CS 114 - Fall 2004

Lecture 6 - Friday, October 8, 2004

Regular expressions

Text files are the primary way of storing data in Unix. A consequence of this is that you'll often need to search for something in text files. For example, you are editing a file in `emacs` or `vi` and looking for something, or you are in the shell and want to know which files contain a certain string.

The simplest searches are for a single word. In `emacs` you can use `C-s` to search for a word or phrase. In `vi` you can use the `/` command. But, sometimes you often want to search for something more general. You don't want to find an exact string, but any string that matches some "shape". For example, you might want to search for any digit, or any capitalized word in the first column, etc.,

There is a convenient notation for writing such things: *regular expressions*. Formally, a regular expression (or just regexp or regex) *r* represents a set of strings. A string *matches* a regex *r* if the string is one of the strings represented by *r*.

The simplest regex is just a string of characters, say `abc`. This regex represents exactly one string, namely "`abc`". Some characters have a special meaning in a regex:

- `.` matches any character. Hence the regex `a.c` matches strings such as `aac`, `abc`, `a3c`, `a;c`, etc.,
- `[abcd]` is called a *character class* and matches any single character in the brackets, in this case: `a`, `b`, `c`, or `d`. For example, `a[01b]c` matches any of `a0c`, `a1c`, or `abc`. The special notation *x*`-`*y* can be used anywhere in the brackets to represent all characters between *x* and *y*. For example, the regex `[a-z]` matches any lowercase letter; `[A-Za-z]` matches any lower- or uppercase letter. The notation `[^abc]` matches any character *except* those matched by `[abc]`.
- `*` matches 0 or more occurrences of the preceding regex. For example, `ab*c` matches any string that starts with an `a`, is followed by any number (including zero) of `b`'s, and then ends with a `c`. Similarly, `a[01b]*c` matches any string starting with `a`, is followed by any number of `0`'s, `1`'s or `b`'s, and then ends with a `c`.

If you want to match a special character, you need to *escape* it by preceding it with a backslash (`\`). Thus, the regex `a\**c` matches an `a` followed by zero or more `*`'s, followed by `c`. To match a `\`, you escape it too by writing `\\`.

Some characters help delineate where a matching can occur. These characters are called *anchors*:

- `^` matches only at the beginning of a line. For example, `^abc` will match the string `abc` only if it appears at the beginning of a line.
- `$` matches only at the end of a line. For example, `abc$` will match the string `abc` only if it appears at the end of a line.

The above operators define *basic regular expressions*. These operators are supported in most programs that use regular expressions of some sort. Some programs implements *extended* regular expressions, which provide the following additional operators:

- `?` matches 0 or 1 occurrence of the preceding regex. For example, `ab?c` matches the strings `ac` and `abc`.
- `+` matches 1 or more of the preceding regex. For example, `ab+c` matches the strings `abc` and `abbc`, but not `ac`.
- `(` and `)` can be used to pack together a regular expression so that `*`, `+`, or `?` can be applied to it. Thus, `a(bc)*d` matches `ad`, `abcd`, `abcbcd`, `abcbcbcd`, etc., For some programs, `\(` and `\)` are the grouping characters and `(` and `)` match literal left and right parentheses.

- `|` can be used to define choice. For example, `a(bc|de)f` matches `abcf` and `adef`.

More operators exist, some allowing you to match the same substring in multiple places in a given string.

Using regular expressions in `grep`

Last time we saw that `grep` can be used to search for a string in a text file. For example:

```
% grep Z cs-faculty
Zabih, Ramin
Haas, Zygmunt
```

`grep` can search using regular expressions as well. In fact, the name `grep` comes from an old `ed` command: `g/re/p`, meaning global regular expression print.

There are three versions of `grep`:

- `fgrep 'string' file1 file2 ...` (fast grep) searches the given files for the *`string`*.
- `grep 'regex' file1 file2 ...` searches the given files for the basic regular expression *`regex`*.
- `egrep 'eregex' file1 file2 ...` (extended grep) searches the given files for the extended regular expression *`eregex`*.

By default, `grep` and company return every line of the files that contains a match of the regular expression. Some interesting options to `grep` are `-i` to ignore the case of letters during matches, `-v` to invert the match (that is, to output lines for which there is no match), and `-c` to count the lines that match.

Using regular expressions in `sed`

The `sed` command is a *stream editor*. While it can perform many functions, the most common use of `sed` is to perform substitutions on a text file. For example, the following command outputs to `stdout` the contents of `file` with all strings matching *`regex`* replaced with the string *`string`*.

```
% sed -e 's/regex/string/g' file
```

You can use the grouping operators `\(` and `\)` to *save* a matching string and use it later. For example, given a file containing a list of names in the format `Last, First`, you can make replace all occurrences of `Last, First` with `First Last` by using the command:

```
% sed -e 's/\(.*\), \(.*\)/\2 \1/' file
```

This matches all lines of the form `.*, .*`, that is any line containing a ", ", but also saves the portion of the matched string before the comma (the last name) in a buffer named `\1` and the portion of the matched string after the comma (the first name) in a buffer named `\2`. The entire matched string is replaced with `\2 \1`, that is with the first name followed by the last name.

Using regular expressions in `vi`

The `/` and `?` commands in `vi` accept regular expressions as search strings. For example, `/\.$` will find the next line ending with a "`.`".

You can also perform replacements with regular expressions using the `:s` (substitute) command. This command is identical to the `s` expression used by `sed`. For example, given a file containing a list of names in the format `Last, First`, you can make replace all occurrences of `Last, First` with `First Last` by using the command:

```
:1,$ s/\(.*\), \(.*\)/\2 \1/g
```

`vi` also allows you to run the text of a program through an external filter using the `!` command. For example, remove all lines in the file being edited that contain social security numbers, you could do:

```
:1,$! grep -v '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]'
```

Using regular expressions in `emacs`

The commands `C-s` and `C-r` can be used to search for strings in `emacs`. To search for a regex instead, use `M-C-s` and `M-C-r`. The minibuffer will ask you for a regex to search for.

You can also perform replacements of the strings matched by the regex. See the documentation for the function `replace-regexp` (use `M-x describe-function`).