# Contributing Guidelines

## Welcome!

Thank you for considering contributing to the development of this project's

development and/or documentation. Just a reminder: if you're new to this project

or to OSS and want to find issues to work on, please check the following labels

on issues:

- [help wanted][helpwantedlabel]

- [docs][docslabel]

- [good first issue][goodfirstissuelabel]

[docslabel]: https://github.com/bats-core/bats-core/labels/docs

[helpwantedlabel]: https://github.com/bats-core/bats-core/labels/help%20wanted

[goodfirstissuelabel]: https://github.com/bats-core/bats-core/labels/good%20first%20issue

To see all labels and their meanings, [check this wiki page][labelswiki].

This guide borrows \*\*heavily\*\* from [@mbland's go-script-bash][gsb] (with some

sections directly quoted), which in turn was

drafted with tips from [Wrangling Web Contributions: How to Build

a CONTRIBUTING.md][moz] and with some inspiration from [the Atom project's

CONTRIBUTING.md file][atom].

[gsb]: https://github.com/mbland/go-script-bash/blob/master/CONTRIBUTING.md

[moz]: https://mozillascience.github.io/working-open-workshop/contributing/

[atom]: https://github.com/atom/atom/blob/master/CONTRIBUTING.md

[labelswiki]: https://github.com/bats-core/bats-core/wiki/GitHub-Issue-Labels

## Table of contents

\* [Contributing Guidelines](#contributing-guidelines)

\* [Welcome!](#welcome)

\* [Table of contents](#table-of-contents)

\* [Quick links <g-emoji alias="link" fallback-src="https://assets-cdn.github.com/images/icons/emoji/unicode/1f517.png" ios-version="6.0">?</g-emoji>](#quick-links-)

\* [Contributor License Agreement](#contributor-license-agreement)

\* [Code of conduct](#code-of-conduct)

\* [Asking questions and reporting issues](#asking-questions-and-reporting-issues)

\* [Updating documentation](#updating-documentation)

\* [Environment setup](#environment-setup)

\* [Workflow](#workflow)

\* [Testing](#testing)

\* [Coding conventions](#coding-conventions)

\* [Formatting](#formatting)

\* [Naming](#naming)

\* [Function declarations](#function-declarations)

\* [Variable and parameter declarations](#variable-and-parameter-declarations)

\* [Command substitution](#command-substitution)

\* [Process substitution](#process-substitution)

\* [Conditionals and loops](#conditionals-and-loops)

\* [Generating output](#generating-output)

\* [Gotchas](#gotchas)

\* [Open Source License](#open-source-license)

\* [Credits](#credits)

## Quick links &#x1f517;

- [Gitter channel ?][gitterurl]: These messages sync with the IRC channel

- [IRC Channel (#bats on freenode) ?][ircurl]: These messages sync with Gitter

- [README ?][README]

- [Code of conduct ?][CODE\_OF\_CONDUCT]

- [License information ?][LICENSE]

- [Original repository ?][repohome]

- [Issues ?][repoissues]

- [Pull requests ?][repoprs]

- [Milestones ?][repomilestones]

- [Projects ?][repoprojects]

[README]: https://github.com/bats-core/bats-core/blob/master/README.md

[CODE\_OF\_CONDUCT]: https://github.com/bats-core/bats-core/blob/master/docs/CODE\_OF\_CONDUCT.md

[LICENSE]: https://github.com/bats-core/bats-core/blob/master/LICENSE.md

## Contributor License Agreement

Per the [GitHub Terms of Service][gh-tos], be aware that by making a

contribution to this project, you agree:

\* to license your contribution under the same terms as [this project's

license][osmit], and

\* that you have the right to license your contribution under those terms.

See also: ["Does my project need an additional contributor agreement? Probably

not."][cla-needed]

[gh-tos]: https://help.github.com/articles/github-terms-of-service/#6-contributions-under-repository-license

[osmit]: #open-source-license

[cla-needed]: https://opensource.guide/legal/#does-my-project-need-an-additional-contributor-agreement

## Code of conduct

Harrassment or rudeness of any kind will not be tolerated, period. For

specifics, see the [CODE\_OF\_CONDUCT][] file.

