Unix Tools	Lecture 7
CS114 Fall 2002	Wednesday, October 16, 2002

More text processing

Recall from last time that when we execute a command, we can redirect `stdout` to `file` by appending `> file`. If you want to *add* the output to the end of an existing file, you can use the `>> file` redirection operator.

(What happens if you attempt to redirect to a file that already exists? In fact, it depends. You may either get an error saying that the file already exists, or you may have the file deleted and replaced by the output of the command. The behavior depends on the shell that you are using the access Unix. We will see shells in the next lecture.)

Here are some interesting programs that can be used with redirection and pipelines:

- `cat` *`file1 file2 ...`* outputs all the files to `stdout`, concatenating them together.
- `head` *`file1 file2 ...`* lists the first 10 lines of each file to `stdout` (you can specify how many lines to list by using a *`-number`* option).
- `tail` *`file1 file2 ...`* lists the last 10 lines of each file to `stdout` (you can specify how many lines to list by using a *`-number`* option).
- `less` *`file`* displays *`file`* a screenful at a time.

For example, assume that you have a file `f`, for which you want the first ten lines in alphabetical order containing a match for `rp65`. The following pipeline does this.

```
fgrep 'rp65' f | sort | head
```

Since sorting can be quite time consuming, what you should do with the above is send the output for a file `out`, and compute the whole thing in the background. (Recall last lecture.):

```
fgrep 'rp65' f | sort | head > out  &
```

Most commands that expect input from a file will accept input from `stdin` when you do not specify a file (see the man pages for the command to confirm that this is the case). For example, `grep` will behave this way. This allows you, for example, to perform the above transformation on many files by using `cat` to concatenate the files, before piping the result to `fgrep`:

```
cat f1 f2 f3 | fgrep 'rp65' | sort | head
```

One question arises: what happens when you execute a command that reads its input from `stdin` (for example, grep without a filename), and you do not specify a redirection to read the input from a file or from a pipe? Unix will ask you to enter the file by hand. This can be confusing, so I'll walk you through it. Let's pick a simple example. The command `cat` sends the contents of a file to `stdout`. If you do not specify a filename, it will send the content of `stdin` to `stdout`. If you redirect the output to a file `f`, it will send the content of `stdin` to `f`, as in

```
cat < f
```

However, if you execute the command above from the command line, as is, then Unix will simply display a cursor and wait for you to type in something. In essence, it is asking for you to type something directly to the `stdin` of the command, to type in a file by hand. So you can just type away, pressing ENTER to start new lines, etc. Everything you type is fed to the `stdin` of `cat`. To tell Unix you're done entering the file, you press control-d (this is the end-of-file character, or EOF). This will indicate to Unix that you have finished entering the type, and it continues executing the command, in this case sending whatever you typed in to file `f`. So, if you try:

```
babbage% cat > f
this is a test
this is a line
this is another line
I'm done
babbage%
```

(pressing control-d after "I'm done"), you get these lines in file `f`, which you can see by doing a `cat f`:

```
babbage% cat f
this is a test
this is a line
this is another line
I'm done
```

Stream processing with sed

A very convenient utility to use in a pipeline is `sed`, which is a program that can perform subsitutions based on regular expression. There are two ways of invoking `sed`:

- `sed -e 'script' file` which applies the instructions in *script* to the content of *file* and sends the result to `stdout`, or
- `sed -f scriptfile file` which applies the instructions in the file *scriptfile* to the content of *file* and sends the result to `stdout`.

If you do not specify a file from which to get input, `sed` will takes its input from `stdin`.

A script is a set of instructions that you can use to indicate to `sed` what actions you want performed on the file. Refer to the man pages for `sed` for a complete description of possible instructions. Here, I will only describe one, namely a global form of regular expression substitution. The instruction:

`s/`*regexp*`/`*substexp*`/g`

will perform the substitution of any string matching *regexp* with *substexp*. So, if you want to replace every instance of, say, `rp56` in a file `f2` by `<ric netid>`, you could use:

```
sed -e 's/rp65/<ric netid>/g' f2
```

and if file `f2` contains:

```
this is rp65 a line
this rp65 is also a line
```

we get the following:

```
babbage% sed -e 's/rp65/<ric netid>/g' f
this is <ric netid> a line
this <ric netid> is also a line
```

More complex substitutions can occur, and I'll refer you again to the man pages for `sed` for more information.