CS2211b

Software Tools and Systems Programming

Western
UNIVERSITY · CANADA

Week 3b

Filters & Regular Expression

Announcements

Quiz Today!

Posted *emp.lst* and *shortlist* on OWL
(example files from CH9 and CH10 of textbook)

In-class Activity From Last Class

In-class Activity

Warmup activity don't have to hand in

Using pipes:

1. Write a command to count the number of files in the current directory and both display the result to the screen and save it in out.txt
2. Using the `hostname` command, write a command to replace any number in the `hostname` with the '#' character (e.g. cs2211b would become cs#b) and save the result in out.txt (do not display).
3. Write a command to display the unique lines in *readme.txt*. Remove . (single dots), brackets and numbers. Use `tr` and `sort`. **Hint:** you will need to make every word on it's own line.

In-class Activity

Warmup activity don't have to hand in

Using pipes:

1. Write a command to count the number of files in the current directory and both display the result to the screen and save it in out.txt

In-class Activity

Warmup activity don't have to hand in

Using pipes:

1. Write a command to count the number of files in the current directory and both display the result to the screen and save it in out.txt

```
ls | wc -l | tee out.txt
```

In-class Activity

Warmup activity don't have to hand in

Using pipes:

2. Using the `hostname` command write a command to replace any number in the `hostname` with the '#' character (e.g. cs2211b would become cs#b) and save the result in out.txt (do not display).

In-class Activity

Warmup activity don't have to hand in

Using pipes:

2. Using the `hostname` command write a command to replace any number in the `hostname` with the '#' character (e.g. cs2211b would become cs#b) and save the result in out.txt (do not display).

```
hostname | tr –s '[0-9]' '#' > out.txt
```

In-class Activity

Warmup activity don't have to hand in

Using pipes:

3. Write a command to display the unique lines in *readme.txt*. Remove . (single dots), brackets and numbers. Use `tr` and `sort`.

In-class Activity

Warmup activity don't have to hand in

Using pipes:

3. Write a command to display the unique lines in *readme.txt*. Remove . (single dots), brackets and numbers. Use `tr` and `sort`.

```
tr -s '[0-9]. ()\n' '\n' < readme.txt | sort -u
```

In-class Activity

Warmup activity don't have to hand in

Using pipes:

3. Write a command to display the unique lines in *readme.txt*. Remove . (single dots), brackets and numbers. Use `tr` and `sort`.

```
tr -s '[0-9]. ()\n' '\n' < readme.txt | sort -u
```

A lot we could improve

Remove all special chars

Make words all lowercase

In-class Activity

Warmup activity don't have to hand in

Using pipes:

3. Write a command to display the unique lines in *readme.txt*. Remove . (single dots), brackets and numbers. Use `tr` and `sort`.

```
tr '[A-Z]' '[a-z]' < readme.txt | tr -sc '[a-zA-Z]' '\n' | sort -u
```

First make everything lowercase

-c option is negation

SET1 is anything but [a-zA-Z]

/dev/null

- Some times you want to throw away your output and not display it to the screen or save it in a file.
- */dev/null* is a special file that is always empty.
- Any files saved to it or output sent to it is discarded.
- Examples:
  - Copy things to here and they disappear:

    ```
    cp myfile /dev/null
    ```

  - Copy from here and get an empty file:

    ```
    cp /dev/null myfile
    ```

  - Redirect error messages to this file and they are discarded:

    ```
    ls -l fileThatDoesNotExist 2> /dev/null
    ```

Filters

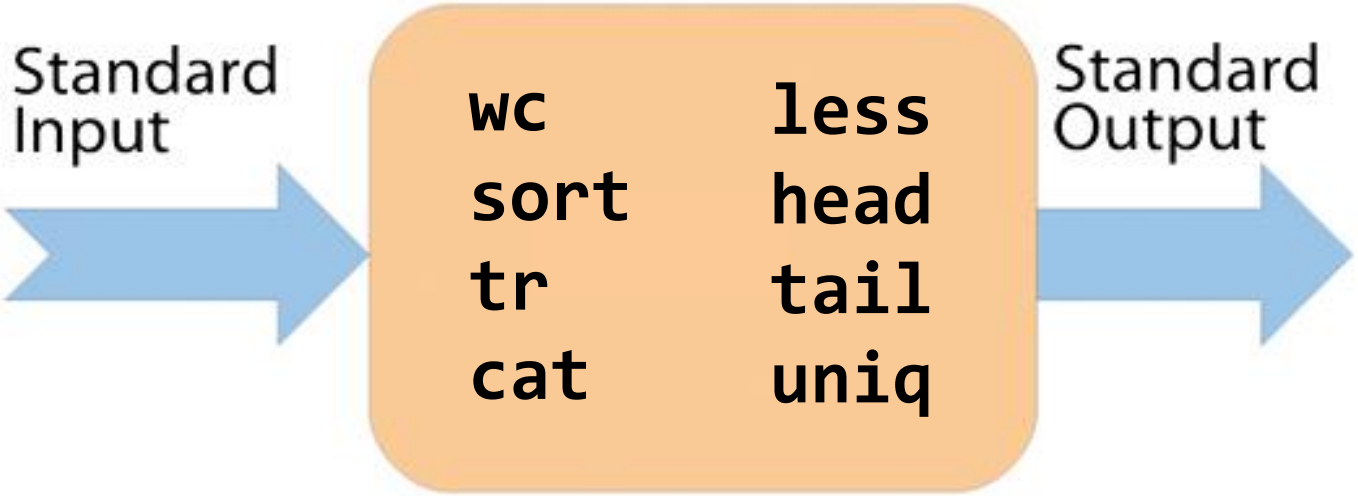
Filters

- A filter is a program that reads data from standard input, performs some operation on that data and then outputs the result to standard output.

