CIS 18A
Introduction to Linux / Unix

Filters

De Anza College
Instructor: Clare Nguyen

Topics

- Filters
- head, tail
- cut, paste
- uniq, wc
- diff, cmp
- tr
- sort
- egrep, fgrep, grep

Filters (1 of 2)

- Filters are a special group of utilities that work the following way:
  - They accept multiple lines of input.
  - Their output is a subset of the input lines because some of the input lines have been 'filtered out'.
- Features that apply to all filters:
  - They accept multiple lines of input which can come from:
    - multiple lines of a text file, or
    - multiple lines of output that are piped in from another command.
  - They return multiple lines of output. The output lines can be as many as the input, or none, or somewhere in between, depending on what you specify to be filtered out.
  - A filter reads in one line of input at a time, and based on what you specify, it will filter this line and send any possible output to standard out.
  - This means that if the input comes from a file, the input file itself is not modified by the filter. It also means that if you want the output of the filter to be saved in a file, you have to redirect the output to a file.

Filters (1 of 2)

- Filter that you've worked with:
  - cat: not much filtering, the number of input and output lines are the same.
  - more, less: the output is one screen of the input lines.

head and tail

head: displays the beginning part of a file.

- Format:
  - head filename — shows the first 10 lines of the file.
  - head –k filename — k is a number, shows the first k lines.
- If the file has fewer lines than 10 or k, head prints all of the file.

tail: displays the end part of a file.

- Format:
  - tail filename — shows on screen the last 10 lines of the file.
  - tail –k filename — k is a number, shows on screen the last k lines of the file.
  - tail –n +m filename — n is the option name and must be used, m is a number, shows on screen the from line m to the last lines of the file.
- If the file has fewer lines than 10 or k or m, tail prints all of the file.

cut and paste (1 of 3)

- cut and paste are designed to work with input lines that have fields and regular delimiters.
  - Fields are columns of data.
  - Delimiters are the symbols that separate 2 fields. A regular delimiter means that only one symbol is used throughout the file as delimiter.
- Example: The lines in the following file has 4 fields, with a regular delimiter of comma:

```
cis18a,Intro to Linux,fall,2016
cis18b,Advanced Linux,winter,2017
cis18c,Shell Scripting,spring,2017
```

cut and paste (2 of 3)

cut: prints to screen the specified fields in a file

- Format: cut -fn -d'c' filename
  - n is the specified field number, explained below.
  - c is the character used as the delimiter. The character needs single quotes if it is a metacharacter of the shell.
  - The default delimiter is tab. You don't need to use the –d option if tab is the delimiter used in the input lines.
- To select the field(s) that follow the –f option:
  - n to select field number n (n is a number).
  - n,m,k to select fields number n, m, and k (n, m, k are numbers).
  - n-k to select from field number n to field number k (n, k are numbers).

  field numbering starts at 1
- From the previous example file: cut -f1,3 -d',' exampleFile will result in:

```
cis18a,fall
cis18b,winter
cis18c,spring
```

cut and paste (3 of 3)

paste: puts together input lines side by side, resulting in appending fields together from left to right.

- Format: paste -d'c' file_list
  - c is the character used as the delimiter. The character needs single quotes if it is a metacharacter of the shell.
  - The default delimiter is tab. You don't need the -d option if you want to use tab as a delimiter between the files that are pasted.
  - file_list is the list of file names that are pasted in order from left to right.
- Example: If FileA is:

```
x
y
z
```

FileB is:

```
1 a
2 b
```

Then paste FileA FileB will result in:

```
x       1 a
y       2 b
z
```

paste puts the default tab between the first and second files when it pastes the files together because the delimiter is not specified.

uniq, wc

uniq: (for **uniq**ue) filters out non-unique consecutive lines. If *consecutive* input lines are identical, only one line will remain and the rest of the lines are filtered out.

- Format: uniq filename

wc: (for **w**ord **c**ount) shows the number of lines, words, and characters in a file.

