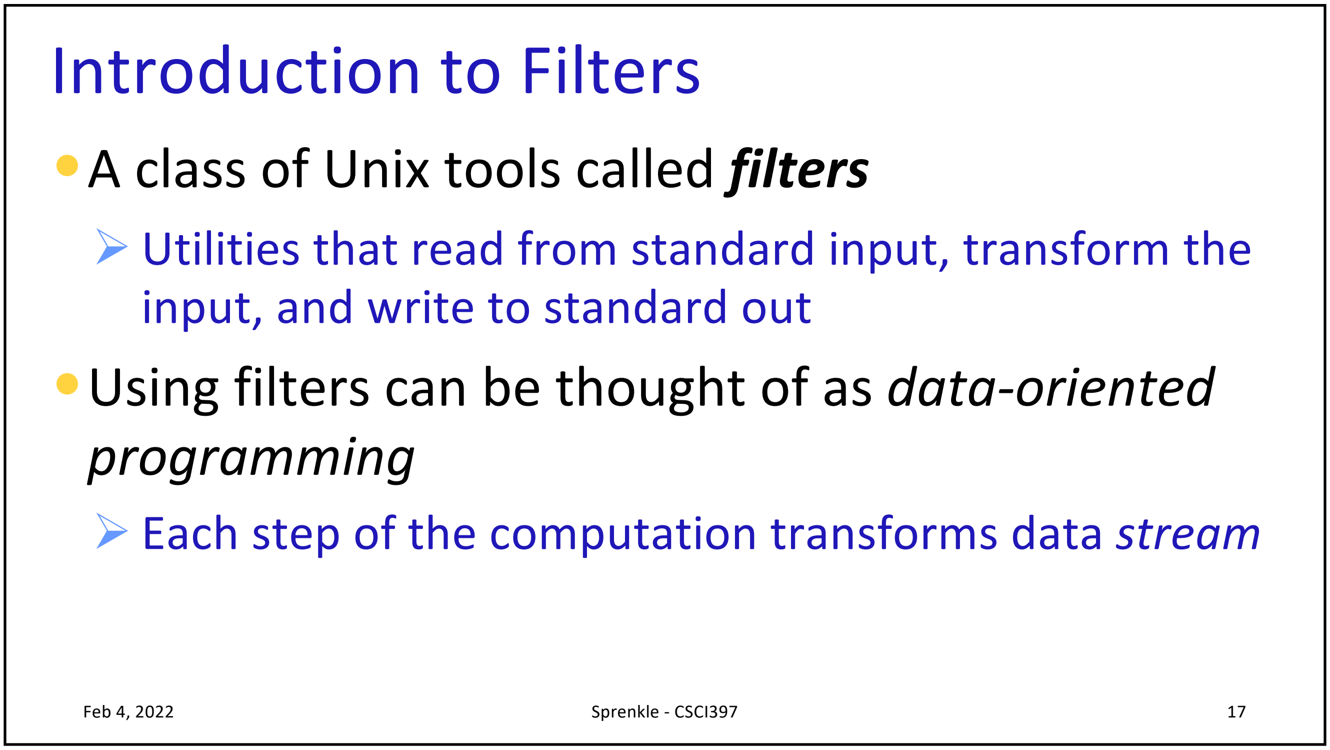
Difference: An Extra Process

```
$ cat file | command
```



```
$ command < file
```

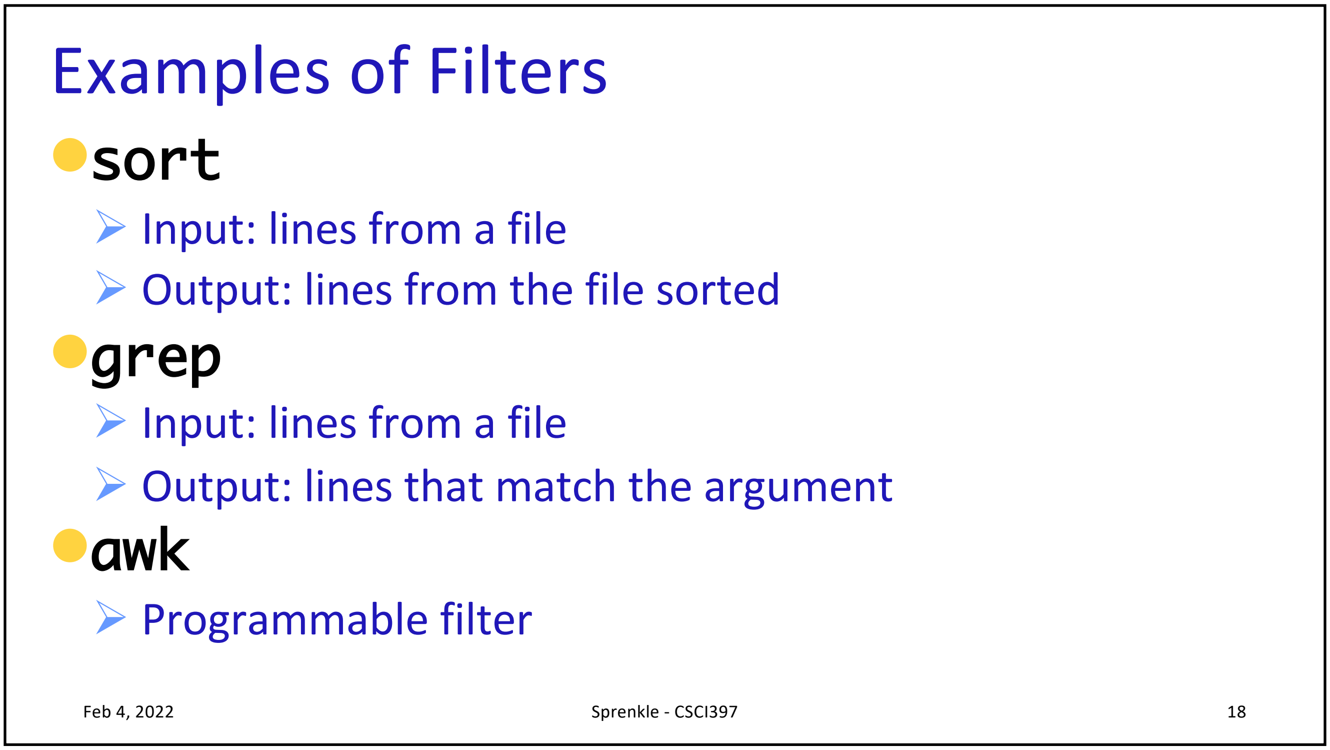




Introduction to Filters

- 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*

17

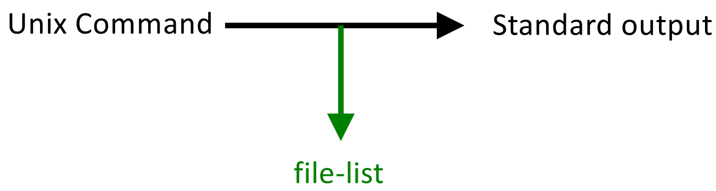


Examples of Filters

- `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

18

tee



Unix Command → Standard output

file-list

- Read from standard input
- 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

What is the end result of each command?

```
$ ls | head -10 | tee ~/first_10 | tail -5
$ who | tee ~/user_list | wc
```