Filter diagrams from: http://www.tuxradar.com/content/exploring-filters-and-pipes

Filters

- We have already encountered a number of filter programs including `wc`, `sort`, `tr`, `cat`, `less`, `head`, `tail`, and `uniq`.



Filters

Question

Is echo a filter?

Filters

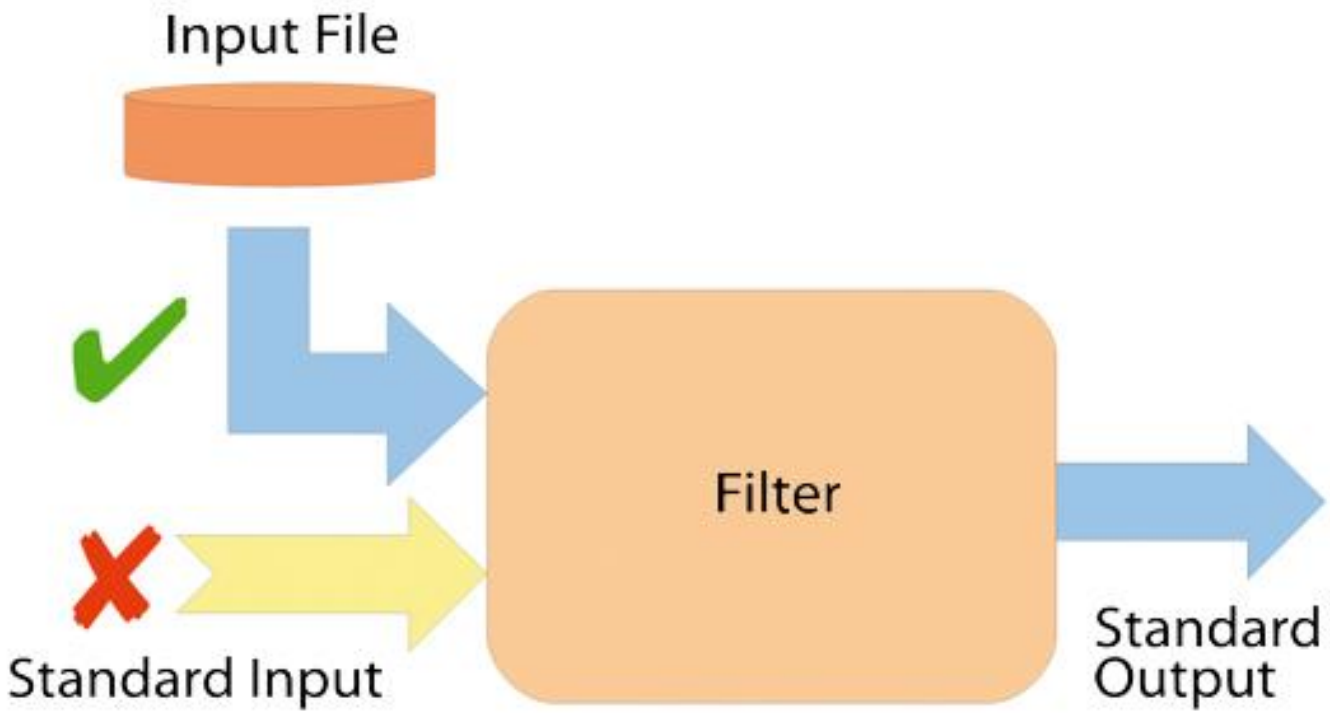
Question

Is echo a filter?

No, it does not take input from standard input

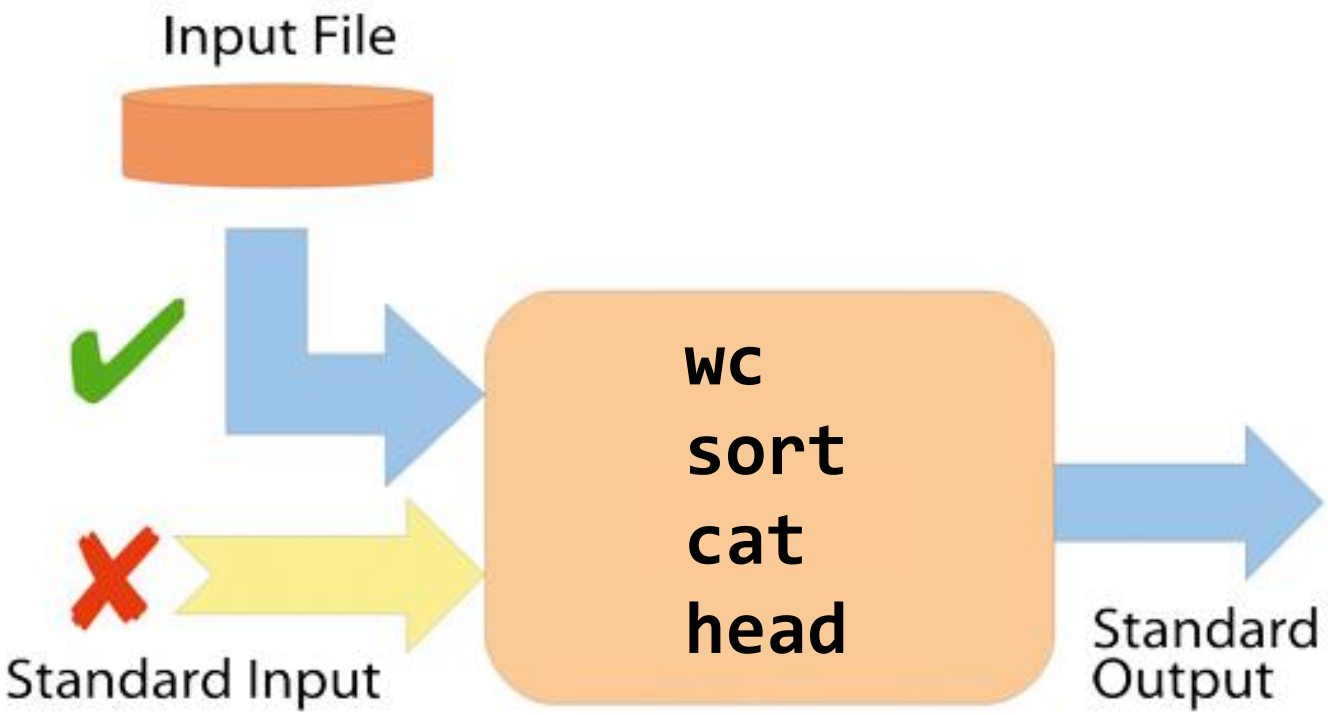
Filters

- Some (but not all) filters allow you to give one or more files as arguments. In this case, the file is used as a source of input rather than standard input.



