Commit Message Guidelines

About

How to better format your commit messages to meet the standards of the FreeSWITCH project.

Guidelines for a Good Commit

To the extent possible and appropriate, address only one issue per commit. When we review your commit, anything that doesn't need to be there will only create confusion.

This means that, for example, unrelated refactoring or whitespace cleanups should generally happen in separate commits. Whitespace cleanup commits should not change anything other than whitespace, and refactoring commits should strive to preserve identical behavior.

However, don't go overboard. A commit should do some identifiable thing completely. If you're adding a new module, the build changes for that module should go in the commit that adds the module itself. If you're adding a feature, the feature should work after applying that commit.

We don't need to see your missteps and corrections. Use `git rebase -i` to squash those out of your history before submitting the commit series to us. It should look like you got everything right the first time. Eg.:

```
git rebase -i <last_upstream_commit>
```

Verify that each diff is correct and minimal, and that your git author name is correct and complete:

```
git log -p
```

Writing a Good Commit Message

Your commit message consists of two parts: the subject and the body.

The subject is like the subject in an email message. It should be short -- typically less than 50 characters -- and it should concisely describe the purpose or effect of your change. If you're having a difficult time writing a short subject for your commit, perhaps your commit should be broken into smaller separate commits.

The commit body can be longer and can consist of multiple paragraphs. The text of the body should be hard wrapped to 68-72 characters. (In Emacs you can hard wrap text with `M-q`.)

When writing the commit body, describe in detail the problem that your commit aims to solve, how your commit solves the problem, and any changes in behavior that result from your change, such as new variables, command flags, or breaks in backward compatibility.

Your commit message should be written in the present tense in imperative style. Your message should talk about what the patch *does*, not what you *did* to write it.

The commit subject is the first line of your commit message, then there is an empty line, then your commit body starts. A good commit message might look like this:

```
commit b5c64234ea5d4417abe02b282d6f41c019f2252f
 Author: John Doe <johndoe@example.com>
 Date: Tue Jan 19 03:14:07 2014 +0000

 FS-XXXX: [mod_foo] Add frobinator support to mod_sofia

 #comment Without proper frobinator support users had to make multiple calls
 to shell scripts to do the sort of frobbing needed in high call
 volume environments.

 With this change, we now link to libfrob and support the IETF
 draft-cross-voip-frobbing API.

 After appropriate amounts of frobbing have been done, a new variable
 `frobbing_done` is set in the caller's channel.
```

Patches Related to JIRA Issues

When your patch is related to an issue logged in JIRA, add the identifier for the issue (e.g. FS-XXXX) to the body of your commit message at the beginning of a line, typically the last line or just before "Signed-off-by:" and "Thanks-to:" lines. This helps our JIRA bot do useful things by relating the commit to the issue.

If you believe your patch resolves the issue in question, follow the issue number with a space and the "#resolve" directive as in the example above.

Avoid Merges

When you've created a local git branch to make and test your changes, it can be tempting to merge that branch periodically against our git HEAD, particularly if the branch lingers for some time. Don't do this. Random "update branch to master" merges make our history hard to understand and make it more difficult to isolate regressions with `git-bisect`. Instead, please rebase your branch onto our tree before submitting the commits to us:

```
git fetch && git rebase origin/master
```

If you already ran `scripts/setup-git.sh`, `git pull` will be equivalent to the command above.

Do As We Say...

When you look in our git history, you'll find not all commits follow the guidelines here. Don't be fooled by this. These guidelines are what we want, and your commits should follow them.

It's always difficult to counsel, "do as we say and not as we do," but the truth is that the format of your commits will be held to a different standard than the commits of people who have written the majority of the code in FS. For one thing, your commits will be reviewed, so following a careful format helps us to review and merge your patches quickly and efficiently.

We want a clean and sensible git history, and over time more contributors will be following the guidelines here.