NexusWide Project Managemet Guidelines

Nima Bavar

March 28, 2024

©opyright

Dedicated to all the computation enthusiastics

and

FREE AND OPEN SOURCE SOFTWARE producers around the globe.

Contents

1   Conventions

Numerous conventions are globally used throughout this document in order to draw attention or to cite particular meanings, which are stated in the posterior table:

Emphasization	definition
⚠ Text with Tribar	Nexusrule
⟩ _e.x :_ **: (Red bold text with angle sign)**	Rule example
SMALL CAPS TEXT	Name
Italic text	Important Signification

The acknowledgment of the aforementioned elements shall be enough for you to be prepared to start reading this documentation.

Do not hesitate to reach out to this table anytime you have forgave the reason behind a specific emphasizing regularity.

2 Preface

DURABILITY and CONSISTENCY are the two most prominent chunk of a bested project management course, and no project is capable of achieving satisfactory outcomes without proper administration.

One branch of behavior that can further advance durability by obligating consistency in the best feasible manner, is including consequential regulations and laws, which is preserved by affixing sharp style guides to the association.

Thereby to achieve pleasing result in order to avail the esteem and sagacious community of FOSS[1] with our works of knowledge and researches, and most essentially, provide value to our respected readers and users, we shall possess our own particular style guides for the contributors to adhere.

The foundation of the following pertinent standards and rules, was gathered by the acknowledgment of the fore-mentioned principles.

We *oblige* all of our contributors to adhere to them, in order to acquire accommodate flaw-free products.

Please reach out to the succeeding pages, and interpret them precisely.

---

[1]Free and Open and Source Software.

3   Project Management Guidelines

Comprised throughout numerous well-known projects of the present era, DVCS^2 is one of most comfortable branches of VCS^3s.

Because of its distributed bearing, the altering of the project is inexpensively feasible in the offline status.
This actuality stems from the fact that each individual is permitted to conduct his\her own local copy of the project, and as labeled in the realm of Version Control, *push* the developed Revisions[4] to a central cloud repository once a satisfied condition of the project is prevailed.

Thence, it is the default VCS which is utilized by the NexusWide organization in the field of project management.

This section elucidates all of the guidelines and practices which are employed throughout the organization in order to take full advantage of the DVCS capabilities.

---

[2]Distributed Version Control System.
[3]Version Control System.
[4]Also known as Commits.

3.1 List of Rules

⚠ All NexusWide project repositories MUST contain a DOCS directory.

⚠ All NexusWide project repositories MUST have a README.md file located in their DOCS directory which follows the Markdown formatting and style guidelines.

⚠ All NexusWide project repositories MUST contain a SECURITY POLICY file.

⚠ All of the following segments: SECURITY POLICY, CODE OF CONDUCT, CONTRIBUTING GUIDELINES, and TERMS OF SERVICE MUST be located within the DOCS directory.

⚠ Only the LICENSE file is permitted to be located in the main directory of the repository.

⚠ All NexusWide owned repositories MUST be licensed under (Creative Commons Zero v1.0 Universal).

⚠ Force commits are not allowed in NexusWide repositories.

⚠ Fast-forward MERGEs are not allowed in NexusWide repositories.

⚠ All versions of the project repository MUST be controlled under semantic versioning (HTTPS://SEMVER.ORG/).

⚠️   The version numbering of the projects MUST be split into two phases:
(pre_release phase, release phase).

⚠️   During the pre release phase all changes in the development process are a
part of the version numbering.

⚠️   The release version numbering MUST govern only the user side features.

⚠️   When a project reaches the release phase, a (changelog.md) file MUST
be implemented in the repository (docs) directory.

⚠️   The release phase version numbering is only mentioned in the
(changelog.md) file.

⚠️   Release versioning and development versioning are seperate concepts, do
not attempt to increment the release version number while proposing updates to
the back end side of the projects.

⚠️   Local branches MUST have their name set as the part of the API they are
going to affect, followed by
the branch number (The branch number must be used in order to avoid conflicts
with the previous remote branches of the same name).
⟩ _e.x :_ **(documentations1)**

⚠   Contributors MUST always create a PULL REQUEST and branch before attempting to make their local changes and submit an update.

⚠   Remote branches MUST be deleted after they have been MERGEd into the master (Main.Project) branch in any form.

⚠   PULL REQUEST titles MUST refer to the fix, feat or MAJOR change commit message which is expected to be the outcome after all the ISSUE to-do tasks are finished.

⚠   The convention of naming PULL REQUESTs is the same as commit messages.

⚠   All PULL REQUESTs MUST have at least one comment before being MERGEd.

⚠   All PULL REQUESTs MUST contain a well formatted description comment.

⚠   The PULL REQUEST description comment MUST always be the first comment.

⚠   Contributors MUST NOT directly list any of the (fix) or (feat) changes in the description comment of a PULL REQUEST (using bulleted or numbered lists, ETC) They MUST be self explanatory in the commit messages.

⚠️   PULL REQUESTs MUST NOT contain a to do list, as that would make the existence of ISSUEs purposeless.

⚠️   The PULL REQUEST description comment MUST start with a header containing the update name tag
(this is not the same as the PULL REQUEST title, it is any name tag that you would want to assign) followed by the version number.
⟩ *e.x* : **(Presuming changes in documentations: UDock | 1.0.0)**

⚠️   The PULL REQUEST description comment MUST contain four sections:
What has changed (API changes)?
What were the reasons behind the changes?
What results are expected?
Which ISSUE notes are affected by this update?