Filters

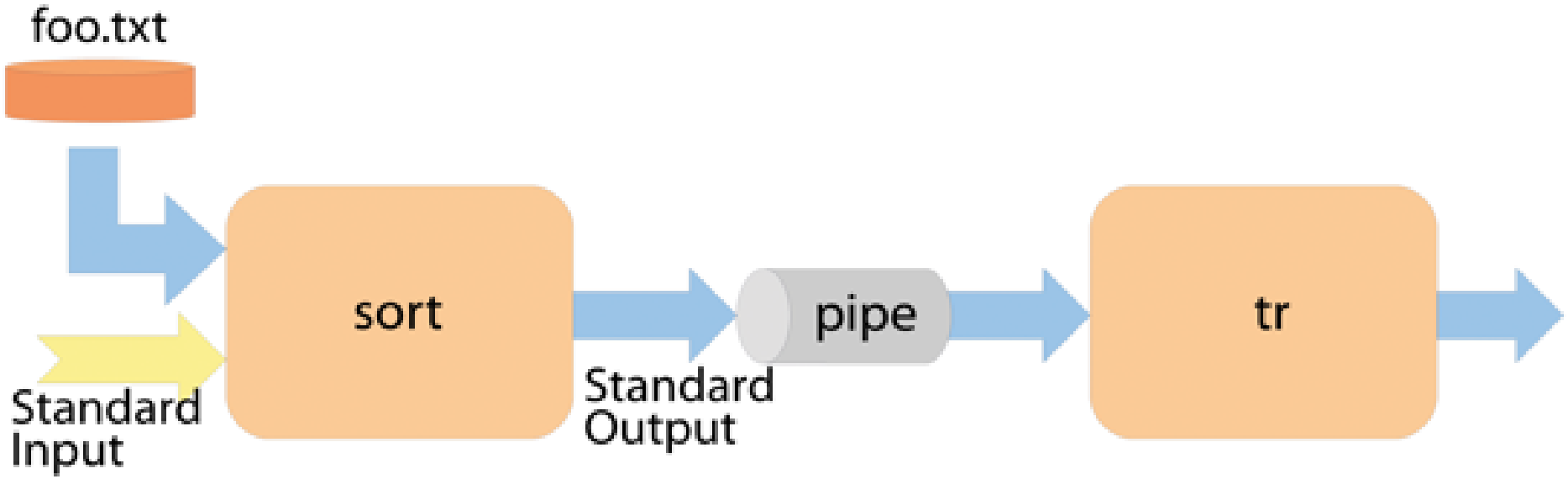
- For example, `wc`, `cat`, `sort` and `head` support this while `tr` does not.



Filters

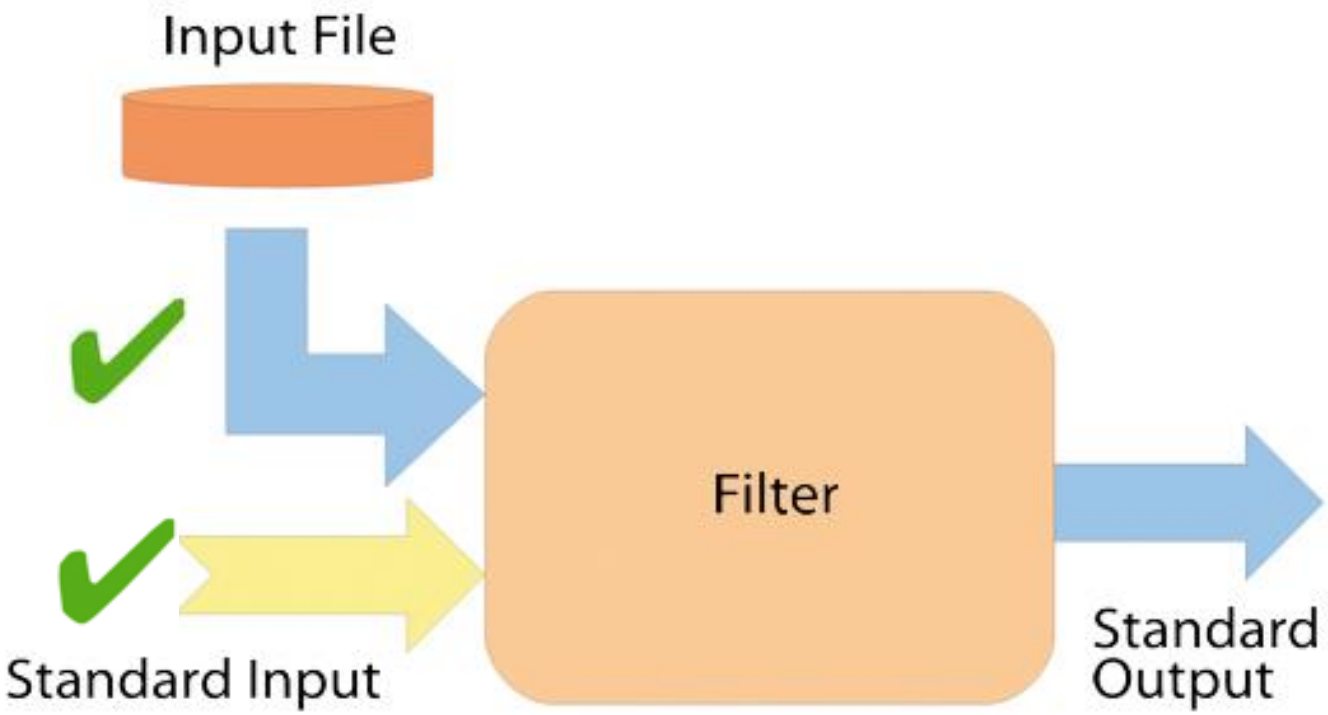
Example:

```
sort foo.txt | tr '[A-Z]' '[a-z]'
```



