CSE 303:
Concepts and Tools for Software Development

Ben Hindman / Dan Grossman

Spring 2005

Lecture 6 - Line Editors and Filters

Line Editors and Filters

ed, grep, sed, tr, sort, awk, uniq . . and many more

In practice grep/sed used a lot. Many other UNIX commands help the operation of sed and grep but most jobs can be accomplished within sed.

Awk is very powerful, it can be considered a programming language.

Tradeoffs:

- new syntax vs. using something you know (if hard to accomplish task?)
- speed & efficiency (c/c++) vs. builtin constructs (why shell scripting?)

Gnu Sed, Gawk, commercial products and regular expressions: see man pages for which version you are using and how they deal with regex's!

Line Editors

Using an arcane syntax you manipulate line by line of a file.

grep: from ed, g/re/p “**g**lobal **r**egular **e**xpression **p**rint”

- can’t grep (grab) multiple lines (grep -B2 -A2 [-C])
- can’t do replace or substitution
- can’t limit what is printed, whole line or filename

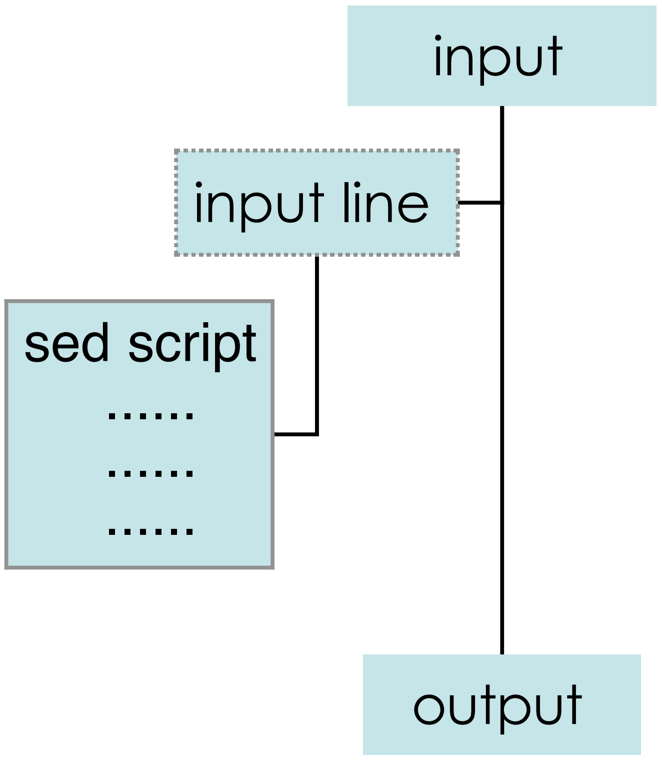
sed: “Stream Editor”, reads in one line at a time and executes instructions you give it

sed

You tell sed what to do by appending commands to the list of commands.

sed -e ‘p’ -e ‘l’
sed -n -e ‘p’ -e ’l’

(Unless you control the flow of the script or line grabing in any way, sed executes your commands in the order you appended them then grabs the next line.)



sed continued . . . .

sed reads each line into a pattern space.

You can address line(s) [pattern spaces] by line numbers and regular expressions . . . from the man pages:

- a command with no addresses selects every pattern space
- a command with one address selects all pattern spaces that match the address (1, /./, $)
- a command with two address selects the "inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second . . ." -> see the man page please.
- ! Matches all lines but the line that the address matches (1!, $!)

Lets get to some examples!

sed continued . . . .

