Slide 1

Administrivia

- Reminders: Homework 3 due today; Homework 4 due date changed to next Wednesday (in deference to this being possibly a busy weekk?).
- Homework 5 coming soon (about today's topic).

Slide 2

More Administrivia

- I've set up the Google Drive folders I've had in mind to use to communicate grade information. (If you were in one of my classes this past spring — same idea.)
- Folders are set up and shared. I didn't notify when I shared, but you should find the folder in GD's "shared with me", or search for a folder with name of the form *lastname,_initial* (a bit ugly but you probably can guess why I didn't want spaces — I have things set up to upload from a Linux directory, and spaces in filenames . . .).
- Right now folders contain only attendance-score reports; I'll add more as I grade, but this will let you see how this will work.

Slide 3

Minute Essay From Last Lecture

- Not a lot of replies yet, but interesting ones.
- I think a difficulty in using any old-style editor is if you use the standard GUI keyboard a lot it's hard not to. I had the opposite problem years ago trying to learn enough MS Word to collaborate with colleagues. My Word documents tended to have sequences of `jjjjkl`, and `vi` (yeah, not `vim`) beeped at me a lot.

  (I'm told you can become "bilingual" but it takes practice.)

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. (Double quotes allow variable substitution; single quotes don't.)

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

  (This may be different for some tools in Mac OS X. Linux usually uses GNU versions; Mac doesn't.)

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.)
- Note 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 . . .
- Often there's more than one way to do something. Useful advice from `vim`'s help:

  > Which of these should you use? Whichever one you can remember.

  (In fact this is good advice in general, though it's good to occasionally expand the things you remember.)
- There are also programs that offer a GUI-ish environment for trying things out. I haven't tried one lately but they can be a help.

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? (Probably in Scala and maybe C++?) Indeed, was anything today unfamiliar other than syntax?

Slide 15

Minute Essay Answer

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

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