Filters

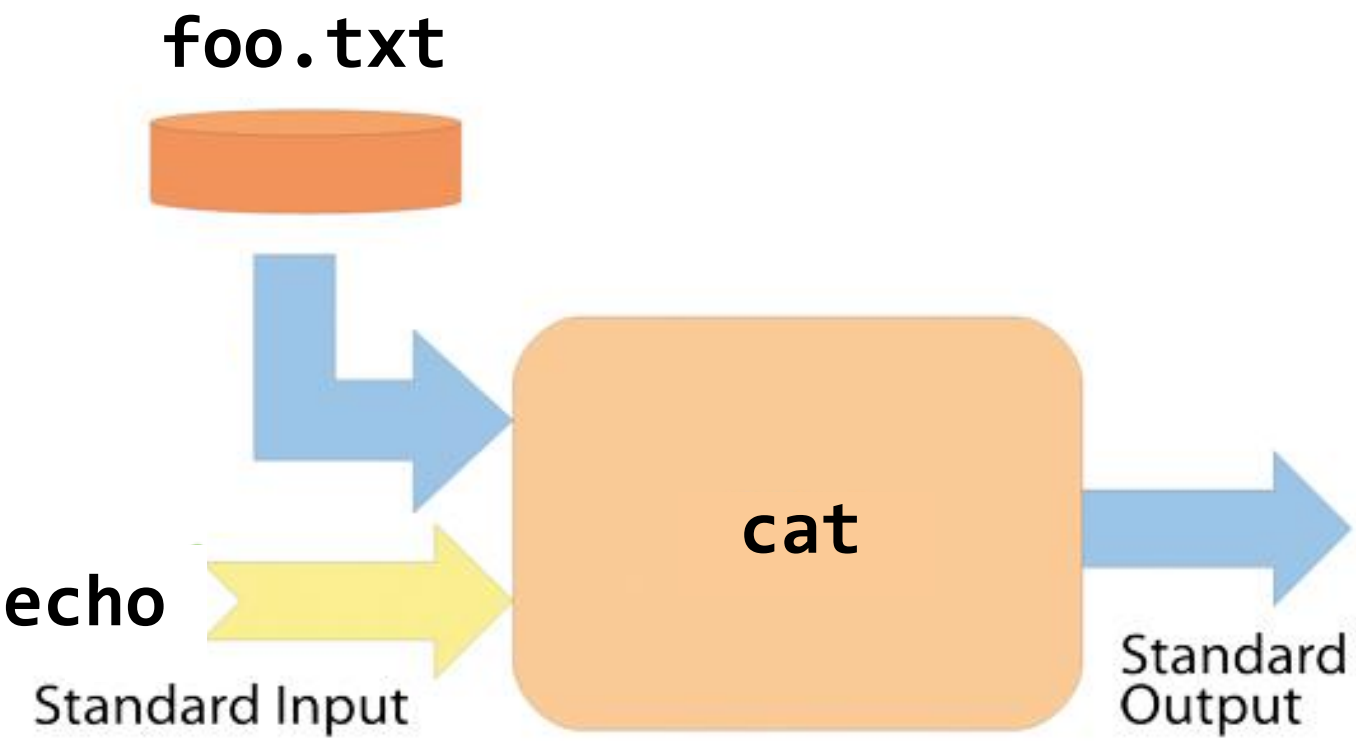
- Some filters support using - (a dash) as an argument to take input from both a file and standard input.



Filters

Example:

