210 Systems Programming
Pipes, text processing, `grep`

Fall 2025

Week 3

Overview

- Piping
- `wc`, `head`, `tail`, `tr`, `sort`, and `uniq` commands
- The `grep` command
    -
    - Standard `grep`
    - Extended regular expressions `egrep`

A Better Way to connect I/O: Pipes

A mechanism to channel standard input and standard output between programs.

- Syntax: `s1 | s2 | s3 | ...`
- Chain together multiple commands/programs
- Data flows left to right
- The first program gets input from the standard input, the last program writes to standard output
- The standard output of the first program becomes the standard input of the second program
- ...

Text Processing with Pipes

- There are a suite of commands made especially for piping
- We'll go through a few in this lecture and a very important one (`grep`) next lecture

head and tail

`head [-n number] file`

Prints the first `number` lines of a file. By default the first 10 lines are displayed.

`tail [-n number] file`

Prints the last `number` lines of a file. By default the last 10 lines are displayed.

- Question: How can you display lines 56-60 of a file that has 1000 lines using pipes?

head and tail

`head [-n number] file`

Prints the first `number` lines of a file. By default the first 10 lines are displayed.

`tail [-n number] file`

Prints the last `number` lines of a file. By default the last 10 lines are displayed.

- Question: How can you display lines 56-60 of a file that has 1000 lines using pipes?
- Answer: `head -n 60 file.txt | tail -n 5`

sort

```
sort [-r] [-k field1[,field2]] [file ...]
```

Sorts a file (or files) line by line.

- `-r` sorts in reverse order
- Use `-k` to sort by a specific column (default: first column)
- Note that the `file` input is optional. Why?

- Sorting order:
  - Special characters, numbers, letters (lowercase before uppercase of same letter)

tee

tee [-a] file

The tee utility copies standard input to standard output, making a copy in file.

- -a Append the output to the file rather than overwriting it.

- Example:

  -
    ```
    echo "Hello" | tee greetings.txt
    Hello
    ```

uniq

uniq [-c|-d|-u] file

Check for **uniq**ue lines in `file`.

- Behavior is greatly dependant on given options.
- By default, `file` is printed to `stdout` with duplicates removed
- IMPORTANT: Input must be sorted! Duplicates are only found if adjacent
- `-c` count occurrences of each line
- `-d` output just the duplicate lines
- `-u` output just unique lines

tr

tr [-C] string1 string2 | tr -d string1

The tr utility either substitutes the characters in string1 to characters in string2 or deletes characters in string1 in standard input and outputs the result in standard output.

- Note: does not take file input
- Use -C to complement the characters in string1

- Example:
  - `cat file.txt | tr "[a-z]" "[A-Z]" > uppercase.txt`

wc

