Tools of the Trade

Regular Expressions
Cut, paste, sed, tr, grep, uniq, and sort
Regular Expressions

• Regular expressions are used by several different Unix commands, providing a convenient and consistent way of specifying patterns to be matched (recall filename substitution using *, ?, and [ ] construct.)
Regular Expression

• Matching Any Single Character: The Period ( . )

$vi$ intro or $ex$ intro

/ ... /

/ 1, $p$

1, $s/p.o/XXX/g$
Regular Expression

• Matching the Beginning of the Line: The Caret ( ^ )
  /\^the/
  1, $s/^/>>/
  1, $s/^/ /

• Matching the End of the Line: The Dollar Sign ( $ )
  contents$
  .$?
In general, if you want to match any of the characters that have a special meaning in forming regular expressions, you must precede the character by a backslash( \ ) to remove that special meaning.
  \.$ in /\.$/
  ^\.
  ^$ and ^$
Regular Expression

• Matching a Choice of Characters: The [ . . . ] Construct
  
  `/[tT]he/`  
  1, `$s/[aeiouAEIOU]///g`
  `[0123456789] or [0-9]    [A-Z] [A-Za-z]`
  `/[0-9]/    /^[A-Z]/`  
  1, `$s/[A-Z]/*/g`

  in general, regular expression characters such as *, . , [ . . . ], $ and ^ are only meaningful in the search string and have no special meaning when they appear in the replacement string.

  inverted match
  `[^A-Z]    [^A-Za-z]` (recall the shell usage of !)
  1, `$s/[^a-zA-Z]//g`
Regular Expression

• Matching Zero or More Characters: The Asterisk

\[(*)\]

\[X^* \ XX^*\]

1, \$s/ \ */ /g

.* a regular expression matches the longest string of characters that match the pattern.

e.*e in 1, \$s/e.*e/+++/  

[A-Za-z][A-Za-z]* in  

1, \$s/ [A-Za-z][A-Za-z]* /X/g

[A-Za-z0-9][A-Za-z0-9]* in

1, \$s/ [A-Za-z0-9][A-Za-z0-9]* /X/g
Regular Expression

• Matching a Precise Number of Characters: \{ ...
\}
There is a more general way to specify a precise number of characters to be matched by using construct
\{min,max\}
where min specifies the minimum number of occurrences of the preceding regular expression to be matched, and max specifies the maximum.
For example, X\{1,10\}, [A-Za-z]\{4,7\}
Regular Expression

• Matching a Precise Number of Characters: \{ ...

Examples in vi:
:1, $s/[A-Za-z]\{4,7\}/X/g

Special cases:
1. \{10\} exactly that many times
   [a-zA-Z]\{7\} and .\{10\}

   :1,$s/^./\{10\}/// and :1,$s/.\{5\}$/\\

2. +\{5,\} at least that many occurrences
   :1,$s/[a-zA-Z]\{6,\}/X/g
Regular Expression

- Saving Matched Characters:`\(...\)`
  It is possible to capture the characters matched within a regular expression by enclosing the characters inside backslashed parentheses. These captured characters are stored in “registers” number 1 through 9. For example, `^\(\.\)\)` matches the first character on the line and stores it into register 1.
Regular Expression

• Saving Matched Characters:\((...\)\)
To retrieve the characters stored in a particular register, the construct \n is used, where n is from 1-9.
for example,

\((...\)\)\1
\((...\).\)*\1$

\((...\)\)\(\(...\)\)
Regular Expression

• Saving Matched Characters: `(…\)`
When using the substitute command in vi, a register can also be referenced as part of the replacement string.
Phonebook example,
:1,$s/\((.*\)\) \((.*\)\)/\2 1/g
cut

- This command comes in handy when you need to extract various fields of data from a data file or the output of a command.
- The general format of the cut command is `cut -cchars file` where `chars` specifies what characters you want to extract from each line of `file`. For example, `-c5; -c1,13,50; -c20-50`. `cut -c5- data`
- Results to standard output
- Used as a filter in a pipeline.
cut –c examples

• The cut command is useful when you need to extract data from a file or command provided that file or command has a fixed format.

  $ who | cut –c1-8
  $ who | cut –c1-8 | sort
  $ who | cut –c10-16
  $ who | cut –c1-9,23-32
cut –d and –f Options

• For instance, the file /etc/passwd is the master file that contains the usernames of all users on your computer system and other information such as your user id number, your home directory, and the name of the program to start up when you log in. Its data fields are delimited by a colon character.

