SKXXX - How to Write a Great Git Commit Message

Purpose - Standardize Git commit messages with descriptive messages for future selves

Policy -

1. Separate subject from body

```
$ man git commit
```

> Though not required, it's a good idea to begin the commit message with a single short (less than 50 character) line summarizing the change, followed by a blank line and then a more thorough description. The text up to the first blank line in a commit message is treated as the commit title, and that title is used throughout Git.

There will be times where a straightforward change will not require both a subject and a body. For example:

```
Commit Message: Change user login icons
```

In the above example, a `single short (less than 50 character) line summarized the change`. If the reader needs additional information on the change, commands such as `git show`, `git diff`, or `git log -p` can be used.

A one-lined subject can be committed with:

```
$ git commit -m "Change user login icons"
```

However, there may be commits that benefit from a more detailed explanation.

```
Update passwords to SHA-512

Utilized the python crypt library to generate a SHA-512 hashed
password from clear text. This encryption algorithm is salted for
additional security.
```

There are two methods to creating a Git commit message with a subject and a body.

1.

```
$ git commit
```

```
{Should be in the git commit text editor screen; start appending text}
Update passwords to SHA-512

Utilized the python crypt library to generate a SHA-512 hashed
password from clear text. This encryption algorithm is salted for
additional security.
```

2.

```
$ git commit -m "Update passwords to SHA-512" \
> -m "Utilized the python crypt library to generate a SHA-512 hashed
> password from clear text. This encryption algorithm is salted for
> additional security
```

The `git commit` option is more elegant, as it opens up your default text editor.

2. Limit subject line to 50 characters

As stated in the previous rule, a concise 50-character summary of the change is a rule of thumb, not a hard limit. If there is difficulty in coming up with a concise summary, it could mean that there are too many changes going on at once. If this is the case, split up the commits into multiple commits. Github will truncate any subject line greater than 69 characters, so consider the subject line to be between 50-69 characters.

3. Capitalize the subject line

```
Update passwords to SHA-512 <-- Correct
update passwords to SHA-512 <-- Incorrect
```

4. Do not end the subject line with a period

Consider the subject line to be the title of your commit message. Title's do not have periods.

```
Update the passwords to SHA-512 <-- Correct
Update the passwords to SHA-512. <-- Incorrect
```

5. Prefix subject line with a gist

Appending a gist before the subject line provides the reader a general idea of what area was changed.

```
Security: Update the passwords to SHA-512
```

A few other examples taken directly from Linus Torvalds' linux repository:

```
dma-mapping: consolidate dma_set_mask
zswap: update docs for runtime-changeable attributes
zsmalloc: do not take class lock in zs_shrinker_count()
```

6. Use the imperative mood in the subject line

Imperative mood - Giving a command or instruction

- Change the lightbulb
- Create a mess
- Delete the paragraph
- Update passwords to SHA-512

The subject lines above are all in the **present tense**, so commits like the following would be incorrect:

- Fixed bug that prevented payments
- Testing user login credentials
- Just changed the index page

A commit subject line should be able to complete the following sentence:

If applied, this commit will *subject line*

If applied, this commit will *Update passwords to SHA-512*.

The imperative mood is only necessary for the subject line, so the body does not have to follow this rule.

7. Avoid meaningless content

Do not include the **date** in the commit subject or body. Such information is already provided by Git when you commit.

```
Complete the warehouse report on 09-09-2015 <-- Incorrect
Complete the warehouse report <-- Correct
```

At the same time, these words are taboo when you commit:

1. test
2. Test
3. tesT
4. TEST

Development, Testing, or Production quality code should never contain commits with the aforementioned words. The only exception to this rule is if you `git rebase` and cover up your tracks with a new commit message.

8. Wrap the body at 72 characters

Git does not automatically wrap text at 72 characters, so this will be at your own discretion. Text editors such as VI/VIM will show the number of characters per line. This will allow Git to perform proper indentation while keeping everything under 80 characters overall.

The carriage return [ENTER key] will create a new line and wrap the text.

For example:

```
This is the subject of the git commit message [PRESS ENTER TWICE TO BEGIN THE BODY]

This is the body of the git commit message. This is going to be [PRESS ENTER]
filled up with Lorem Ipsum text. Lorem ipsum dolor sit amet, [PRESS ENTER]
consectetur adipiscing elit. Donec a diam lectus. Sed sit amet ipsum [PRESS ENTER]
mauris. Maecenas congue ligula ac quam viverra nec consectetur ante [PRESS ENTER]
hendrerit. [PRESS ENTER TWICE TO BEGIN ANOTHER BODY]

Vivamus fermentum semper porta. Nunc diam velit, adipiscing ut [PRESS ENTER]
tristique vitae, sagittis vel odio. Maecenas convallis ullamcorper [PRESS ENTER]
ultricies. Curabitur ornare, ligula semper consectetur sagittis, [PRESS ENTER]
nisi diam iaculis velit, id fringilla sem nunc vel mi.
```

9. Use the body to explain what and why versus how

For the most part, you can omit details about how a change has been made. This should be self-explanatory in the code or source comments. The important part here is to focus on the following:

1. Behavior of the code before the change, and what was wrong with the behavior previously
2. Behavior of the code after the change
3. Why you decided to solve it the way you did

10. Branch naming conventions

When feature branches are merged into master with the no fast-forward (--no-ff) option, it creates a commit with the feature branch's name. Since commits will include the feature branch's name, it is also important to lay out a standardized branch naming convention.

Examples of merging branches into master:

```
Commit Message: Merge branch 'testing-on-live' into master
Commit Message: Merge branch 'test' into master
```

Examples of a standarized branch naming system:

1. **proj** - Extensive assignment with a due date
2. **feat** - Feature being added or expanded
3. **bug** - Bug fix
4. **junk** - Disposable branch created for experiment

[[**Notice in example 4, I did not use the word *test***]]

Examples of branch names

1. proj/tidy-up-connect
2. proj/warehouse-report
3. feat/reset-password
4. feat/electronic-picklists
5. junk/foo
6. junk/bar

11. References

If applicable, include a URL to the end result of a task or project in the body of the Git commit. If the task or project results in multiple URLs, reference all of them.

```
Security: Update passwords to SHA-512

Utilized the python crypt library to generate a SHA-512 hashed
password from clear text. This encryption algorithm is salted for
additional security.

URL: nagios.sks.com/nagios
```

12. Guidelines

Here are links to some successful repositories:

1. GitHub: github.com/git/git/commits/master
2. Linux: github.com/torvalds/linux/commits/master
3. Spring Boot: github.com/spring-projects/spring-boot/commits/master