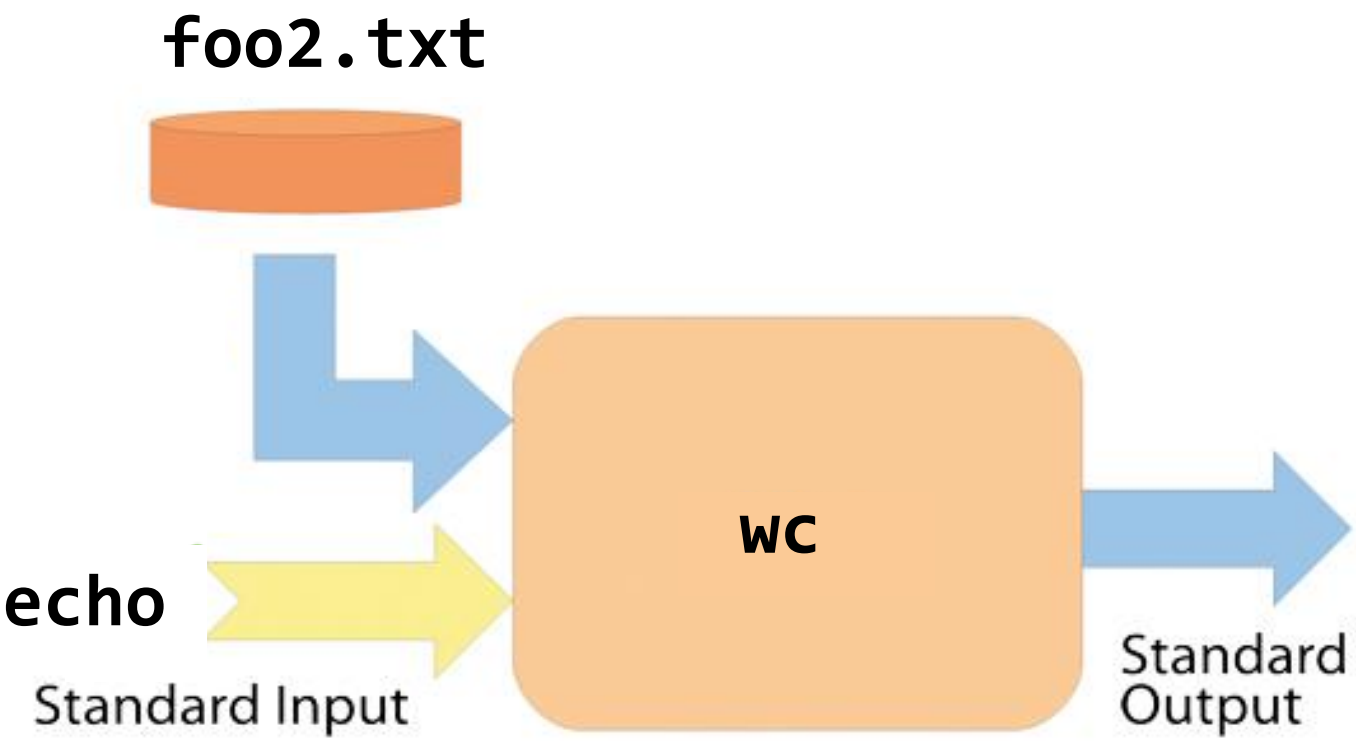
echo “hello world” | cat foo.txt -



Filters

Example:

echo “hello world” | wc -w - foo2.txt



Filters

- Chapter 9 and 10 give details on a number of filter programs we need to know about.

You should know how to use:

cat	diff
tail	comm
head	grep
less	tr
wc	uniq
sort	tee

Not covered in class.
See Chapter 9

Any filter discussed in a lab or lecture

You do not need to know the following but they are very useful:

sed	awk
pr	cmp
cut	paste
expand	fold
more	nl
split	tac

Covered in the textbook

`grep` &
Regular Expression

grep

- grep stands for *global regular expression and print*
- Allows us to search for text in a file using *regular expressions*.
- Can also take input via standard input rather than a file.
- Basic syntax:

```
grep [options] pattern_string [filename]
```

grep

- grep stands for *global regular expression and print*
- Allows us to search for text in a file using *regular expressions*.
- Can also take input via standard input rather than a file.
- Basic syntax:

grep [options] pattern_string [filename …]

Regular expression

File(s) to search
If no file is given will search input from standard in

grep

Simple Examples

- Find all of the lines in *textbook.txt* that contain the word UNIX:

```
grep UNIX textbook.txt
```

- Find only show entries in who for the user dservos5:

```
who | grep dservos5
```

- Count the number of ordinary files in the CWD with the permissions rw-------:

```
ls -l | grep '\-rw-------' | wc -l
```

grep

Simple Examples

- Find all of the lines in *textbook.txt* that contain the word UNIX:

```
grep UNIX textbook.txt
```

- Find only show entries in who for the user dservos5:

```
who | grep dservos5
```

- Count the number of ordinary files in the CWD with the permissions rw-------:

Need to escape first - so grep does not think it is an option

Will not work if there is a file named -rw-------

```
ls -l | grep '\-rw-------' | wc -l
```

grep

grep Options

- Some useful options grep takes:

-i	Ignore case (capitalization) in patterns.
-v	Inverse match. Match everything but the given pattern.
-w	Only match whole words (not parts of words).
-x	Only match whole lines (not parts of lines or words).
-c	Output the number of matches instead of normal output.
-r	Recursive, search all files in a directory (and its directories).
-E	Interpret pattern as an extended regular expression (use slightly different syntax).

grep

grep Options

- Some useful options grep takes:

-i	Ignore case (capitalization) in patterns.
-v	Inverse match. Match everything but the given pattern.
-w	Only match whole words (not parts of words).
-x	Only match whole lines (not parts of lines or words).
-c	Output the number of matches instead of normal output.
-r	Recursive, search all files in a directory (and its directories).

ression (use

Last example could just be:

```
ls -l | grep -c '\-rw-------'
```

Regular Expressions

- Real power of grep command comes from using it with regular expressions.
- Regular expressions are a language to describe text patterns.
- Basically, a regular expression is a pattern describing a certain amount of text.
- Used in all sorts of editors and programming tools.
- Used by the commands like: `grep`, `awk` and `sed` as well as text editors like: `vi`, `vim`, `emacs` and `nano`.
- Syntax is not always compatible between different programs (some use slightly different syntax or modes of operation).

Regular Expressions

- Regular expressions are ***different*** from file name wildcards:
  - File name ***wildcards*** are interpreted and matched by the ***shell***
  - ***Regular expressions*** are interpreted and matched by the ***command*** or ***program***
  - Regular expressions have different syntax and meaning for metacharacters (*, ?, +, etc.).
  - Some similarities:
    - Both use character classes ([a-z], [0-9], [abc123], etc).
    - Both use \ to escape characters.

Regular Expressions

- It is very important to know when you are using a shell wildcard and when you are giving regular expression to grep as an argument

```
grep cat* myfile
```

```
grep cat? myfile
```

VS.

```
grep 'cat*' myfile
```

```
grep 'cat?' myfile
```

Regular Expressions

- It is ***very important*** to know when you are using a shell wildcard and when you are giving regular expression to grep as an argument

```
grep cat* myfile
```

Shell wildcard
grep would get all files in current directory starting with “cat” as the first argument
Likely not what we want

VS.

```
grep 'cat*' myfile
```

Literal String
grep would get the string “cat*” as the first argument.
What we want

Regular Expressions

- It is ***very important*** to know when you are using a shell wildcard and when you are giving regular expression to grep as an argument

Shell wildcard
grep would get all filenames in current directory of length 4 and starting with the word "cat"
Likely not what we want

```
grep cat? myfile
```

VS.

```
grep 'cat?' myfile
```

Literal String
grep would get the string "cat?" as the first argument.
What we want

Regular Expressions

- It is ***very important*** to know when you are using a shell wildcard and when you are giving regular expression to grep as an argument

Shell wildcard

Recommended that you always surround your grep patterns with 's to escape them.

e

Literal String

grep would get the string “cat?”
as the first argument.