- A word is a series of non-space characters surrounded by space.
- Each space is counted as a character.
- Format:
  - wc filename — shows number of lines, words, and characters
  - wc -l filename — (l for **l**ine) shows number of lines
  - wc -w filename — (w for **w**ord) shows number of words
  - wc -c filename — (c for **c**haracter) shows number of characters

diff, cmp

diff and cmp are both used to check whether there is any difference between 2 files.

diff (for **diff**erence): diff file1 file2

- If there is no output, the files are identical.
- If the files are different, diff shows the action (add, delete) that can be done on each line in file1 that's different so it can become identical to file2.

cmp (for **c**o**mp**are): cmp file1 file2

- If there is no output, the files are identical.
- If the files are different, cmp shows the location of the first character that's different.

tr (1of 3)

tr: (for **tr**ansliteration) accepts a *source* set of characters and a *destination* set of characters. tr searches each input line for each character in the source set, and changes it to a corresponding character in the destination set.

- tr does not accept a filename as an input. (< filename)
- Format: tr -options 'source chars' 'destination chars'
  - options can be c, d, or s (shown on next slides).
  - 'source chars' is the set of characters that will be replaced.
  - 'destination chars' is the set of characters used for replacement.
- There is a *one-to-one correspondence* between the 2 sets.
  - The first character of the source is replaced by the first character of the destination, the second source character by the second destination character, etc.
  - if the source set is shorter than destination set: extra characters in destination set are ignored.
  - if the source set is longer than destination set: the last character of destination set is used for each extra character of source set.

tr (2 of 3)

- The characters in the source set and destination set can be any text character.
  - This means any space or comma in the set will be used for replacement. Don't add these in the set if you don't want to replace them.
  - Even if the set of characters looks like a word, tr still looks at each individual character. For example: tr 'linux' 'LINUX' means that all l, i, n, u, and x characters will be replaced with an uppercase equivalence, and not just the word linux will be replaced.
- You can specify a range of characters for the set:
  - 'a-z' or 'A-Z' or '0-9' — complete set of letters or numbers.
  - 'a-f' or '5-9' — partial set of letters or numbers.
  - 'a-f0-9' — combined sets of letters or numbers.

  put the - character at the end of the set if you want to include it as a character in the set.
- Example: tr 'a-d' 'xyz'

  a becomes x, b becomes y, c becomes z, d becomes z in the output lines.

tr (3 of 3)

Options

- c : (for **c**omplement) all characters that are **not** in the source set will get changed to characters in the destination set.

  Example: tr -c 'a-z' '*'
  all characters that are *not* a lowercase letter become a * character.
- d : (for **d**elete) this option requires only the source set as an argument. All characters in the source set get deleted.

  Example: tr –d 'a-z'
  all lowercase letters get deleted.
- s : (for **s**queeze or **s**quash) after all characters in the source set are replaced, any consecutive characters that are identical are squeezed into 1 instance of this character.

  Example: tr -s 'ab' 'xy' on the input line aaabc a
  results in an output of: xyc x
  (first 3 a's become 3 x's, and then squeezed into 1 x)

You can combine options:

tr –dc 'a-z' means all characters that are not a lowercase letter will be deleted.

sort Introduction

- sort : puts the input lines in a specified order.
- The default order is based on ASCII value.
  - ASCII value: every text character is stored as a specific number in memory. This number is the ASCII value of the character. The one-to-one correspondence between each text character and its number representation is stored in the ASCII table.
  - In the ASCII table, the digits 0-9 are in numeric order, and letters a-z and A-Z are in alphabetical order. In the table, the digits come first, then uppercase, then lowercase letters.
  - You can run man ascii to see the complete ASCII table
- Default format: sort filename
- How sort works by default:
  - Start by comparing the first character of the input lines
  - Keep comparing the corresponding characters in the input lines until it finds 2 characters that are not the same. The character with the smaller ASCII value comes first, therefore the line with that character comes first.
- For voyager, you need to set an environment variable before using sort: export LC_ALL="POSIX"

sort Options (1 of 2)

- c : (for **c**heck) sort only prints a "disorder" message to screen if the input lines are not sorted, otherwise, if the file is sorted, nothing is printed to screen. This option is useful if you want to check if the file is sorted but don't want to see the file printed to screen.
- r : (for **r**everse) the lines will be sorted in reverse ASCII order.
- f : (for **f**old over) sort "folds" all lowercase letters into uppercase first, then does the sorting. sort will then see all uppercase letters while sorting, resulting in a case insensitive sort. This means 'a' is considered to be the same as 'A', for example. The output lines will remain in the same case as the input lines and will not be in all uppercase.
- d : (for **d**ictionary) sort ignores all characters that are not letters, only letters are used to determine the sort order. If a line does not have letters, sort ignores the line and pushes it up to the top of the sorted list.

sort Options (2 of 2)

- n : (for **n**umeric) sort looks at 123 as the number 123 rather than the character 1, character 2, character 3. This option is useful if you have numbers that need to be sorted.