- Use /re/ to specify an address with a regular expression and /re1/,/re2/ to specify two addresses.
- To group multiple commands together for certain line matches use the '{' and '}' characters:

```
{function; function; } =>   {function
                            function
                            }
```

(notice the ; and new line correspondence)

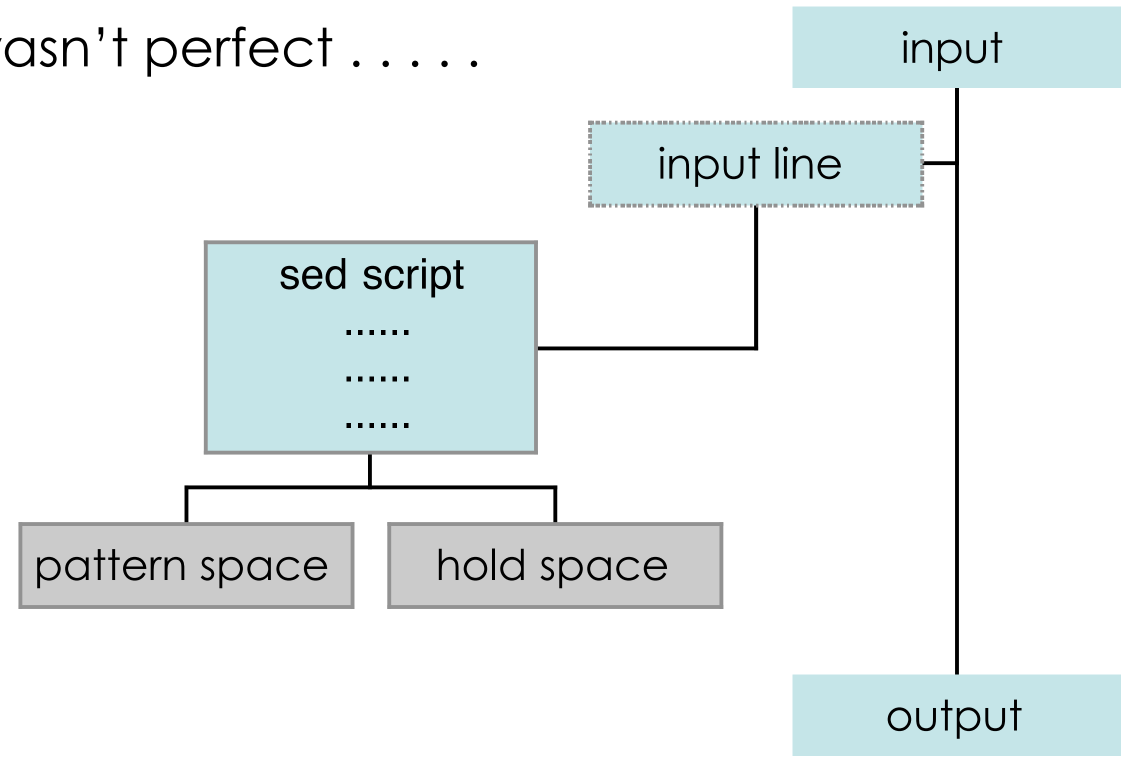
Some functions/commands can be addressed with two addresses while some are addressed with one address and some with no addresses.

See the man page for all the functions/commands that can be used and try them out.

sed continued . . . .

I lied, the picture wasn't perfect . . . . .

sed has a buffer to hold lines or text you might want to use later. Functions and commands operate on either the hold space or the buffer space, or both. The hold space is persistent over multiple line reads.



sed gotchas

- sed doesn’t let you match the new line at the end of a file, use $ for the end of a line. (Don’t confuse it with the last line address)
- you can match \n inside RE’s only after you have grabbed a second line. (N or n)
- if you want to add a new line into the output with the substitute command actually include the new line in the command script.

  ```
  sed -e ‘/hello/s//\\
  &/’
  ```
- when scripts get complicated command lines get confusing, put them in a file and use sed -f

awk

Awk is best used on structured input. In fact, awk makes the assumption that its input is structured and not just an endless string of characters.

……..

In the simplest case (which is all we are going to talk about today) awk takes each input line as a record and each word, separated by spaces or tabs, as a field. We use the word delimiter to describe the characters separating the fields and if we choose to we can set the delimiters ourselves (excel comma delimited files anyone?)

