THE UNIVERSITY OF WESTERN AUSTRALIA

Sed

Lecture 11

Michael J Wise

Sed – Stream Editor

- `sed` and `awk` are, in my view, the two most powerful general purpose Open Source shell scripting tools
- `cat`, `tr`, `wc`, `ls`, `cut`, etc are special purpose. Options used to specialise performance, but essentially do just one thing
- sed and awk go beyond options; each has its own little language
  - *See the article from Jon Bentley's Programming Pearls column in CACM on Little Languages*

sed

- sed performs editing functions "on the fly".
- sed is intended to work as part of a Unix pipeline, i.e. a filter, reading lines one-by-one from a file (or stdin), performing edit functions on the lines and then sending the lines to stdout

`sed` <*options*> <*file*> ...

- `-e` <sed operation>
  - *Apply this one operation to the file(s)*
- `-f` <file of sed operations>
  - *A file of sed operations, executed top-down on each line*
- `-n`
  - *Transformed output not sent to stdout (need* `p`*)*

Inline sed

- Simple actions can be specified in-line using one or more -e options.

```
sed -e 's/[()]/ & /g' -e 's/ */ /g' infile
```

- Note: Edit functions are performed sequentially, so an action performed by an earlier function may affect subsequent edit operations.
- For example, this command first looks for occurrences of either `(` or `)` and places a space around each found.
- The `g` at the end of the edit operation specifies that this is to be done for every occurrence of `(` or `)`, rather than just the first (which is the default action).

Inline sed

- The second edit operation then replaces every sequence of one or more blanks with a single blank (and again, this is done for every occurrence).
  - *Keep in mind that otherwise, a space before a bracket ends up being two spaces, then bracket*

Sed script

- Specifying sed commands using multiple `-e` operations can become (very) tedious.
- Alternative: place the commands in a file (or *sed script*) which is named via the `-f` <*file*> option.
- In a sed script sed operations are listed one per line
- sed script comments begin with a `#` in the first column (and go to <CR>).
- The format for commands is:

[<address> [, <address>]] <function> [<arguments>]

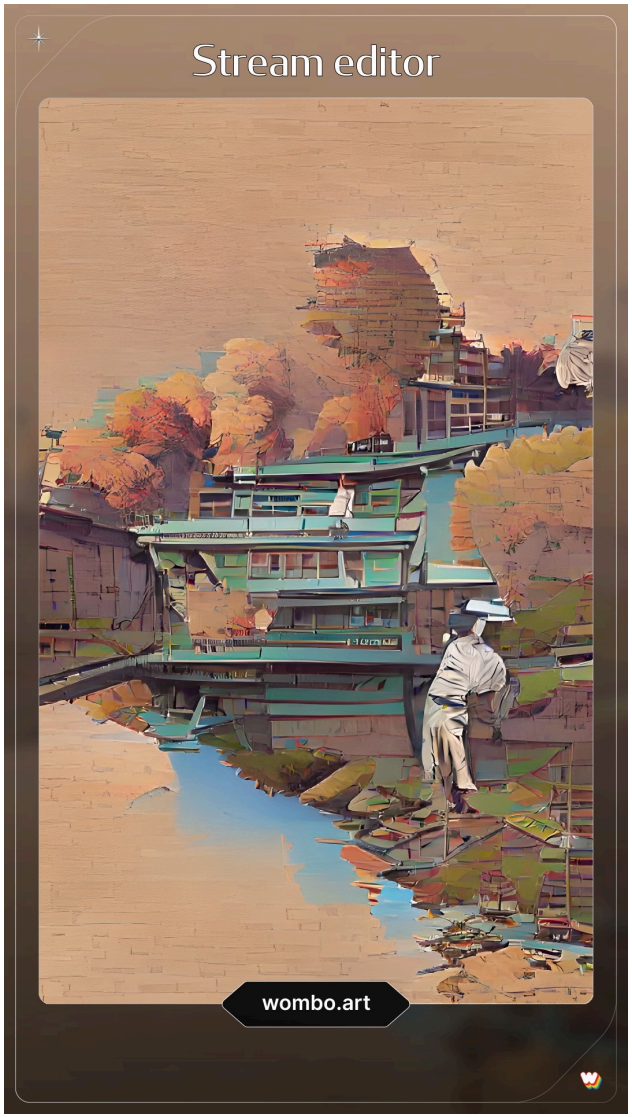
Addresses

- If no addresses are supplied, the function is applied to all input lines
- If one address is supplied, the function is applied just to that line: <*address*> <*operation*>, e.g. `42p`
- If two addresses are supplied, the function is applied to all lines in the range:
- <*address*>,<*address*> <*operation*>, *e.g.* `50,200p`
- A ! before the function selects lines OTHER THAN those specified by the addresses

Addresses

- An address can be:
  - *Line number*
  - *. (the current line)*
  - *$ (the last input line)*
  - *0 (before the first line, only some commands!)*
  - *A context address specified by a regular expression enclosed between / /*
  - *Simple arithmetic on an address,*

Stream editor



Expo-sed: Some commands

`s/`*<regular expression>*`/`*<text>*`/[g]`

- Probably the most common operation – search for a string that matches *<regular expression>* and replace it with *<text>*.
- If the operation is followed by `g`, the replacement is done globally, i.e. everywhere in the string.

```
s/^[a-z][a-z]*/{&}/
```

- Take lines beginning with strings of lower-case alphabetics and place parentheses around the strings.
  - *`&` is whatever was matched by the regular expression, i.e. works like `\1` following `\(  \)`*

Expo-sed: Some commands

`d` Delete the lines indicated by the addresses.

```
1,5d
/^D[TR]/d
```

- Note that, after a line has been deleted there is no point applying the remainder of the script to the current line (!!), so a new line is obtained and the script started from the top.

`p` This option prints out the addressed lines. It is only found together with the `-n` command-line option.

`l` This option is similar to `p`, except that non-printable characters are displayed and long lines are folded

Sed tips

- Here, in no particular order, are some tips for getting sed scripts to work.
- Work incrementally
  - *Build your scripts up slowly, line by line, testing each bit as it is added against known examples.*
- In general, the best advice is 3 words,

  test *Test* *TEST*
- Delete early
  - *If you can identify extraneous input that is not required, delete it early (saves work downstream!)*

Sed tips

- sed is sequential
  - *Remember that the operation of one command may change conditions for latter commands. Where you are up to is the sum of the changes made to this point (can be very complex; changes to changes to ...)*
- complex operations can be broken down into a sequence of simple ones.
- Don't try to do everything in one script
  - *It is often easier to have a separation of concerns and multiple scripts (connected with pipes), or scripts piping to other utilities (e.g. cut, awk).*
- Avoid long one-liners; can be very hard to maintain/alter as needed

Demo

- Data cleaning is the first in nearly all data-science analyses. (Someone gave you perfect data? Really? 🤨)
- In the case of the Alice text, there are several instances of words that were accidently joined during transcription, e.g. “yardwhile”, “meanstomakeanythingprettier”
- Assume that any such infelicity is random so likely to appear only once
- Notice that the `count_occurences` script does at least part of what is needed so let’s munge that.
- Good idea to exclude common words (that just happen to appear only once)
  - */usr/share/dict/words*