Ruby One-Liners

- [Reference Index (/)](/)
- [Programming Languages (/programming_languages)](/programming_languages)
- [Ruby (/programming_languages/ruby)](/programming_languages/ruby)
- Ruby One-Liners

Source

[Source by Dave Thomas (http://www.fepus.net/ruby1line.txt)](http://www.fepus.net/ruby1line.txt)

File Spancing

```
# double space a file
    $  ruby -pe 'puts' < file.txt
# triple space a file
    $  ruby -pe '2.times {puts}' < file.txt
# undo double-spacing (w/ and w/o whitespace in lines)
    $  ruby -lne 'BEGIN{$/="\n\n"}; puts $_' < file.txt
    $  ruby -ne 'BEGIN{$/="\n\n"}; puts $_.chomp' < file.txt
    $  ruby -e 'puts STDIN.readlines.to_s.gsub(/\n\n/, "\n")' < file.txt
```

Numbering

```
# number each line of a file (left justified).
    $  ruby -ne 'printf("-%6s%s", $., $_)' < file.txt
# number each line of a file (right justified).
    $  ruby -ne 'printf("%6s%s", $., $_)' < file.txt
# number each line of a file, only print non-blank lines
    $  ruby -e 'while gets; end; puts $.' < file.txt
# count lines (emulates 'wc -l')
    $  ruby -ne 'END {puts $.}' < file.txt
    $  ruby -e 'while gets; end; puts $.' < file.txt
```

Text Conversion and Substitution

```
  # convert DOS newlines (CR/LF) to Unix format (LF)
  # - strip newline regardless; re-print with unix EOL
      $  ruby -ne 'BEGIN{$\="\n"}; print $_.chomp' < file.txt

  # convert Unix newlines (LF) to DOS format (CR/LF)
  # - strip newline regardless; re-print with dos EOL
      $  ruby -ne 'BEGIN{$\="\r\n"}; print $_.chomp' < file.txt

  # delete leading whitespace (spaces/tabs/etc) from beginning of each line
      $  ruby -pe 'gsub(/^\s+/, "")' < file.txt

  # delete trailing whitespace (spaces/tabs/etc) from end of each line
  # - strip newline regardless; replace with default platform record separator
      $  ruby -pe 'gsub(/\s+$/, $/)' < file.txt

  # delete BOTH leading and trailing whitespace from each line
      $  ruby -pe 'gsub(/^\s+/, "").gsub(/\s+$/, $/)' < file.txt

  # insert 5 blank spaces at the beginning of each line (ie. page offset)
      $  ruby -pe 'gsub(/%/, "   ")' < file.txt
      FAILS! $  ruby -pe 'gsub(/%/, 5.times{putc " "})' < file.txt

  # align all text flush right on a 79-column width
      $  ruby -ne 'printf("%79s", $_)' < file.txt

  # center all text in middle of 79-column width
      $  ruby -ne 'puts $_.chomp.center(79)' < file.txt
      $  ruby -lne 'puts $_.center(79)' < file.txt

  # substitute (find and replace) "foo" with "bar" on each line
      $  ruby -pe 'gsub(/foo/, "bar")' < file.txt

  # substitute "foo" with "bar" ONLY for lines which contain "baz"
      $  ruby -pe 'gsub(/foo/, "bar") if $_ =~ /baz/' < file.txt

  # substitute "foo" with "bar" EXCEPT for lines which contain "baz"
```

```
    $  ruby -pe 'gsub(/foo/, "bar") unless $_ =~ /baz/' < file.txt

# substitute "foo" or "bar" or "baz".... with "baq"
    $  ruby -pe 'gsub(/(foo|bar|baz)/, "baq")' < file.txt

# reverse order of lines (emulates 'tac') IMPROVE
    $  ruby -ne 'BEGIN{@arr=Array.new}; @arr.push $_; END{puts @arr.reverse}' < file.txt

# reverse each character on the line (emulates 'rev')
    $  ruby -ne 'puts $_.chomp.reverse' < file.txt
    $  ruby -lne 'puts $_.reverse' < file.txt

# join pairs of lines side-by-side (like 'paste')
    $  ruby -pe '$_ = $_.chomp + " " + gets if $. % 2' < file.txt

# if a line ends with a backslash, append the next line to it
    $  ruby -pe 'while $_.match(/\\$/); $_ = $_.chomp.chop + gets; end' < file.txt
    $  ruby -e 'puts STDIN.readlines.to_s.gsub(/\\\n/, "")' < file.txt

# if a line begins with an equal sign, append it to the previous line (Unix)
    $  ruby -e 'puts STDIN.readlines.to_s.gsub(/\n=/, "")' < file.txt

# add a blank line every 5 lines (after lines 5, 10, 15, etc)
    $  ruby -pe 'puts if $. % 6 == 0' < file.txt
```