Two or more consecutive spaces or tabs count as a single delimiter.

$0 refers to the whole record, $1 the first field, $2 the second field, and so on.

See example.

bash and beyond . .

Other command interpreters (shells) do exist! You don't have to use tcsh or csh on your own linux boxes or on attu. So, since we end the shell section of the course now, let me say a few words about the most popular shell.

- all builtin's in bash are considered programs, ie if [expression] means that if and [are programs. This isn't necessary in the shell, since you can program it to parse however you want, but this requires less parsing. (diff, test syn.) [Pros or Cons?]
- help is builtin!!! (help [)
- my favorite: reverse history search! (^r)
- functions in bash (alias is not as powerful in bash, word events)
- less limitations in terms of variable size and word size (man tcsh)
- bash is big and slow . . . (man bash /bugs)

I encourage everyone to learn more! When you put this stuff together the possibilities are endless!

```
cat myninja.htm | sed -e 's/ninja/koala/g' > koala.htm
```

Since we don’t have | (alternation, union) howelse can we grab all possible types of ninja

```
cat myninja.htm | sed -e 's/ninja/koala/g;s/Ninja/Koala/g;s/NINJA/KOALA/g' > koala.htm
```

Oops! We need to change the links first!

```
cat myninja.htm | sed -e '/src/s/src="\([^"]*\)"/src="http:\/\/www.realultimatepower.net\/\1"/g' > koala.htm
```

Now lets go back and change the ninjas.

```
cat koala.htm | sed -e '/src/!{s/ninja/koala/g;s/Ninja/Koala/g;s/NINJA/KOALA/g;}' > koala2.htm
```

Commands used: ! s /address/ {}s//g \n (back referencing)

```
cat koala.htm | sed -e 's/<[^>]*>//g;/<[^>]*.*$/{:loop;$!N;s/<[*>]*>//g;t
      end;b loop;};:end’
cat koala.htm | sed -e 's/<[^>]*>//g
/<[^>]*.*$/{:loop
$!N
s/<[*>]*>//
t end
b loop
}
:end’
```

A shorter version:

```
sed -e :a -e 's/<[^>]*>//g;/</N;//b a’
```

Now get rid of empty lines

```
/^$/d
/./!d
```

Commands used: b t :label N $ (address) ! d

Now, if I told you that = wrote the line number followed by a new line character, then tell me how to count lines with sed instead of say, wc:

```
cat myninja.htm | sed -n -e '$='
```

Will I ever use sed in real life?

First, where is the file dict? How did Dan get it before, I could look for it with find, but instead of waiting all that time, maybe its in my path, hmm, how can I easily put all those path's on my command line without typing them all out, argg.

```
find `echo $PATH | sed -e 's/:/ /g'` -name "dict"
```

I guess I'm a moron and its not in my path, oh well

Last homework people used tac, tac is reverse cat, its not on my linux box or my mac, AM I HELPLESS! NO!

```
head -100 /usr/share/dict/words | sed -n -e '1!G;h;$p' | less
tac /usr/share/dict/words | less
```

How about removing extra lines?

`sed '/./,/^$/!d'` (what is the caveat for this expression")

Commands used: /addr1/,/addr2/ G h p -n =

dos2unix:

```
printf "i am from windows\r\n\r\n" | sed -e 's/.$//’
```

```
printf "i am from windows\r\n\r\n" | sed -e ’l;s/.$//;l’
```

Commands used: l

```
w | awk '$1 == "jrp" && $5 ~ /.*[.:]..[ms]/ { if (system("echo jrp online | mail -s
    \"online\" 9704010147@vtext.com") != 0) print "Command Failed" }’
```

```
printf "1 * * * *       source `pwd`/wonline\n" | crontab
```

```
* * * * *       w | awk '$1 == "b" && $5 ~ /.*[.:]..[ms]/ { if (system("echo b online |
    mail -s \"online\" 9704010147@vtext.com") != 0) print "Command Failed" }'
```