Slide 1

Administrivia

- Reminder: Homework 3 due today, Homework 4 next week.
- Homework 5 on the Web. Also due next week? I'll ask you again in the minute essay about the pace of the class, possibly choose a later deadline, and tell you by e-mail.

Slide 2

Minute Essay From Last Lecture

- Half of you said `vim`. One said he'd used it even in 2320 and likes that it's "dependable". (Maybe that's in contrast to big IDEs, which have their advantages but sometimes misbehave in baffling ways?) If you like using the mouse, I believe `:set mouse=a` gives some support for it (not remotely of course).
- Other "votes": two for "sublime" (never heard of it, but Google has), one for "Spacemacs" (`emacs` with some configuration tweaks?), one for `nano` (newer/enhanced version of `pico`).

Slide 3

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 4

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 5

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 6

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 7

"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 8

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 9

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 10

A Few More Tricks

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

  Example: `<hello>`

  (Notice that this doesn't work on Mac OS X. Instead, one must use "character classes" `[[:<:]]` and `[[:>:]]`.)

- Examples of use:

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

Slide 11

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 12

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 13

Minute Essay

- Anything noteworthy about Homework 3?
- I asked this before but I'll ask for an update — is the pace/workload of the class still okay for a two-unit course?
- 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.)

Slide 14

Minute Essay Answer

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

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