Selective Printing of Certain Lines

```
# print first 10 lines of a file (emulate 'head')
    $  ruby -pe 'exit if $. > 10' < file.txt

# print first line of a file (emulate 'head -1')
    $  ruby -pe 'puts $_; exit' < file.txt

# print the last 10 lines of a file (emulate 'tail'); NOTE reads entire file!
    $  ruby -e 'puts STDIN.readlines.reverse!.slice(0,10).reverse!' < file.txt

# print the last 2 lines of a file (emulate 'tail -2'); NOTE reads entire file!
    $  ruby -e 'puts STDIN.readlines.reverse!.slice(0,2).reverse!' < file.txt

# print the last line of a file (emulates 'tail -1')
    $  ruby -ne 'line = $_; END {puts line}' < file.txt

# print only lines that match a regular expression (emulates 'grep')
    $  ruby -pe 'next unless $_ =~ /regexp/' < file.txt

# print only lines that DO NOT match a regular expression (emulates 'grep')
    $  ruby -pe 'next if $_ =~ /regexp/' < file.txt

# print the line immediately before a regexp, but not the regex matching line
    $  ruby -ne 'puts @prev if $_ =~ /regex/; @prev = $_;' < file.txt

# print the line immediately after a regexp, but not the regex matching line
    $  ruby -ne 'puts $_ if @prev =~ /regex/; @prev = $_;' < file.txt

# grep for foo AND bar AND baz (in any order)
    $  ruby -pe 'next unless $_ =~ /foo/ && $_ =~ /bar/ && $_ =~ /baz/' < file.txt

# grep for foo AND bar AND baz (in order)
    $  ruby -pe 'next unless $_ =~ /foo.*bar.*baz/' < file.txt

# grep for foo OR bar OR baz
    $  ruby -pe 'next unless $_ =~ /(foo|bar|baz)/' < file.txt

# print paragraph if it contains regexp; blank lines separate paragraphs
    $  ruby -ne 'BEGIN{$/="\n\n"}; print $_ if $_ =~ /regexp/' < file.txt

# print paragraph if it contains foo AND bar AND baz (in any order); blank lines separate paragraphs
    $  ruby -ne 'BEGIN{$/="\n\n"}; print $_ if $_ =~ /foo/ && $_ =~ /bar/ && $_ =~ /baz/' < file.txt

# print paragraph if it contains foo AND bar AND baz (in order); blank lines separate paragraphs
    $  ruby -ne 'BEGIN{$/="\n\n"}; print $_ if $_ =~ /(foo.*bar.*baz)/' < file.txt

# print paragraph if it contains foo OR bar OR baz; blank lines separate paragraphs
    $  ruby -ne 'BEGIN{$/="\n\n"}; print $_ if $_ =~ /(foo|bar|baz)/' < file.txt

# print only lines of 65 characters or greater
    $  ruby -pe 'next unless $_.chomp.length >= 65' < file.txt
    $  ruby -lpe 'next unless $_.length >= 65' < file.txt

# print only lines of 65 characters or less
    $  ruby -pe 'next unless $_.chomp.length < 65' < file.txt
    $  ruby -lpe 'next unless $_.length < 65' < file.txt

# print section of file from regex to end of file
    $  ruby -pe '@found=true if $_ =~ /regex/; next unless @found' < file.txt

# print section of file based on line numbers (eg. lines 2-7 inclusive)
    $  ruby -pe 'next unless $. >= 2 && $. <= 7' < file.txt

# print line number 52
    $  ruby -pe 'next unless $. == 52' < file.txt

# print every 3rd line starting at line 4
    $  ruby -pe 'next unless $. >= 4 && $. % 3 == 0' < file.txt

# print section of file between two regular expressions, /foo/ and /bar/
    $  ruby -ne '@found=true if $_ =~ /foo/; next unless @found; puts $_; exit if $_ =~ /bar/' < file.txt
```

Selective Deletion of Certain Lines

```
   # print all of file except between two regular expressions, /foo/ and /bar/
       $  ruby -ne '@found = true if $_ =~ /foo/; puts $_ unless @found; @found = false if $_ =~ /bar/' < file.txt

   # print file and remove duplicate, consecutive lines from a file (emulates 'uniq')
       $  ruby -ne 'puts $_ unless $_ == @prev; @prev = $_' < file.txt

   # print file and remove duplicate, non-consecutive lines from a file (careful of memory!)
       $  ruby -e 'puts STDIN.readlines.sort.uniq!.to_s' < file.txt

   # print file except for first 10 lines
       $  ruby -pe 'next if $. <= 10' < file.txt

   # print file except for last line
       $  ruby -e 'lines=STDIN.readlines; puts lines[0,lines.size-1]' < file.txt

   # print file except for last 2 lines
       $  ruby -e 'lines=STDIN.readlines; puts lines[0,lines.size-2]' < file.txt

   # print file except for last 10 lines
       $  ruby -e 'lines=STDIN.readlines; puts lines[0,lines.size-10]' < file.txt

   # print file except for every 8th line
       $  ruby -pe 'next if $. % 8 == 0' < file.txt

   # print file except for blank lines
       $  ruby -pe 'next if $_ =~ /^\s*$/' < file.txt

   # delete all consecutive blank lines from a file except the first
       $  ruby -e 'BEGIN{$/=nil}; puts STDIN.readlines.to_s.gsub(/\n(\n)+/, "\n\n")' < file.txt

   # delete all consecutive blank lines from a file except for the first 2
       $  ruby -e 'BEGIN{$/=nil}; puts STDIN.readlines.to_s.gsub(/\n(\n)+/, "\n\n")' < file.txt

   # delete all leading blank lines at top of file
       $  ruby -pe '@lineFound = true if $_ !~ /^\s*$/; next if !@lineFound' < file.txt
```