```
wc [-l | -w | -c | -m] file ...
```

Count the number of lines, words, bytes, or characters in `file`

- Use `-l` to count lines
- Use `-w` to count words
- Use `-c` to count bytes
  - Same as `-m` if multibyte characters are not present
- Use `-m` to count characters

grep History

- Ken Thompson, one of the inventors of Unix, was helping a fellow coworker do some textual analysis on The Federalist Papers.
- Thompson written his own program that allowed text searching by using regular expressions.
- He named this tool "**G**lobal **R**egular **E**xpression **P**rint", or simply grep

grep History

- Thompson's boss, Doug McIlroy, approached him about the need for a text searching utility.
- Thompson promised to work on it overnight, but really only spent about an hour fixing bugs, since he'd already written Grep and had been using it privately.
- He presented it to McIlroy the next day. And the rest is history.

Simple grep

`grep [-i|-c|-l|-n|-v|-o|-R] pattern file ...`

Search for `pattern` in each `file` and print matched lines

- `-i` ignore-case
- `-c` return total match count (**of lines**) instead of line contents
- `-l` return names of matched files, instead of line contents
- `-n` show line numbers
- `-v` return lines which do not match `pattern`
- `-o` print only the matching parts on separate lines
- `-R` read the files in directories, recursively

Using grep

- Learning how to use grep effectively boils down to learning the language to define patterns: *regular expressions*
- grep manual at gnu.org
  - https://www.gnu.org/software/grep/manual/
  - A 43-page document!

Extended Regular Expression Syntax

Special characters: `.?*+{|()[\^$`

All the other characters are ordinary characters.

`?*+{` are repetition operators. The ones beginning with `{` are called *interval expressions*:

Symbol	*Meaning*
.	Matches any single character
?	Matches preceding item 0 or once
*	Matches preceding item 0 or more times
+	Matches preceding item 1 or more times

Extended Regular Expression Syntax

Repetition operators continued

Symbol	*Meaning*
`{n}`	Matches preceding item exactly `n` times
`{n,}`	Matches preceding item `n` or more times
`{,m}`	Matches preceding item at most `m` times
`{n,m}`	Matches preceding item at least `n` times, but not more than `m` times.

Extended Regular Expression Syntax

Additional Notes

- The empty regular expression matches the empty string.
- Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated expressions.
- Two regular expressions may be joined by ‘|’. Either of the two expressions, which are called *a*lternatives, are matched.
- Repetition takes precedence over concatenation, which in turn takes precedence over alternation.
- A whole expression may be enclosed in parentheses to override precedence rules. An unmatched ‘)’ matches just itself.

RE Syntax

Bracket Expressions

- A *bracket expression* is a list of characters enclosed by '[' and ']'.
- It matches any single character in that list.
- If the first character of the list is the caret ^, then it matches any character **not** in the list.
- Examples:
  - `[0123456789]` matches any single digit.
  - `[^()]` matches any single character that is not opening or closing parenthesis.
- Special characters lose their special meaning inside bracket expressions.

RE Syntax

Range Expressions within Brackets

- Within a bracket expression, a *range* expression consists of two characters separated by a hyphen.
- It matches any single character that sorts between the two characters, inclusive.
- In the default C locale, the sorting sequence is the native character order; for example, '`[a-d]`' is equivalent to '`[abcd]`'.
- To obtain the traditional interpretation of bracket expressions, you can use the 'C' locale by setting the `LC_ALL` environment variable to the value 'C'.

RE Syntax

Character Classes

Symbol	*Meaning*
`[:alnum:]`	Alphanumeric characters. Same as `[0-9A-Za-z]`
`[:alpha:]`	Alphabetic characters. Same as `[A-Za-z]`
`[:lower:]`	Lowercase letters
`[:upper:]`	Uppercase letters
`[:digit:]`	Digits
`[:xdigit:]`	Hexadecimal digits
`[:blank:]`	Blank characters: space and tab
`[:punct:]`	Punctuation characters

RE Syntax

Special Backslash Expressions

The ‘\’ character, when followed by certain ordinary characters, takes a special meaning:

Symbol	*Meaning*
`\b`	Matches the empty string at the edge of a word
`\B`	Match the empty string provided it’s not at the edge of a word
`\<`	Matches the empty string at the beginning of a word
`\>`	Matches the empty string at the end of a word

Example:

- ‘\brat\b’ matches the separate word ‘rat’, ‘\Brat\B’ matches ‘crate’ but not ‘furry rat’.

RE Syntax

Anchoring

- The caret ‘^’ and the dollar sign ‘$’ are special characters that respectively match the empty string at the beginning and end of a line.
- They are termed anchors, since they force the match to be *anchored* to beginning or end of a line, respectively.

RE Syntax

Differences of the Basic RE

Basic RE has functionally the same power as the Extended RE. However, Basic regular expressions differ from extended regular expressions in the following ways:

- The characters `?`, `+`, `{`, `|`, `(`, and `)` lose their special meaning; instead use the backslashed versions.
- If an unescaped '^' (or '$') appears neither first (last), nor directly after (before) '\(' or '\|', it is treated like an ordinary character and is not an anchor.
- If an unescaped '*' appears first, or appears directly after "\(' or '\|' or anchoring '^', it is treated like an ordinary character and is not a repetition operator.

grep examples

- `grep ^alias .bashrc`
  - Find lines in your `.bashrc` file that **start with** `alias`
- `grep -n ^$ *.c`
  - Find empty lines in all the c files in the current directory and report their line numbers
- `grep '\bse[et]\b' .bashrc`
  - Find occurrences of `see` or `set` that appear as whole words in `.bashrc`
- `grep -o -n -E '(0|[1-9][0-9]*)([eE][+-]?[0-9]+)?' *`
  - Find integers (possibly in scientific notation, as well) and print integers only with line numbers
- See more at: https://phoenixnap.com/kb/grep-regex