Vim script - I

Ketan M

DLSW Group, NCCS

Overview

- Introduction

- How to run a Vim script

- Language Features and Examples (LF&E)
  - Basic
  - Advanced

- Vim script usage
  - rc, syntax files
  - plugins

- Summary and References

```
https://github.com/ketancmaheshwari/vim
```

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Open slide master to edit

Introduction

- Vim

A popular text editor.

- Vim script
  - A scripting language to control Vim:
    rcfiles, colon commands, macros, plugins are all Vim scripts
  - Interpreted, dynamic, partial runtime type checks
  - Somewhat like python but has **lots of quirks**!🇳🇱
  - Some features are simply **mind blowing**!🤯
  - we will cover version <= 8 (I hear v9 has new features)

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING
FACILITY

Open slide master to edit

How to run a Vim script

- Put the script in a .vimrc file and it will run when vim starts

- Type the script in ex (aka colon) mode

- source from within Vim `:so %`

- hashbang on top of script & `chmod u+x` the file
  - #!/usr/bin/env vim -u
  - #!/usr/bin/vim -u
  - Make sure to put the `quit` command in the end else will end up in vim

- run lines selectively from yank buffer with `:@"`

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING
FACILITY

Open slide master to edit

How to get help

- `:help` (or simply `:h`)
- `:h [variables, function, E128, list, dict, ...]`
- Documentation -- Vim's doc game is **top class**! 🤯

- Google / Stackexchange search returns relevant results
- Youtube has a few videos

OAK RIDGE
National Laboratory | LEADERSHIP COMPUTING FACILITY

Open slide master to edit

Language Features and Examples (LF&E) : Hello World!

```
MAC127024:vimscript km0$ cat hello.vim

#!/usr/bin/env vim -u

"This is a comment (only whole line comments allowed🇳🇱)
echo "Hello World!"
let msg = "Hello again!"
"echom saves message in message history
echom msg
"to quit from vim into terminal
quit
```

```
$ chmod u+x hello.vim
$ ./hello.vim
```

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING
FACILITY

Open slide master to edit

LF&E: Variables

- `:h variables`

- Vim script has 10 types of variables:
  string, number, float, list, dict, null, funcref, blob, job, channel

- Create a variable: `let varname = value`

- There is no char type

- Use type() to find variable type -- types are numerically
  encoded (`:h type`) 🇳🇱
  `:echo type(42)`

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING
FACILITY

LF&E: Variable Scoping (Namespace management) 🇳🇱

- By default, a variable is **global** - and it could be a problem

- Scoping is defined by prepending a scope identifier in front of a variable name, `g:` for global, `s:` for script `v:` for vim specific

```
let g:name = "Mithun"
let s:temp = -24
if v:version < 700 | echo "upgrade!" | endif 🇳🇱
```

- Other scopes are `a:` for function arguments `b:` for local to a buffer and `w:` for local to a window
```
"find currently defined variables
:let
```

| `|` is used to separate commands in a line
Workaround to put comments on same line

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING
FACILITY

Open slide master to edit

LF&E: Numbers

- Numbers are integers

- 3 kinds: Decimal, Octal and Hexadecimal

- Octal numbers are represented with a leading `0`
  `011 != 11` 🇳🇱

- Hex numbers represented with `0x` or `0X`

- However, `echo` will print them all in decimal 🇳🇱

- Cool way to convert from hex, octal to decimal or
  do inter-base arithmetic
  `:echo 0xabc` 🤯
  `:echo 0x7f - 036`

Open slide master to edit

LF&E: Strings

- Concatenation: . is the string concatenation operator 🇳🇱 (. is a bit overloaded and is used in other contexts)

- String slicing like python:

```
let lestring="Hello World"

echo lestring[0:4]
echo lestring[5:]
echo lestring[-3:]
echo lestring[:-4]
```

