Slide 1

Administrivia

- As noted in e-mail before the break: Schedule page now has links to readings for all topics except today, homeworks 1 through 4.
- “Sample programs” page, “useful links” pages updated.

Slide 2

Minute Essay From Last Lecture

- Question: What text editor do you currently use under Linux? What do you like/dislike about it?
- Answers: Almost unanimously — `vim` (with one vote for `gedit`, one mention of IDEs). Consensus — simple/compact but gets the job done.

Slide 3

Regular Expressions

- From 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 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.

  Example: `\.5`

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:

  Example: `\s` for whitespace

  Example: `[:alpha:]` for letter

  Example of use: `[^[:print:]]`

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. For `grep`, pipe one `grep` into another.)

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: `(abc)(.*)\1`

Slide 10

A Few More Tricks

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

  Example: `<hello>`

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!
  Example: `s/\(^..\)\(.*\)/\2\1`

Slide 12

Where to Learn More

- `man` and/or `info` pages for `sed`, `grep`.
- 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.

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.)

Minute Essay Answer

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

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