What we want

Regular Expressions

Definitions

- A **literal** is any character we use in a search or matching expression.
  - e.g., to find **ind** in w**ind**ows the **ind** is a **literal** string - each character plays a part in the search, it is **literally** the string we want to find.

Regular Expressions

Definitions

- A **metacharacter** is one or more special characters that have a unique meaning and are NOT used as **literals** in the search expression.
  - Examples: \ ^ $. | ? * + ( ) [ ]

- An **escape sequence** is a way of indicating that we want to use one of our **metacharacters** as a **literal**.
  - *e .g.,* \\ matches \ ,
  - \. matches periods only.

Regular Expressions

Literals

- Letters and Numbers
  - Ordinary characters are matched literally
    - **a** matches a, **b** matches b, **1** matches 1
- Example:
  - Letters:

Pattern:	**xyz**			
Strings:	xyz	xyzz	xxyz	abcxyz132
Match:	xyz	xyz	xyz	xyz

  - Numbers:

Pattern:	**123**			
Strings:	123	1234	0123	321123xyz
Match:	123	123	123	123

Regular Expressions

Character Classes

Sometimes we need more structure than the dot metacharacter provides, we may need to match any character in a set or "character classes".

- [xyz] matches only a single x, y, or z (all the listed characters between the brackets)
- **Example:** Match bat, cat, but not fat or pat

Pattern:	**[bc]at**				
Strings:	bat	cat	fat	pat	bobcat
Match:	bat	cat			cat

Regular Expressions

Character Classes

^ at the start of a character class excludes any characters in the class (matches characters not listed between the square brackets)

- [^xyz] matches any character other than x, y or z (including numbers, spaces, etc.)
- **Example:** Match all three letter words that end in at except bat or cat

Pattern:	**[^bc]at**				
Strings:	bat	cat	fat	Pat's cat	23atb$at
Match:			fat	Pat	3at & $at

Regular Expressions

Character Classes

We can specify a range of characters range than listing each individually.

- [x-y] matches any character between x and y alphabetically (including x and y).
- **Examples:**

[0-9]	Matches any digit.
[a-z]	Matches any lower case letter.
[A-Z]	Matches any upper case letter.
[B-E]	Matches upper case letters between B and E (i.e. B, C, D, and E).

Regular Expressions

Character Classes

We can specify a range of characters range than listing each individually.

- [x-y] matches [illegible] x and y alphabetically
- **Examples:**

Unlike with wildcards, the capitalization here is respected by grep.

[0-9]	Matches any digit.
[a-z]	Matches any lower case letter.
[A-Z]	Matches any upper case letter.
[B-E]	Matches upper case letters between B and E (i.e. B, C, D, and E).

Regular Expressions

Character Classes

We can combine character classes to make more complex matches.

- **Examples:**

[a-zA-Z]	Matches any letter (upper or lower case).
[a-zA-Z0-9]	Matches any letter or digit.
[a-z7!]	Matches any lower case letter, the number 7 or the ! character.
[abc0-9XYZ]	Matches any digit and the letters a, b, c, X, Y, and Z

Regular Expressions

Character Classes

If we need to match the literal character [or], we need to escape it with \

- **Example:** Match a single digit between two square brackets:

Pattern:	**\[[0-9]\]**				
Strings:	[5]	[0]	[12]	5	See citation [3] or [5] in Chapter 3.
Match:	[5]	[0]			[3] & [5]

Regular Expressions

Wildcard

- Wildcard
  - . Match any single character *(letter, digit, whitespace, everything)*
  - To match the period you need to use the escape sequence \.
- **Example:** Match any string of length 3

Pattern:	...				
Strings:	abc	123	1b!	c4	C 8
Match:	abc	123	1b!		C 8

Regular Expressions

Wildcard

- Wildcard
  - . Match any single character *(letter, digit, whitespace, everything)*
  - To match the period you need to use the escape sequence \.
- **Example:** Match a one digit number with a single decimal place (e.g. 1.4, 9.0, 2.7, etc.).

Pattern:	**[0-9]\.[0-9]**				
Strings:	1.3	12.34	c.45	3..5	9.cat
Match:	1.3	2.3			

Regular Expressions

Wildcard

- Wildcard
  - . Match any single character *(letter, digit, whitespace, everything)*
  - To match the period you need to use the escape sequence \.
- **Example:** Match any three letter word starting with a ‘c’ and ending with a ‘t’. Can have numbers, and other characters.

Pattern:	**c.t**				
Strings:	cat	bat	c9tz	Hic.t!	caatc top
Match:	cat		c9t	c.t	c t

Regular Expressions

Repetitions

We can match a character multiple times using {min, max} notation.

- c{n} matches the character c, n times
- c{min,} matches c, min or more times
- c{min, max} matches the character c between min and max times
- **Example:** Match a 5 letter word ending in aaat, such as Baaat, caaat or Faaat

Pattern:	**[a-zA-Z]a{3}t**				
Strings:	Baaat	zaaat	Naat	gaaaat	AAaaattt
Match:	Baaat	zaaat		aaaat	Aaaat

Regular Expressions

Repetitions

We can match a characte
notation.

- c{n} ma
- c{min,} ma
- c{min, max} ma
  mi

When using grep we need to use \ in front of the {s to denote that they are metacharacters. Otherwise they match a literal { or }.