## Asking questions and reporting issues

### Asking questions

Please check the [README][] or existing [issues][repoissues] first.

If you cannot find an answer to your question, please feel free to hop on our

[gitter][gitterurl] [![Gitter](https://badges.gitter.im/bats-core/bats-core.svg)](https://gitter.im/bats-core/bats-core?utm\_source=badge&utm\_medium=badge&utm\_campaign=pr-badge) or [via IRC (#bats on freenode)][ircurl].

### Reporting issues

Before reporting an issue, please use the search feature on the [issues

page][repoissues] to see if an issue matching the one you've observed has already

been filed.

### Updating or filing a new issue

#### Information to include

Try to be as specific as possible about your environment and the problem you're

observing. At a minimum, include:

#### Installation issues

1. State the version of Bash you're using `bash --version`

1. State your operating system and its version

1. If you're installing through homebrew, run `brew doctor`, and attach the

output of `brew info bats-core`

#### Bugs/usage issues

1. State the version of Bash you're using `bash --version`

1. State your operating system and its version

1. Command line steps or code snippets that reproduce the issue

1. Any apparently relevant information from the [Bash changelog][bash-changes]

[bash-changes]: https://tiswww.case.edu/php/chet/bash/CHANGES

Also consider using:

- Bash's `time` builtin to collect running times

- a regression test to add to the suite

- memory usage as reported by a tool such as

[memusg](https://gist.github.com/netj/526585)

### On existing issues

1. DO NOT add a +1 comment: Use the reactions provided instead

1. DO add information if you're facing a similar issue to someone else, but

within a different context (e.g. different steps needed to reproduce the issue

than previous stated, different version of Bash or BATS, different OS, etc.)

You can read on how to do that here: [Information to include][#information-to-include]

1. DO remember that you can use the \*Subscribe\* button on the right side of the

page to receive notifications of further conversations or a resolution.

## Updating documentation

We love documentation and people who love documentation!

If you love writing clear, accessible docs, please don't be shy about pull

requests. Remember: docs are just as important as code.

Also: \_no typo is too small to fix!\_ Really. Of course, batches of fixes are

preferred, but even one nit is one nit too many.

## Environment setup

Make sure you have Bash installed per the [Environment setup in the

README][env-setup].

[env-setup]: https://github.com/bats-core/bats-core/blob/master/README.md#environment-setup

## Workflow

The basic workflow for submitting changes resembles that of the [GitHub Git

Flow][github-flow] (a.k.a. GitHub Flow), except that you will be working with

your own fork of the repository and issuing pull requests to the original.

[github-flow]: https://guides.github.com/introduction/flow/

1. Fork the repo on GitHub (look for the "Fork" button)

1. Clone your forked repo to your local machine

1. Create your feature branch (`git checkout -b my-new-feature`)

1. Develop \_and [test](#testing)\_ your changes as necessary.

1. Commit your changes (`git commit -am 'Add some feature'`)

1. Push to the branch (`git push origin my-new-feature`)

1. Create a new [GitHub pull request][gh-pr] for your feature branch based

against the original repository's `master` branch

1. If your request is accepted, you can [delete your feature branch][rm-branch]

and pull the updated `master` branch from the original repository into your

fork. You may even [delete your fork][rm-fork] if you don't anticipate making

further changes.

[gh-pr]: https://help.github.com/articles/using-pull-requests/

[rm-branch]: https://help.github.com/articles/deleting-unused-branches/

[rm-fork]: https://help.github.com/articles/deleting-a-repository/

## Testing

- Continuous integration status for Linux and macOS: [![Build Status on Travis](https://travis-ci.org/bats-core/bats-core.svg?branch=ci-configs)](https://travis-ci.org/bats-core/bats-core)

- Continuous integration status for Windows: [![Build status on AppVeyor](https://ci.appveyor.com/api/projects/status/tokwm9t9jp5fe7af?svg=true)](https://ci.appveyor.com/project/bats-core/bats-core)

## Coding conventions

- [Formatting](#formatting)

- [Naming](#naming)

- [Variable and parameter declarations](#variable-and-parameter-declarations)

- [Command substitution](#command-substitution)

- [Conditions and loops](#conditionals-and-loops)

- [Gotchas](#gotchas)

### Formatting

- Keep all files 80 characters wide.