• The –d and –f options are used with cut when you have data that is delimited by a particular character. The format of the cut command in this case becomes

  cut –dchar –ffields file

where dchar is the character that delimits each field of the data, and fields specifies the fields to be extracted from file. Field numbers follow the same formats as character position numbers.
cut –d and –f Options Examples

$ cut -d: -f1 /etc/passwd
  $ cut -d: -f1,6 /etc/passwd

• If cut command is used to extract fields from a file and the –d option is not supplied, cut uses the tab character as the default field delimiter.
  $ cut –f1 phonebook
use sed –n 1 file to tell whether the delimiter is tab or not.
  $ sed –n 1 phonebook
paste

• Instead of breaking lines apart, it puts them together. The general format of the paste command is

    paste files

where corresponding lines from each of the specified files are “pasted” together to form single lines that are then written to standard output.

The dash character – can be used in files to specify that input is from standard input.
paste examples

- $ paste names numbers
- $ paste names addresses numbers
paste –d Option

• Specify the field delimiter using –d option with the format `-dchars`
where `chars` is one or more characters that will be used to separate the lines pasted together. That is, the first character listed in `chars` will be used to separate lines from the first file that are pasted with lines from the second file; the second character listed in `chars` will be used to separate lines from the second file from lines from the third, and so on.
If there are more files than there are characters listed in `chars`, paste “wraps around” the list of characters and starts again at the beginning.
For example, `$ paste –d’+’ names addresses numbers`
`$ paste –d’+-’ names addresses numbers`
paste –s Option

• The –s option tells paste to paste together lines from the same file, not from alternate files.

For example,

$ paste –s names
$ ls | paste –d’ ’ -s - (recall echo *)
sed

• sed is a program used for editing data. It stands for *stream editor*. The general form of the sed command is

    `sed command file`

where `command` is applied to each line of the specified `file`. If no file is specified, standard input is assumed.

As sed applies the indicated command to each line of the input, it writes the results to standard output.
sed Examples

- $ sed ‘s/Unix/UNIX/’ intro
- $ sed ‘s/Unix/UNIX/’ intro > temp  
  $ cat temp
  $ mv temp intro
- $ sed ‘s/Unix/UNIX/g’ intro > temp
- $ who | sed ‘s/ .*$////’
sed -n Option and Deleting Lines

- $ sed -n ‘1,2p’ intro
  $ sed –n ‘/UNIX/p’ intro
- $ sed ‘1,2d’ intro
  $ sed ‘/UNIX/d’ intro
The `tr` filter is used to translate characters from standard input. The general form of the command is

```
tr from-chars to-chars
```

where `from-chars` and `to-chars` are one or more single characters. The result is written to standard output.
tr examples

• $ tr e x < intro

• $ cut –d: -f1,6 /etc/passwd
  $ cut –d: -f1,6 /etc/passwd | tr : ‘ ‘

Enclose b/w the single quotes is a tab character and it musted enclosed in quotes to keep it from the shell and give tr a chance to see it.
• The octal representation of a character can be given to `tr` in the format `\nnn`

• **Table 4.3. Octal Values of Some ASCII Characters**

<table>
<thead>
<tr>
<th>Character</th>
<th>Octal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bell</td>
<td>7</td>
</tr>
<tr>
<td>Backspace</td>
<td>10</td>
</tr>
<tr>
<td>Tab</td>
<td>11</td>
</tr>
<tr>
<td>Newline</td>
<td>12</td>
</tr>
<tr>
<td>Linefeed</td>
<td>12</td>
</tr>
<tr>
<td>Formfeed</td>
<td>14</td>
</tr>
<tr>
<td>Carriage Return</td>
<td>15</td>
</tr>
<tr>
<td>Escape</td>
<td>33</td>
</tr>
</tbody>
</table>
tr examples

• $ date | tr ‘ ‘ ‘\12’
• $ tr ‘[a-z]’ ‘[A-Z]’ < intro
• $ tr ‘[A-Z]’ ‘[a-z]’ < intro
tr –s Option

• Use the –s option to tr to “squeeze” out multiple occurrences of characters in to-chars.
  For example, tr –s ‘:’ ‘\11’
  $ cat lotsaspaces
  $ tr –s ‘ ‘ ‘ ‘ < lotsaspaces
tr –d Option

• Use the –d option to delete single characters from a stream of input. The general format of tr in this case is

   tr –d from-chars

where any single character listed in from-chars will be deleted from standard input. For example,

  $ tr –d ‘ ‘ < intro
  $ sed ‘s/ //g’ intro
grep

• Grep is used to search one or more files for particular character patterns. The general format of this command is

  `grep patterns files`

Every line of each file that contains pattern is displayed at the terminal (standard output). If more than one file is specified to grep, each line is also immediately preceded by the name of the file, thus enabling you to identify the particular file that the pattern was found in.
grep Examples

- $ grep Unix intro ($ sed –n ‘/Unix/p’ intro)
- $ grep cracker intro
- $ grep Susan phonebook
- $ grep shell *
- $ grep * intro
  $ grep ‘*’ intro
- $ who | grep jin
- $ who | grep chu
## grep and Regular Expression

