学到一个 vim 技巧 :v//d 作用是只显示匹配字符所在的行数，具体应用场景 vim 打开一个 simrun.log 文件，v/uvm_error/d，页面只显示 UVM_ERROR 行。

现在我们知道了这一个技巧，那么就产生了一个疑问，每一个代码是什么意思？这就引出了主角，vim 的 global 命令，让我们带着问题看看什么是 global 命令。

1：认识 global 命令

:global 命令通常采用以下形式

```
:[range] global[!] /{pattern}/ [cmd]
```

- 首先，在缺省情况下，:global 命令的作用范围是整个文件（%），这一点与其他大多数 Ex 命令（包括:delete、:substitute 以及 :normal）有所不同，这些命令的缺省范围仅为当前行（.）。
- 其次，{pattern} 域与查找历史相互关联。这意味着如果将该域留空的话，Vim 会自动使用当前的查找模式。
- 另外，[cmd]可以是除 :global 命令之外的任何 Ex 命令。在实际应用中，如表 5-1 中所列的那些 Ex 命令，无一不在处理文本过程中起到了极大的作用。顺便提一下，如果不指定任何 [cmd]，Vim 将缺省使用 :print。
- 还有，可以用 :global! 或者 :vglobal（v 表示 invert）反转:global 命令的行为。
- 最后需要指出的是 :global 命令在指定 [range] 内的文本行上执行时通常分为两轮。第一轮，Vim 在所有[pattern]的匹配行上做上标记。第二轮，再在所有已标

记的文本行上执行 [cmd]。另外，由于 [cmd] 的范围可单独设定，因此可在多行文本段内进行操作。

2：global 基本技巧

- **删除所有包含模式的文本行** :g/re/d

  将 :global 命令与 :delete 命令一起组合使用，可以快速裁剪文件内容。对于那些匹配 {pattern} 的文本行，既可以选择保留，也可以将其丢弃。

  该技巧就是本文开头列的第一个例子， :g//d ： v//d

  ||抄来的小知识 -- Grep 一词的来历：

  ||请仔细琢磨一下 :global 命令的简写形式：

  ||⇒ :g/re/p

  ||re 表示 regular expression，而 p 是 :print 的缩写，它作为缺省的 [cmd]使用。如果我们把符号 / 忽略掉，便会发现单词||“grep”已然呼之欲出了。

- **只保留匹配行**

  反转操作， :v/re/d

3：进阶技巧

- **将指定词收集进寄存器**

  通过把 :global 和 :yank 这两条命令结合在一起，可以把所有匹配 {pattern}的文本行收集到某个寄存器中。(yank（y）是复制命令）

  :g/UVM_ERROR

这样会把 UVM_ERROR 所有行搜集到一起，但一旦执行其他命令，这些信息会消失。

所以我们可以将包含单词行的文本复制到寄存器，再把寄存器内容粘贴到其他文件中。

例如我们用寄存器 a，

a. ⇛ **:g/TODO/yank A**

b. ⇛ **:reg a**

此处有一个窍门，即要用大写字母 A 引用寄存器。这意味着 Vim 将把内容附加到指定的寄存器，用小写字母 a 的话，则会覆盖原有寄存器的内容。

因此，这条 global 命令可以被解读为“将所有匹配模式 /TODO/ 的文本行依次附加到寄存器 a。此后，只需在任意分割窗口中打开一个新缓冲区，再运行 "ap 命令，就可以将寄存器 a 的内容粘贴进去了。

另一种方案是在原文件末尾追加，

⇛ :g/TODO/t$

- **递增**

有如下一个例子：

```
module.inputFIFO(0).txt;
module.inputFIFO(1).txt;
module.inputFIFO(2).txt;
module.inputFIFO(3).txt;
module.inputFIFO(4).txt;
```

我们要把代码中括号中的数字，替换成 v 由 0 开始的一个顺序递增序列，使用命令：

```
⇒ :let n=0 | g/inputFIFO(\zs\d\+/s//\=n/ | let n+=1
```

我们现在解释下每个命令的含义和执行过程：

```
let             为变量赋值
|               用来分隔不同的命令
g               在匹配后面模式的行中执行指定的 ex 命令
\zs             指明匹配由此开始
\d\+            查找 1 个或多个数字
s               在选中的区域中进行替换
\=              指明后面是一个表达式
```

||这条命令的执行过程为：

||给变量 n 赋值为 0；

||查找模式"opIndex(\zs\d\+"，使用变量 n 的值替换匹配的模式字符串；

||给变量 n 加 1；

||回第二步；

||需要说明一下"|"，它用来分隔不同的命令。

||(**\zs** /^\s*\zsif matches an "If" at the start of a line, ignoring white space. Can be used multiple times
|| /\(.\{-}\zsFab\)\{3}
|| Finds the third occurrence of "Fab"
||)
||(**\ze** end\ze\(if\|for\) matches the "end" in "endif" and "endfor"
||matches at any position, and sets the start of match there: The next char is the first char of whole match

||匹配结束，可以用于在多个相似值中指定要替换的数值

||)

- 数字序列（**range** 函数）

  我们只想生成类似下面的数字序列怎么办呢？

```
5
7
9
11
13
...
99
```

在 vim 里有很多方法，我们使用 range 函数如下：

```
:s/^\=range(5, 99, 2)/
```

其中 2 为步进的数值。

- 思考题

  range 这么好用，那递增函数可不可以不用 let 这么麻烦的函数呢，让命令行更简洁一点?

  我们试一下结合 range 函数，使用如下表达式：

```
:g/inputFIFO(\zs\d\+\ze/s//\=range(0,3)/
```

  会生成如下的序列：

```
module.inputFIFO(0
1
2
3).txt;
...
```

  并不是我们想要的结果。

4：结论

- **global 命令的广义形式如下：**

```
:g/{start}/ .,{finish} [cmd]
```

  可以将其解读为“对从 {start} 开始，到 {finish} 结束的所有文本行，执行指定的 [cmd]”。