- Indent using two spaces.

- Enclose all variables in double quotes when used to avoid having them

interpreted as glob patterns (unless the variable contains a glob pattern)

and to avoid word splitting when the value contains spaces. Both scenarios

can introduce errors that often prove difficult to diagnose.

- \*\*This is especially important when the variable is used to generate a

glob pattern\*\*, since spaces may appear in a path value.

- If the variable itself contains a glob pattern, make sure to set

`IFS=$'\n'` before using it so that the pattern itself and any matching

file names containing spaces are not split apart.

- Exceptions: Quotes are not required within math contexts, i.e. `(( ))` or

`$(( ))`, and must not be used for variables on the right side of the `=~`

operator.

- Enclose all string literals in single quotes.

- Exception: If the string contains an apostrophe, use double quotes.

- Use quotes around variables and literals even inside of `[[ ]]` conditions.

- This is because strings that contain '[' or ']' characters may fail to

compare equally when they should.

- Exception: Do not quote variables that contain regular expression patterns

appearing on the right side of the `=~` operator.

- \_Only\_ quote arguments to the right of `=~` if the expression is a literal

match without any metacharacters.

The following are intended to prevent too-compact code:

- Declare only one item per `declare`, `local`, `export`, or `readonly` call.

- \_Note:\_ This also helps avoid subtle bugs, as trying to initialize one

variable using the value of another declared in the same statement will

not do what you may expect. The initialization of the first variable will

not yet be complete when the second variable is declared, so the first

variable will have an empty value.

- Do not use one-line `if`, `for`, `while`, `until`, `case`, or `select`

statements.

- Do not use `&&` or `||` to avoid writing `if` statements.

- Do not write functions entirely on one line.

- For `case` statements: put each pattern on a line by itself; put each command

on a line by itself; put the `;;` terminator on a line by itself.

### Naming

- Use `snake\_case` for all identifiers.

### Function declarations

- Declare functions without the `function` keyword.

- Strive to always use `return`, never `exit`, unless an error condition is

severe enough to warrant it.

- Calling `exit` makes it difficult for the caller to recover from an error,

or to compose new commands from existing ones.

### Variable and parameter declarations

- \_Gotcha:\_ Never initialize an array on the same line as an `export` or

`declare -g` statement. See [the Gotchas section](#gotchas) below for more

details.

- Declare all variables inside functions using `local`.

- Declare temporary file-level variables using `declare`. Use `unset` to remove

them when finished.

- Don't use `local -r`, as a readonly local variable in one scope can cause a

conflict when it calls a function that declares a `local` variable of the same

name.

- Don't use type flags with `declare` or `local`. Assignments to integer

variables in particular may behave differently, and it has no effect on array

variables.

- For most functions, the first lines should use `local` declarations to

assign the original positional parameters to more meaningful names, e.g.:

```bash

format\_summary() {

local cmd\_name="$1"

local summary="$2"

local longest\_name\_len="$3"

```

For very short functions, this \_may not\_ be necessary, e.g.:

```bash

has\_spaces() {

[[ "$1" != "${1//[[:space:]]/}" ]]

}

```

### Command substitution

- If possible, don't. While this capability is one of Bash's core strengths,

every new process created by Bats makes the framework slower, and speed is

critical to encouraging the practice of automated testing. (This is especially

true on Windows, [where process creation is one or two orders of magnitude

slower][win-slow]. See [bats-core/bats-core#8][pr-8] for an illustration of

the difference avoiding subshells makes.) Bash is quite powerful; see if you

can do what you need in pure Bash first.

- If you need to capture the output from a function, store the output using

`printf -v` instead if possible. `-v` specfies the name of the variable into

which to write the result; the caller can supply this name as a parameter.

- If you must use command substituion, use `$()` instead of backticks, as it's

more robust, more searchable, and can be nested.