- $ grep ‘[tT]he’ intro
  $ sed –n ‘;/[tThe]/p’ intro
- $ grep –i ‘the’ intro

### Some grep Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>grep '^[A-Z]' list</td>
<td>Lines from list containing a capital letter</td>
</tr>
<tr>
<td>grep '^[0-9]' data</td>
<td>Lines from data containing a number</td>
</tr>
<tr>
<td>grep '^[A-Z][0-9]' list</td>
<td>Lines from list containing five-character patterns that start with a capital letter and end with a digit</td>
</tr>
<tr>
<td>grep '.pic$' filelist</td>
<td>Lines from filelist that end in .pic</td>
</tr>
</tbody>
</table>
grep The –v Option

• Used to finding the lines that do NOT contain a specified pattern

• For instances,
  $ grep –v ‘Unix’ intro
  $ grep –v –i ‘unix’ intro
grep The -l Option

• Find Move_history in all C source files
  $ grep ‘Move_history’ *.c

• The -l option to grep gives you just a list of files that contain the specified pattern, not the matching lines from the files:
  $ grep -l ‘Move_history’ *.c
  $ grep -l ‘Unix’ *
  $ grep -l ‘Unix’ * | wc -l
  $ grep -l -i ‘unix’ * | wc -l
grep The -n Option

• Used to precede the each line from the file that matches the specified pattern by its relative line number in the file.

$ grep -n ‘Unix’ intro
grep The -n Option

• Used to precede the each line from the file that matches the specified pattern by its relative line number in the file.

$ grep -n ‘Unix’ intro
sort

• $ sort names

• By default, sort takes each line of the specified input file and sorts it into ascending order. Special characters are sorted according to the internal encoding of the characters. For example, on a machine that encodes characters in ASCII, the space character is represented internally as the number 32, and the double quote as the number 34.
sort

- $ sort names
- By default, sort takes each line of the specified input file and sorts it into ascending order. Special characters are sorted according to the internal encoding of the characters. For example, on a machine that encodes characters in ASCII, the space character is represented internally as the number 32, and the double quote as the number 34.
sort -Options

• The **-u** Option (to eliminate duplicate lines from the output)
  
  `$ sort -u names`

• The **-r** Option (to reverse the sort order)
  
  `$ sort -r names`

• The **-o** Option (to have the sort result to a file)
  
  `$ sort names -o sorted_names`
  `$ sort names > sorted_names`
  `$ sort names > names` *(won’t work—wiping out the names file)*

  `$ sort names -o names`
sort -Options

- The `-n` Option to `sort` specifies that the first field on the line is to be considered a number, and the data is to be sorted arithmetically. Compare the output of `sort` used first without the `–n` option and then with it:
  
  `$ sort data & $ sort -n data$

- Skipping Fields (fields are delimited by space or tab characters by default)
  
  `$ sort +1n data$
sort -Options

• In the -t Option, the character that follows the -t is taken as the delimiter character.
  $ cat /etc/passwd
  $ sort /etc/passwd
  $ sort +2n -t: /etc/passwd sort by 3rd field

• Other Options: Other options to sort enable you to skip characters within a field, specify the field to end the sort on, merge sorted input files, and sort in “dictionary order” (only letters, numbers, and spaces are used for the comparison).
  $ man sort
uniq

- It is used to find duplicate lines in a file. The basic format of the command is
  \texttt{uniq in\_file out\_file}
  In this format, uniq copies \texttt{in\_file} to \texttt{out\_file},
  removing any duplicate lines in the process.
  uniq’s definition of duplicated lines are \textit{consecutive-occurring} lines that match exactly.
- if \texttt{out\_file} is not specified, the results will be written to standard output. If \texttt{in\_file} is also not specified, uniq acts as a filter and reads its input from standard input.
uniq Example

- $ cat names   (not consecutive duplicate)
  Charlie
  Tony
  Emanuel
  Lucy
  Ralph
  Fred
  Tony
  $ uniq names
  $ sort names | uniq
uniq Example

- $ cat names  (not consecutive duplicate)
  Charlie
  Tony
  Emanuel
  Lucy
  Ralph
  Fred
  Tony
  $ uniq names
  $ sort names | uniq
  $ sort -u names
uniq -d Option

• It is used to find the duplicate entries in a file. It tells uniq to write only the duplicate lines to out_file (or standard output). Such lines are written just once, no matter how many consecutive occurrences there are. For examples,

  $ sort names | uniq -d
  $ sort /etc/passwd | uniq -d
  $ sort /etc/passwd | cut -f1 -d: | uniq -d
uniq -c Option

• It behaves like uniq with no option except that each output line gets preceded by a count of the number of times the line occurred in the input. For example,

$ sort names | uniq -c