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING
FACILITY

Open slide master to edit

LF&E: Conditionals

- Three forms:
  if endif, if else endif, if elseif endif

- 0 is false everything else is true

- Ternary conditional expression like C

- str str comparison: byte values used

- str num comparison: If str don't look like number, they are converted to 0. 🇳🇱

```
let n = 4
echo n > 5 ? "n is big" : "n is small"
```

```
let x = 100
if x%2 == 0
   echom "even"
else
   echom "odd"
endif
q
```

```
if 0 == "one"
   echom "True"
else
   echom "False"
endif
q |"will print True
```

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING
FACILITY

LF&E: Comparison operators🇳🇱

Operator	Use 'ignorecase'	Match case	Ignore case
Equal	==	==#	==?
Not equal	!=	!=#	!=?
Greater than	>	>#	>?
Greater than or equal	>=	>=#	>=?
Less than	<	<#	<?
Less than or equal	<=	<=#	<=?
Matches with	=~	=~#	=~?
Does not match	!~	!~#	!~?
Same instance	is	is#	is?
Different instance	isnot	isnot#	isnot?

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Open slide master to edit

LF&E: Loops

- While and for loops are available

- break and continue available and behave as expected

```
"a multiplication table using while and for

let s:ctr=1
let s:n=13

while s:ctr <= 10
    echo s:n." X ".s:ctr." = ".s:n*s:ctr
    let s:ctr += 1
endwhile

"for loop
for s:i in range(11)
    echo s:n." X ".s:i." = ".s:nX*s:I
endfor
```

OAK RIDGE
National Laboratory | LEADERSHIP
COMPUTING
FACILITY

Open slide master to edit

LF&E: Lists

- Dynamic lists are supported -- pretty much like python

- Many builtin functions to work with lists, + for concat

- Looping through lists is similar to in python

```
"lists and list operations
let alist = [] |"empty list
let blist = ['foo','bar','baz']

call add(alist, 'baa')
call add(alist, 'moo')
echo alist |"display list items

for n in blist
  echo n
endfor

"len(), empty(), insert(), sort(),max()
reverse(),split(), join() etc. functions
available
```

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING FACILITY

LF&E: Dicts

- Dictionaries in Vim script look similar to those in python

- Supported by numerous functions

- Close integration with user-defined functions (more later)

```
"dicts and dict operations
let adict = {} |"empty dict
let bdict = {'cow':'moo','crow':'caw'}

echo bdict['crow']
echo bdict.crow |"same as above

let bdict.sheep = 'baa' |"update dict

"iteration over a dict
echo "key => val"
for key in sort(keys(bdict))
   echo key."=>".bdict[key]
endfor
```

OAK RIDGE
National Laboratory | LEADERSHIP
COMPUTING
FACILITY

Open slide master to edit

Built-in functions 🤯

- Tons of builtin functions, following are just categories, each category has 5-30 functions:
  string manipulation, list manipulation, dict manipulation, floating point computations, variables, cursor and mark position, buffer manipulation, system functions, file manipulation, date and time, windows and argument lists, command line, syntax, spelling, highlight, mappings, interactive, testing, ipc, timers, job, gui, misc.

- `:h functions` for an alphabetical list of all functions

OAK RIDGE
National Laboratory | LEADERSHIP
COMPUTING
FACILITY

Open slide master to edit

LF&E: User Defined Functions

- User defined functions are allowed, by default have a global scope but may be made local to script with `s:`

- Functions may be called with `call`

- Function name must start with a capital letter 🇳🇱

- `:fun` To find all the user defined functions

```
function Min(n1, n2)
    if a:n1 < a:n2
        return a:n1
    else
        return a:n2
    endif
endfunction
```

```
"redefine function
function! Min(n1, n2,
n3)
    <body>
endfunction
```

OAK RIDGE
National Laboratory | LEADERSHIP COMPUTING FACILITY

LF&E: User Defined Functions (contd)

- Functions may have variable number of arguments

- Those arguments may be called and used as `a:1, a:2,...` up to the **value in** `a:0` which holds the total count of arguments 🇳🇱

```
"may be invoked with up to 20 more args
function Show(st, ...)
   echo "First arg is: ".a:st
   let s:n = 1
   echo "Rest of the args are:"
   while s:n <= a:0
      echo a:{s:n}
      let s:n += 1
   endwhile
endfunction

call Show("Hello", "Hi", "Greetings")
```