[win-slow]: https://rufflewind.com/2014-08-23/windows-bash-slow

[pr-8]: https://github.com/bats-core/bats-core/pull/8

### Process substitution

- If possible, don't use it. See the advice on avoiding subprocesses and using

`printf -v` in the \*\*Command substitution\*\* section above.

- Use wherever necessary and possible, such as when piping input into a `while`

loop (which avoids having the loop body execute in a subshell) or running a

command taking multiple filename arguments based on output from a function or

pipeline (e.g. `diff`).

- \*Warning\*: It is impossible to directly determine the exit status of a process

substitution; emitting an exit status as the last line of output is a possible

workaround.

### Conditionals and loops

- Always use `[[` and `]]` for evaluating variables. Per the guideline under

\*\*Formatting\*\*, quote variables and strings within the brackets, but not

regular expressions (or variables containing regular expressions) appearing

on the right side of the `=~` operator.

### Generating output

- Use `printf` instead of `echo`. Both are Bash builtins, and there's no

perceptible performance difference when running Bats under the `time` builtin.

However, `printf` provides a more consistent experience in general, as `echo`

has limitations to the arguments it accepts, and even the same version of Bash

may produce different results for `echo` based on how the binary was compiled.

See [Stack Overflow: Why is printf better than echo?][printf-vs-echo] for

excruciating details.

[printf-vs-echo]: https://unix.stackexchange.com/a/65819

### Signal names

Always use upper case signal names (e.g. `trap - INT EXIT`) to avoid locale

dependent errors. In some locales (for example Turkish, see

[Turkish dotless i](https://en.wikipedia.org/wiki/Dotted\_and\_dotless\_I)) lower

case signal names cause Bash to error. An example of the problem:

```bash

$ echo "tr\_TR.UTF-8 UTF-8" >> /etc/locale.gen && locale-gen tr\_TR.UTF-8 # Ubuntu derivatives

$ LC\_CTYPE=tr\_TR.UTF-8 LC\_MESSAGES=C bash -c 'trap - int && echo success'

bash: line 0: trap: int: invalid signal specification

$ LC\_CTYPE=tr\_TR.UTF-8 LC\_MESSAGES=C bash -c 'trap - INT && echo success'

success

```

### Gotchas

- If you wish to use command substitution to initialize a `local` variable, and

then check the exit status of the command substitution, you \_must\_ declare the

variable on one line and perform the substitution on another. If you don't,

the exit status will always indicate success, as it is the status of the

`local` declaration, not the command substitution.

- To work around a bug in some versions of Bash whereby arrays declared with

`declare -g` or `export` and initialized in the same statement eventually go

out of scope, always `export` the array name on one line and initialize it the

next line. See:

- https://lists.gnu.org/archive/html/bug-bash/2012-06/msg00068.html

- ftp://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-025

- http://lists.gnu.org/archive/html/help-bash/2012-03/msg00078.html

- [ShellCheck](https://www.shellcheck.net/) can help to identify many of these issues

## Open Source License

This software is made available under the [MIT License][osmit].

For the text of the license, see the [LICENSE][] file.

## Credits

- This guide was heavily written by BATS-core member [@mbland](https://github.com/mbland)

for [go-script-bash](https://github.com/mbland/go-script-bash), tweaked for [BATS-core][repohome]

- Table of Contents created by [gh-md-toc](https://github.com/ekalinin/github-markdown-toc)

- The [official bash logo](https://github.com/odb/official-bash-logo) is copyrighted

by the [Free Software Foundation](https://www.fsf.org/), 2016 under the [Free Art License](http://artlibre.org/licence/lal/en/)

[repoprojects]: https://github.com/bats-core/bats-core/projects

[repomilestones]: https://github.com/bats-core/bats-core/milestones

[repoprs]: https://github.com/bats-core/bats-core/pulls

[repoissues]: https://github.com/bats-core/bats-core/issues

[repohome]: https://github.com/bats-core/bats-core

[osmit]: https://opensource.org/licenses/MIT

[gitterurl]: https://gitter.im/bats-core/bats-core

[ircurl]: https://kiwiirc.com/client/irc.freenode.net:+6697/#bats