grep, awk, sed family tool

Grep

Global Regular Expression Pattern

With grep you can do simple text-based or regular expression search on the file you passed or can be also piped.

You can only provide in one pattern, you can provide in multiple pattern to search for by using the pipe symbol, but any parameter after the pattern are treated as filenames

Use the `-E` flag to use the extended regular expression notation, i.e. you don't have to escape the following character to get their special meaning, *(), ?, +, {},* and | in the regular expression.

^ and $ anchor symbol to force the match in the beginning of the line and end of the line respectively. They do not need to be escaped.

In addition, if you escape non-special characters like `\a, \t` it will just be treated as literal of `a, t` respectively.

Usages

`grep <insert word to search> filename`	This will perform a search on the entire file to find where the word occurs
`grep -i <word> filename`	Perform a case-insensitive search on the entire file
`grep -R <word> .`	Perform a search on all of the file in the current directory as well as the sub-directories
`grep -c <word> filename`	Count the number of matches
`grep -A -B -C <word> filename`	Use -A, -B, and -C to get context surrounding the matched text, after, before, and both before and after respectively

Awk

The *awk* tool allows you to basically split each of line of your given text into different fields, kind of like *pandas* in Python. You can then access each columns individually allowing you to perform

some level of numerical analysis.

The way to invoke *awk* is via:

`awk -F <delimiters> -f <awk program file/provide program in a single quoted string> -v var=4 <file to process>`

Basic Structure of Awk

```
BEGIN {
# Applied before processing every line
}

/regex/ {
# Applied to only lines that match the regex
}

$1 ~ /regex/ {
# Applied to only lines that has it's first field match the regex
}

$3 == "hello" {
# Applied to only lines that has it's third field equal to hello
}

{
# Applied to every line
}

END {
# Applied after every line is ran
}
```

First the way that awk program is ran is through these pattern block:

The *BEGIN* and *END* pattern block are special in that they are only ran before processing through rest of the line and after every line is ran. If you want to do some setup work before processing through the rest of the line, i.e. setting up variables, you can do it in those two pattern block.

Then you can have a regex match based pattern block so that the code inside will only run if the lines matched the regex.

You can have nothing as well, so that the code is run for every line.

If you have multiple pattern block matched, they will all run, it is not like if-statement where one is true the rest are skipped, if they are all true, they are all run, from the order they are prescribed.

Array In Awk

If you want to use an array in awk, just use it you don't need to declare it to use. Just start insert elements into the variable you want to use as an array.

Sed

The *sed* tool or stream editor allows you to do textual replacement for the lines of files you have pass it.

The syntax for *sed* is kinda meh, but let's go over the different kind of operation you can do with it.

Each *sed* command follows the syntax:

`[addr]X[options]`

Where *X* is a single-letter *sed* command. *[addr]* is an optional line addressing, telling from where to where to apply the command.

- Address can be a single line number (to only execute on that line)
- A regular expression (to only execute on matched lines)
- Range of lines (to execute command between the two ranges)

Without *[addr]* the command is implicitly global, applying to every line.

Two Common Sed Operations

1. **d (delete)**
   The delete operation will delete the lines specified. Again if no addresses then it is implicitly deleting every line
2. **s/regexp/replacement/flags**
   The substitute command allow you to specify the text or a regular expression that's matched to be replaced by whatever you want. You can also use groups within the regexp, but they have to be escaped otherwise, they will be interpreted literally.

   In addition, the *g* flag can be provided as the last argument to do every replacement possible across the entire line, otherwise, it will only replaced the first match it found.

Addressing

There are different ways to do addressing, picking which line you are going to apply the command over.

1. Empty: If none, the command is global by default, meaning it applies to every line
2. One number: By providing one number you are only executing the command for that particular line
3. /regex/: If you provide a regular expression as the address, then it will only apply to the lines that matches the regular expression
4. start,end: You can also specify an address range using comma, the range can be numeric, regular expression, or even a mix of both.

   Ex: '4,17d': Delete line 4 to 17 inclusive.

Note: You can also append "!" to negate the address specification if you provide it right after the address. To pick every line that is not part of the provided address.

Examples

```
# This command replaces cat with dog, however, it is only the first occurrence it finds.
sed 's/cat/dog/' input

# Same as previous, but replaces every occurrence of cat with dog in a line.
sed 's/cat/dog/g' input

# Only apply the substitution to lines that contains apple, and only replaces the first
occurence, from apply -> app.
sed '/apple/s/pple/pp/' input

# Delete line 3-4 inclusive
sed '3,4d' input

# Delete every line
sed 'd' input
```

Revision #13
Created 2022-11-18 16:56:18 UTC by Tamarine
Updated 2023-05-14 02:46:19 UTC by Tamarine