For example:

```
grep '[a-zA-Z]a\{3\}t' myfile
```

- **Example:** Match a 5 letter word ending in aaat, such as Baaat, caaat or Faaat

Pattern:	**[a-zA-Z]a{3}t**				
Strings:	Baaat	zaaat	Naat	gaaaat	AAaaattt
Match:	Baaat	zaaat		aaaat	Aaaat

Regular Expressions

Repetitions

We can match a charac
notation.

- c{n} r
- c{min,} r
- c{min, max} matches the character c between min and max times

We could also use the -E option to tell grep to use extended regular expression:

For example:

```
grep -E '[a-zA-Z]a{3}t' myfile
```

- **Example:** Match a 5 letter word ending in aaat, such as Baaat, caaat or Faaat

Pattern:	**[a-zA-Z]a{3}t**				
Strings:	Baaat	zaaat	Naat	gaaaat	AAaaattt
Match:	Baaat	zaaat		aaaat	Aaaat

Regular Expressions

Repetitions

We can match a character multiple times using {min, max} notation.

- c{n} matches the character c, n times
- c{min,} matches c, min or more times
- c{min, max} matches the character c between min and max times

- **Example:** Match a three to five lower case vowels in a row.

Pattern:	**[aeiou]{3,5}**				
Strings:	caat	baaat	aaiie	ouou	aeioua
Match:		aaa	aaiie	ouou	aeiou

Regular Expressions

Repetitions

We can match a character multiple times using {min, max}

When used with grep:

```
grep '[aeiou]\{3,5\}' myfile
```

OR

```
grep -E '[aeiou]{3,5}' myfile
```

r c, n times

re times

r c between

- **Example:** Match a three to five lower case vowels in a row.

Pattern:	**[aeiou]{3,5}**				
Strings:	caat	baaat	aaiie	ouou	aeioua
Match:		aaa	aaiie	ouou	aeiou

Regular Expressions

In-class Activity

Write a regular expression that

Matches cat? Hat. Bat# 9at- !at(

But does not match catt catZ cAt? Bet# 9aT-

Write a regular expression to match the phone number pattern ###-###-#### where # can be any number between 0 and 9

Regular Expressions

In-class Activity

Write a regular expression that

Matches cat? Hat. Bat# 9at- !at(

But does not match catt catZ cAt? Bet# 9aT-

.at[^0-9a-zA-Z]

Write a regular expression to match the phone number pattern ###-###-#### where # can be any number between 0 and 9

Regular Expressions

In-class Activity

Write a regular expression that

Matches cat? Hat. Bat# 9at- !at(

But does not match catt catZ cAt? Bet# 9aT-

.at[^0-9a-zA-Z]

Write a regular expression to match the phone number pattern ###-###-#### where # can be any number between 0 and 9

[0-9]{3}-[0-9]{3}-[0-9]{4}

Regular Expressions

Kleene Star and Plus Notation

- c? matches the character c zero or one times
- c* matches the character c zero or more times
- c+ matches the character c one or more times

Examples:

[0-9]+ matches a digit one or more times (9, 12, 343, …)

[a-z]* matches the lower case letters zero or more times. Includes an empty string (zero matches) or things like (a, ab, cat, jess)

xy?z matches xyz or xz

c.?t matches a word starting with c and ending with t that has any or no character between (ct, cat, czt, c6t, c%t)

Regular Expressions

Kleene Star and

- c? matches th
- c* matches th
- c+ matches th

Like with {}s, we need to use a \ in front of + and ? to denote it as a metacharacter.

For example:

```
grep '[0-9]\+' myfile
grep 'xy\?z' myfile
```

Examples:

[0-9]+ matches a

[a-z]* matches the lower case letters zero or more times. Includes an empty string (zero matches) or things like (a, ab, cat, jess)

xy?z matches xyz or xz

c.?t matches a word starting with c and ending with t that has any or no character between (ct, cat, czt, c6t, c%t)

Regular Expressions

Kleene Star and

- c? matches th
- c* matches th
- c+ matches th

Or we can use the -E option.

For example:

```
grep -E '[0-9]+' myfile
grep -E 'xy?z' myfile
```

Examples:

[0-9]+ matches a digit one or more times (9, 12, 345, …)

[a-z]* matches the lower case letters zero or more times. Includes an empty string (zero matches) or things like (a, ab, cat, jess)

xy?z matches xyz or xz

c.?t matches a word starting with c and ending with t that has any or no character between (ct, cat, czt, c6t, c%t)

Regular Expressions

Anchors

- Anchors allow us to match the start or end of a line.
- Denoted with special metacharacters ^ (caret) and $
  - ^ matches the beginning of a line (except when used at the start of a character class like [^…])
  - $ matches the end of a line

Example: Match lines that only have lowercase letters.

Pattern:	**^[a-z]+$**				
Strings:	cats	5dog8cat	a space	nospace	multiple lines of text
Match:	cats			nospace	multiple & text

Regular Expressions

Anchors

- Anchors allow us to match the start or end of a line.
- Denoted with special metacharacters ^ (caret) and $
  - ^ matches the beginning of a line (except when used at the start of a character class like [^…])
  - $ matches the end of a line

Example: Match lines that start with "Hello".

Pattern:	**^Hello.***				
Strings:	Hello World	My Hello	hello Bob	Hello	Hello123!!!
Match:	Hello World			Hello	Hello123!!!

Regular Expressions

Anchors

- Anchors allow us to match the start or end of a line.
- Denoted with special metacharacters ^ (caret) and $
  - ^ matches the beginning of a line (except when used at the start of a character class like [^…])
  - $ matches the end of a line

Example: Using `grep` count the number of ordinary files in the CWD with the permissions rw-------:

Old Answer: `ls -l | grep -c '\-rw-------'`

How do we fix it so a file named -rw------- will not throw off our count?

