Home

Style Guides

This is a collection of style guides for a wide selection of different programming languages. Each guide also provides a brief description of the language. More comparisions of the programming languages can be found in the Lexici repo.

!C++

VimScript

VimScript

Vim Script is the scripting language built into Vim. Based on the ex editor language of the original vi editor, early versions of Vim added commands for control flow and function definitions. Since version 7, Vim script also supports more advanced data types such as lists and dictionaries and object-oriented programming. Built-in functions such as `map()` and `filter()` allow a basic form of functional programming, and Vim script has lambda since version 8.0. Vim script is mostly written in an imperative programming style.

Portability

It's hard to get vimscript right. Many commands depend upon the user;s settings. By following these guidelines, you can hope to make your scripts portable.

Strings

Prefer single quoted strings

Double quoted strings are semantically different in vimscript, and you probably don't want them (they break regexes).

Use double quoted string when you need an escape sequence (such as `"\n"`) or if you know it doesn't matter and you need to embed single quotes.

Matching Strings

Use the `=~#` or `=~?` operator families over the `=~` family.

The matching behavior depends upon the user's ignorecase and smartcase settings and on whether you compare them with the `=~`, `=~#` or `=~?` family of operators. Use the `=~#` and `=~?` operator families explicitly when comparing strings unless you explicitly need to honor the user's case sensitivit settings.

Regular Expressions

Prefer all regexes with `\m\C`.

In addition to the case sensitivity settings, regex behavior depends upon the user's nomagic setting. to make regexes act like nomagic and noignorecase are set, preend all regexes with `\m\C`.

You are welcome to use other magic levels (`\v`) and case sensitivities (`\c`) so long as they are intentional and explicit.

Dangerous Commands

Avoid commands with unintended side effects.

Avoid using `:s[ubstitute]` as it moves the cursor and prints error messages. Prefer functions (such as `search()`) better suited to scripts.

For many vim commands, functoins exist that do the same thing with fewer side effects. See `:help function()` for a list of build-in functions.

Fragile Commands

Avoid commands that rely on user settings.

Always use `normal!` instead of `normal`. The latter depends upon the user's key mappings and could do anything.

Avoid `:a[ubstitute]`, as its behavior depends upon a number of local settings.

The sam eapplies to other commands not listed here.

Catching Exceptions

Match error codes, not error text.

Error text may be locale dependant.

General Guidelines

Messaging

Message the user infrequently.

Loud scripts are annoying. Message the user only when:

- A long-running process has kicked off.
- An error has occured.

Type Checking

Use strict and explicit checks where possible.

Vimscript has unsafe, unintuitive behavior when dealing with some types. For instance, `0=='foo'` evaluates to true.

Use strict comparison operators where possible. When comparing against a string literal, use the `is#` operator. Otherwise, check `type()` explicitly.

Check variable types explicitly before using them. Use `type()` and throw your own errors.

Use `:unlet` for variables that may change types, particularly those assigned inside loops.

Python

Use sparingly.

Use python only when it provides critical functionality, for example when writing threaded code.

Other Languages

Use vimscript instead

Avoid using other scripting languages such as ruby and lua. We cannot guarantee that the end user's vim has been compiled with support for non-vimscript languages.

Boilerplate

Plugin boilerplate includes

- Plugin creation
- Error handling
- Dependency checking

Plugin Layout

Organize functionality into modular plugins

Group your functionality as a plugin, unified in one directory (or code repository) which shares your plugin's name (with a "vim-" prefix or ".vim" suffix if desired). It should be split into `plugin/`, `autoload/`, etc. subdirectories as necessary, and it should delcar metadata in the addon-info.json format.

Functions

In the `autoload/` directory, defined with `[!]` and `[abort]`.

Autoloading allows functions to be loaded on demand, which makes startuptime faster and enforces function namespacing.

Script-local functions are welcome, but should also live in `autoload/` and be called by autoloaded functions.

Non-library plugins should expose commands instead of functions. Command logic should be extracted into functions and autoloaded.

`[!]` allows developers to reload their functions without complaint.

`[abort]` forces the function to halt when it encounters an error.

Commands

In the `plugin/commands.vim` or under the `ftplugin/` directory, defined without `[!]`.

General commands go in `plugin/commands.vim`. Filetype-specific commands go in `ftplugin/`.

Excluding `[!]` prevents your plugin from silently clobbering existing commands. Command conflicts should be resolved by the user.

Autocommands

Place them in `plugin/autocmds.vim`, within augroups.

Place all autocommands in augroups.

The augroup name should be unique. It should either be, or be prefixed with, the plugin name.

Clear the augroup with `autocmd!` before defining new autocommands in the augroup. This makes your plugin re-entrable.

Mappings

Place them in `plugin/mappings.vim`, using a prefix.

All key mappings sould be defined in `plugin/mappings.vim`.

Partial mappings (see `:help using-<Plug>`) should be defined in `plugin/plugs.vim`.

Settings

Change settings locally.

Use `:setlocal` and `&l:` instead of `:set` and `&` unless you have explicit reason to do otherwise.

Style

When in doubt, treat vimscript style like python style.

Whitespace

Similar to python.

- Use two spaces for indents
- DO not use tabs
- Use spaces around operators. This does not apply to arguments to commands.

  ```
  let s:variable = "concatenated " . "strings"
  command -range=% MyCommand
  ```

- Do not introduce trailing whitespace. You need not go out of your way to remove it. Trailing whitespace is allowed in mappings which prep commands for user input such as "`noremap <leader>gf :grep -f`".
- Restrict lines to 80 colmns wide
- Indent continued lines by four spaces
- Do not align argumentes of commands

```
command -bang MyCommand call myplugin#foo()
command MyCommand2 call myplugin#bar()
```

Naming

In general, use `plugin-names-like-this`, `FunctionNamesLiekThis`, `CommandNamesLiekThis`, `augroup_names_like_this`, `variable_names_like_this`.

Always prefer variables with their scope

plugin-names-like-this

Keep them short and sweet.

FunctionNamesLikeThis

Prefix script-local functions with `s:`.

Autoloaded functions may not have a scope prefix.

Do not create global functions. Use autoloaded functions instead.

CommandNamesLikeThis

Prefer succinct command names over common command prefixes.

variable__names__like__this

Augroup names count as variables for naming purposes.

Prefix all variables with their scope

- Global variables with `g:`
- Script-local variables with `s:`
- Function arguments with `a:`
- Function-local variables with `l:`
- Vim-predefined variables with `v:`
- Buffer-local variables with `b:`

`g:`, `s:`, and `a:` must always be used.

`b:` changes the variable semantics; use it when you want buffer-local semantics.

`l:` and `v:` should be used for consitency, future proofing, and to avoid subtle bugs. They are not strictly required. Add them in new code but don't go out of your way to add them elsewhere.