  For example, without the –n option, 123 will come before 45 since the character 1 comes before the character 4. With the –n option, the number 45 is less than the number 123, so 45 will come before 123.
- M : (for **M**onth) sort looks at the first 3 characters as abbreviation for the month names (Jan, Feb, Mar, etc.), and sorts by calendar order. If the first 3 characters (regardless of case) don't match the month abbreviation, sort ignores the line and pushes it up to the top of the sorted list.
- t : (for delimi**t**er) the default delimiter is space or tab. Use this option if the delimiter is any other character.

sort by Fields

- If the input lines have fields with delimiters, you can tell sort to sort by a specific field in the line, instead of sort from the beginning of the line.
- Field numbering starts at 1.
- To sort by field(s): sort +n1 -n2 filename
  where n1, n2 are numbers

  sort will skip n1 fields, start comparing characters at field n1+1, and stop comparing characters when it reaches field n2+1
- Examples:

  sort +2 -4 filename

  sort by fields 3 and 4 only (start sorting at field 3 and stop sorting when reaching field 5).

  sort +1 -2 filename

  sort by field 2 only (start sorting at field 2 and stop sorting when reaching field 3).

sort with Multiple Passes (1 of 2)

- By default sort will sort the input lines one time, and then print the sorted lines on screen. This is one *pass* of the sort.
- When sorting by fields, it may be useful to sort with several passes.
- For example, if you sort by field 1 (by month) in the first pass, and there are 8 input lines with identical month in field 1. You can then ask sort to go through a second pass, and sort by field 5 (by day) all 8 lines that have identical month: sort +0 -1 +4 -5 filename
  The command above means that sort will sort by the field 1 only (the month field) in the first pass, then for all lines that have the same month in field 1, sort will have a second pass that sort these lines by field 5 (the day field).
- You can run as many passes as needed.
- The passes don't have to be in field order.
- For example, sort +3 -4 +0 -1 +4 - 5 filename

  The command above will sort by field 4 in the first pass, and for all lines that have the same value in field 4, run a second pass to sort by field 1, and for the lines that have the same values in both field 4 and field 1, run a third pass to sort by field 5.

sort with Multiple Passes (2 of 2)

- Multiple passes with options
  - If an option applies to all passes, put the option as a separate option on the command line:

    sort –r +3 -4 +1 -2 filename

    sort will do a reverse sort of field 4 in the first pass. Then for all lines with identical data in field 4, do a reverse sort of field 2 in the second pass.

  - If an option applies to only 1 pass, combine the option with the +n option:

    sort +5r -6 +0n -1 filename

    sort will do a reverse sort of field 6 in the first pass. Then for all lines with identical data in field 6, do a numeric sort of field 1 in the second pass.

grep (1 of 2)

- grep : (for **g**lobal **r**egular **e**xpression **p**rint) search through each input line for a specified pattern, and print the input lines that match the pattern.
- The pattern is specified by a regular expression (covered next in the Regular Expression module).
- Without using regular expressions, grep is very commonly used for selecting lines that match a specific text string.
- There are 3 utilities in the grep family:
  - fgrep (for **f**ast grep) works the fastest, used with text strings only and not with regular expressions.
  - grep : the oldest, works with a standard set of regular expression.
  - egrep: (for **e**xtended grep) works with the extended set of regular expression.

grep (2 of 2)

- Format: fgrep 'text string' filename
  grep 'regular expression' filename
  egrep 'regular expression' filename
- All 3 grep utilities have options that are covered in the CIS 18B class. In this class we use only egrep as a tool to learn regular expressions.

Next stop: Regular Expression