Regular Expressions

Anchors

- Anchors allow us to match the start or end of a line.
- Denoted with special metacharacters ^ (caret) and $
  - ^ matches the beginning of a line (except when used at the start of a character class like [^…])
  - $ matches the end of a line

Example: Using `grep` count the number of ordinary files in the CWD with the permissions rw-------:

```
ls -l | grep -c '^-rw-------'
```

Match the start of the line

Regular Expressions

Groups

- Sometimes need to reference a match we made previously.
- Groups allow us to "remember" what we matched.
  - (…) Where … is some pattern, class or character.
  - \n Reference grouping number n, where n is a number.

Example: Match a lower case word that starts with and ends with the same letter.

Pattern:	**([a-z])[a-z]*\1**					
Strings:	dad	health	soda	cc	pops	citation
Match:	dad	health		cc	pop	itati

Regular Expressions

Groups

- Sometimes need to
- Groups allow us to '
  - (…) Where .
  - \n Referen number

With grep we need to use \ in front of the ()s or the -E option:

```
grep '\([a-z]\)[a-z]*\1' myfile
```

OR

```
grep -E '([a-z])[a-z]*\1' myfile
```

Example: Match a lower case word that starts with and ends with the same letter.

Pattern:	**([a-z])[a-z]*\1**					
Strings:	dad	health	soda	cc	pops	citation
Match:	dad	health		cc	pop	itati

Regular Expressions

Groups

-
-

How would we match only whole words?

number.

Example: Match a lower case word that starts with and ends with the same letter.

Pattern:	**([a-z])[a-z]*\1**					
Strings:	dad	health	soda	cc	pops	citation
Match:	dad	health		cc	pop	itati

Regular Expressions

Groups

- **How would we match only whole words?**
- **A few ways:**

`^([a-z])[a-z]*\1$` -w option to grep

-x option to grep `\b([a-z])[a-z]*\1\b`

Example: Match a lower case word that starts with and ends with the same letter.

Pattern:	**([a-z])[a-z]*\1**					
Strings:	dad	health	soda	cc	pops	citation
Match:	dad	health		cc	pop	itati

Regular Expressions

Groups

- **How would we match only whole words?**
- Only matches if word is only text on line

```
^([a-z])[a-z]*\1$
```

-x option to grep

Matches any whole word

-w option to grep

```
\b([a-z])[a-z]*\1\b
```

Example: Match a lower case word that starts with and ends with the same letter.

Pattern:	**([a-z])[a-z]*\1**					
Strings:	dad	health	soda	cc	pops	citation
Match:	dad	health		cc	pop	itati

Regular Expressions

Groups

- **How would we match only whole words?**
- Only matches if word is only text on line

```
^([a-z])[a-z]*\1$
```

-x option to grep

Matches any whole word

-w option to grep

```
\b([a-z])[a-z]*\1\b
```

Example: Match a lower case word that starts with and ends with the same letter.

Matches word boundaries

Pattern:	**([a-z])[a-z]*\1**					
Strings:	dad	health	soda	cc	pops	citation
Match:	dad	health		cc	pop	itati

Regular Expressions

Groups

- Sometimes need to reference a match we made previously.
- Groups allow us to "remember" what we matched.
  - (…) Where … is some pattern, class or character.
  - \n Reference grouping number n, where n is a number.

Example: Match a line of characters that begins with a single digit and a lower case letter and ends with the same lower case letter followed by the same digit.

Pattern:	**^([0-9])([a-z]).*\2\1$**					
Strings:	1aa1	9j$Dx2j9	2bb2q	Dog 0zcz0	2d cat d2	1bb2
Match:	1aa1	9j$Dx2j9			2d cat d2	

Regular Expressions

Conditional

- Match one pattern OR another pattern.
  - patt1|patt2 matches the pattern patt1 or patt2.

Examples:

cat|dog	matches the word cat or the word dog
[0-9]|[a-z]	matches a digit or a lower case letter
b[a-z]t|c[a-z]t	matches words like bat, bit, bet, cat, cit, and czt

Regular Expressions

Conditional

- Match one patte
  - patt1|patt2

With grep we need to use \ in front of the | or the -E option:

```
grep 'cat\|dog' myfile
OR
grep -E 'cat|dog' myfile
```

Examples:

cat|dog	matches the word cat or the word dog
[0-9]|[a-z]	matches a digit or a lower case letter
b[a-z]t|c[a-z]t	matches words like bat, bit, bet, cat, cit, and czt

Regular Expressions

More Characters and Classes

- **\b, \B** is/is not a word boundary
- **\w, \W** is/is not a word character ([a-zA-Z0-9])
- **\s, \S** is/is not a space character (space, tab, newline, etc).
- **\<, \>** matches start of word (\<) and end of word (\>).

Have to be used in []s, for example [[:alnum:]]

- **[:alnum:]** same as \w
- **[:alpha:]** upper and lowercase letters
- **[:blank:]** space and tab
- **[:digit:]** digits ([0-9])
- **[:lower:]** lower case letters ([a-z])
- **[:upper:]** upper case letters ([A-Z])
- **[:space:]** any space character (space, tab, newline, etc).
- **[:xdigit:]** Hexadecimal digits ([0-9a-f])

More available depending on tool and options used.

Regular Expressions

Tutorials and other Resources

- Notepad++ Regular Expressions Tutorial
- RegexOne.com: Interactive Tutorial
- RegExr.com: Online Tool for Testing RegEx
- tutorialzin: Learn Regular Expressions in 20 Minutes
- Learn regular expressions in about 55 minutes (has practice exercise)

Some of these might use slightly different syntax and support slightly different character classes than `grep`

Regular Expressions

Tutorials and other Resources

- Notepad++ Regular Expressions Tutorial
- RegexOne.com: Interactive Tutorial
- RegExr.com: Online Tool for Testing RegEx
- tutorialzin: L Minutes
- Learn regula es (has practice exercise)

Helpful for debugging your regular expressions visually.

Some of these might use slightly different syntax and support slightly different character classes than grep