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING FACILITY

Open slide master to edit

LF&E: Exception Handling try … catch … finally

- Similar to other languages
- An exception is simply a string with an exception number that `catch` will regex match🤯
- There maybe multiple catch commands but only one finally

```
try
    call Updatefile("/tmp/data.txt")
catch /E484:/
    echo "Sorry file not found"
catch /E21:/
    echo "Sorry file is not writable"
finally
    call Wrapup(foobar)
endtry
```

OAK RIDGE
National Laboratory | LEADERSHIP COMPUTING FACILITY

Open slide master to edit

Function Qualifiers

- Functions may be qualified with terms to give them special meaning or control them in other ways

- eg., a function qualified with `range` will work on a range of lines. Countwords will be invoked as: `:10,20call Countwords()` `a:firstline,a:lastline` are the built in args that will have 10 and 20 respectively

```
"discontinue running at first error
function Procdata("data.json") abort
.
.
endfunction

function Countwords() range
   let lnum = a:firstline
   let n = 0
   while lnum <= a:lastline
     let n += len(split(getline(lnum)))
     lnum += 1
   endwhile
   echo "found ".n." words."
endfunction
```

OAK RIDGE
National Laboratory | LEADERSHIP COMPUTING FACILITY

Open slide master to edit

Function References

- Functions may be assigned to variables as references using the `function()` function🇳🇱

- The variable that holds a function reference is of type `funcref`

- Funcref variable name must start with a capital letter

```
function Fizzbuzz(n)
   if a:n % 3 == 0 && a:n % 5 == 0
      return "FizzBuzz"
   elseif a:n % 3 == 0
      return "Fizz"
   elseif a:n % 5 == 0
      return "Buzz"
   else
      return "None"
   endif
endfunction

let Afunc = function('Fizzbuzz')

"call function may be used to invoke
the function referred to by the funcref
echo call(Afunc, [15])
```

OAK RIDGE
National Laboratory | LEADERSHIP COMPUTING FACILITY

Open slide master to edit

Dict Functions 🇳🇱🤯

- Functions can be directly associated with a dictionary!

- Apparently, they need not be starting with a capital letter!

- The dict is referred using self inside the function

- A dict function may be invoked using . on the dict

```
let en2es =
{'one':'uno','two':'dos','three':'tres'
}

function en2es.translate(line) dict
  return join(map(
  split(a:line),'get(self,v:val,"???")'
  )
  )
endfunction

echo en2es.translate('one two three
four')

"expected output
uno dos tres ???
```

Open slide master to edit

Miscellaneous

- Abbreviations are allowed 🤯

- No automatic garbage collection, use `unlet var` to delete defined variables, `delfunction Fname` to delete functions

- Special expressions for reading environment variables, Vim options and registers

```
" This is valid code
fu Mul(num)
    for i in range(11)
        ec i." X ".a:num." = ".i*a:num
    endfor
endfu
call Mul(7)
q

let x = 900 |"define a var
unlet x |"undefine it

echo $PATH |"env var
echo &ts |"value of tabstop
echo @r |"contents of register r
```

OAK RIDGE | LEADERSHIP
National Laboratory | COMPUTING
FACILITY

Summary

- A scripting language that comes packed with an editor

- Reasonably featureful

- Portable -- works anywhere vim is installed

- Actively being developed and maintained

OAK RIDGE
National Laboratory | LEADERSHIP
COMPUTING
FACILITY

Open slide master to edit

References

- vimdoc.sourceforge.net

- blog.prabir.me/posts/learning-vimscript

- begriffs.com/posts/2019-07-19-history-use-vim.html

- learnvimscriptthehardway.stevelosh.com

OAK RIDGE
National Laboratory | LEADERSHIP
COMPUTING
FACILITY

Open slide master to edit

Thank you for your time! Questions?

OAK RIDGE
National Laboratory | LEADERSHIP COMPUTING FACILITY

Open slide master to edit