Slide 1

Administrivia

- Reminder: Homework 4 due Monday. Should be relatively easy and quick.
- Homework 5 on the Web. Due next Wednesday.

Slide 2

Minute Essay From Last Lecture

- Most people used `vi` or `vim` at least for some tasks, often because it was what they were shown in CS1. A few mentioned that they liked that it was lightweight and so suitable for quick edits. I agree! I also think usability over a text-mode connection is a plus.
- Several people mentioned liking to use IDEs such as Eclipse or Visual Code/Studio for large programming projects. Much though it pains me to admit it, they really probably are better for that task. One person mentioned that you do have to learn to use them. I agree! not everything with a GUI is "intuitive" :-).
- There were also a couple of votes for Atom and one for Sublime. I think I've seen people using these and they do look okay.

Slide 3

Homework 2 Essays

- Some people mentioned that this assignment was more difficult than the first one. Not a big surprise maybe.
- One person mentioned that there seems to be "some obscure utility for everything". Some truth to that!
- A couple of people mentioned that some problems more or less forced them to read `man` pages, and they found that good.
- One person said piping to `less` is now a favorite thing. Useful to also know how to pipe stderr too! (Or so I say.)

Slide 4

Regular Expressions

- From an old Wikipedia definition:

  > A regular expression (abbreviated as regexp, regex or regxp) is a string that describes or matches a set of strings, according to certain syntax rules. Regular expressions are used by many text editors and utilities to search and manipulate bodies of text based on certain patterns.

- Idea has roots in formal theory of languages, where the "languages" (sets of strings) described by regular expressions are exactly the ones accepted by finite state automata.

Slide 5

Regular Expressions and UNIX Tools

- Tools that use regular expressions include editors and also text-manipulation commands such as `grep` and `sed`. Also supported in many programming languages, especially (but not exclusively) ones for scripting (Perl, Python, `bash`, etc.).
- This being UNIX, not all the tools accept exactly the same syntax. POSIX defines two standards, "basic" and "extended". Some tools/languages add more. Simple stuff is very similar in all versions, fortunately. Key difference: In basic syntax, must precede many special characters with "escape character" (backslash).

  Also notice that to keep shell from doing its thing with your regular expressions (which generally you don't want), must enclose in single or double quotes.

Slide 6

Character Literals and Metacharacters

- Most characters represent themselves.

  `hello` matches what?
- Other characters are "special" (metacharacters):

  `^` matches start of line

  `$` matches end of line

  `.` matches any character (except newline)

  To use these as regular character literals, "escape" with a backslash.

Slide 7

Character Classes

- Character classes represent "one of these characters".

  Examples: `[abcd]`, `[0-9]`
- `^` at the start of a list means "any character other than these":

  Example: `[^abcd]`
- Most tools define some shorthand:

  Examples: `\s` for whitespace, `[:alpha:]` for letter

Slide 8

"OR" (Alternation)

- UNIX pipe symbol (`|`) separates alternatives. (Must escape in basic syntax.)

  Example: `cat|dog`
- (What about AND? Usually don't need it, or can get the same result another way — e.g., for `grep`, pipe one `grep` into another.)
- Example of use:

```
grep 'cat\|dog' foo
```

Slide 9

Quantifiers

- `*` means "preceding character (or group), zero or more times".

  Example: `.*`

- + means "preceding character/group, one or more times". (Must escape in basic syntax.)

  Example: `a+`

- `{N,M}` means "preceding character/group, N to M times". (Must escape curly brackets in basic syntax.)

- Notice that quantifiers are "greedy" — match longest string possible.

Slide 10

Grouping in Regular Expressions

- Use parentheses to group. (Must escape them in basic syntax.)

  Example: `(abc)(def)`

  Example: `(abc)*`

- Can then "backreference" groups, with `\1`, `\2`, etc.

  Example:

  ```
  sed 's/\(\S\+\) \(\S\+\)/\2\1/' foo
  ```

Slide 11

A Few More Tricks

- Angle brackets match beginning/end of word. (Must escape in basic syntax.)

  Example: `<hello>`

  (Note that this may not work on Mac OS X. What worked the last time I checked was "character classes" `[[:<:]]` and `[[:>:]]`.)

- Examples of use:

```
grep '\<bye\>' foo
```

Slide 12

Usage of Regular Expressions, Revisited

- Can use regular expression to search — `grep`, search in `vi`.
- Can also use them to modify — `sed`, search-and-replace in `vi`.
  Backreferences can be useful here!

Slide 13

Where to Learn More

- `man` and/or `info` pages for `sed`, `grep`; `info` page for `regex`.
- Online help for `vim`.
- Books and online references/tutorials . . .
- Useful advice from `vim`'s help:

  Which of these should you use? Whichever one you can remember.
- There are also programs that offer a GUI-ish environment for trying things out. Time permitting I will install one or more on the classroom/lab machines. There are also Web sites that offer these. A student pointed one out in class (`regex101.com`).

Slide 14

Minute Essay

- Try writing a regular expression that would match a "license plate" string of the form "one uppercase letter, then two digits, then three uppercase letters". (Hint: Remember that `[A-Z]` matches one uppercase letter. Similar syntax for digits.)
- Have you seen regular expressions in some other context? (Perhaps in Scala?)

Slide 15

Minute Essay Answer

- A not-so-hard-to-remember answer:

```
[A-Z][0-9][0-9][A-Z][A-Z][A-Z]
```