⚠️   The SIGNED OFF BY section MUST be the last section of the description comment and refer to the contributors who parted in the update.

⚠️   ISSUE references in PULL REQUESTs MUST follow the following format:
(ISSUE Note | ISSUE Number).

⚠️   If a update has been rejected, The branch related to the update MUST be deleted, the PULL REQUEST MUST be closed and labled as REJECTED
(The description comment MUST stay as is).

⚠️   All code MUST be reformatted and pass all the tests adjusted by the GITHUB automation flow before being MERGEd.

⚠️   ISSUE names MUST follow the corresponding regular expression:
(ISSUE Title | ISSUE numeral Count).
⟩ *e.x* : **: (fix(docs): remove harsh rules | 3)**

⚠️   All ISSUEs MUST contain at least 1 comment.

⚠️   The first comment of an ISSUE MUST be the details comment.

⚠️   The ISSUE details comment MUST be formatted in induce to providing a response to the following topics:
What is the assignment?
What are the steps to accomplish?
Why is this fixture necessary?
What is the expected outcome?

⚠️   The ISSUE details comment MUST contain a contributor name at its last section, in order to assign the contributor to the cited task.

⚠️   The GIT commit messages are formatted by the
https://www.conventionalcommits.org/en/v1.0.37. conventions.

⚠️   GIT commits MUST be as small and independent as possible, do not concatenate two possible commits together.

⟩ _e.x_ : **(fix(docs): refactor grammar and add header | is a bad commit message.)**

⚠️   Push changes as often as possible, all the changes to the project (including the ones that are rejected or/and aborted) MUST be recorded.

⚠️   All suspended projects (those that don't receive any new update for a designated period of time) MUST be archived and followed by a note text at the tail of their README file, elucidating the fact that the project have been archived.

⚠️   All GIT commit messages MUST have their branch name and the commit number (the number of times that the contributor have committed to a branch) as their footer.

⟩ _e.x_ : **(style_guide: 3)**

⚠️   In the context of combining branches, use MERGE instead of REBASE.

⚠️   While performing any type of merging operation, adhere to the same guidelines as commit messages and provide the reason behind the MERGE.

⟩ _e.x_ : **(fetch(merge): fix conflict with origin)**

⚠   All remote repository MERGE messages MUST be the same as the title of their PULL REQUEST.

⚠   All remote repository MERGE messages MUST have the URL of their PULL REQUEST at their description section.

⚠   All MERGE messages (Including the remote ones) MUST have the MERGE pseudo-environment count as their footer: (The number of times that a MERGE operation was conducted in the repository)

⟩ _e.x :_ **: (merge: 4)**

⚠   All NexusWide repositories MUST have a Kanban board project derived
from the NexusWide main Kanban board template.

⚠   The NexusWide Kanban board (in_progress and in_review)
columns MUST be limited to one card only.
(Only one update topic is allowed to be worked on at a time.)

⚠   The NexusWide Kanban board (wont_fix, ready, done) columns
MUST NOT introduce any card amount limitations.

⚠   All NexusWide card items MUST be converted to issues when they
reach the (ready) state.

⚠   All NexusWide Kanban card names MUST adhere to the NexusWide
Git commit message guidelines.

⚠   All NexusWide Kanban card names MUST be the name of the issue
which is going to be produced after the item have reached
the (in_progress) state, omitting the issue number.

⚠️   The ISSUEs generated from a Kanban card MUST have the same name as the card.

⚠️   All NEXUSWIDE Kanban cards MUST have a description section aligned.

⚠️   All NEXUSWIDE Kanban card descriptions MUST adhere to the ISSUE details comment formatting guidelines.

⚠️   All NEXUSWIDE Kanban card descriptions MUST be the description of the ISSUE which is going to be produced after the item have reached the (IN_PROGRESS) state.

⚠️   The ISSUEs generated from a Kanban card MUST have the same details comment content as the card description.

⚠️   All NEXUSWIDE Kanban card items MUST be moved to the (DONE) column after the corresponding PULL REQUEST have been MERGEd.

⚠️   All NEXUSWIDE Kanban card items MUST be moved to the (WONT_FIX) column if the update have been rejected.

⚠️   All NEXUSWIDE Kanban card items after/on the (READY) column MUST be assigned a start date.

⚠️   All NEXUSWIDE Kanban card items after/on the (READY) column MUST be assigned a due date.

⚠   All NEXUSWIDE Kanban card items after/on the (READY) column MUST be assigned to a contributor (assignee).

⚠   All NEXUSWIDE Kanban card items after/on the (READY) column MUST be prioritized and labeled based on the tags offered by the NEXUSWIDE main Kanban template.

⚠   All NexusWide Kanban card items after/on the (READY) column MUST be measured in size and labeled based on the tags offered by the NexusWide main Kanban template.

3.2 Examples

Figure 1: Example of NEXUSWIDE name conventions.

Figure 2: Example of NEXUSWIDE description comment conventions.

4   Programming Guidelines

4.1 Preamble

Due to the modular form of a multifarious amount of programming languages, including LaTeX and Python[5] several people are enabled to work on the same application or feature concurrently, each with their own considerations and mentality.

in such circumstances, the absence of a comparable programming style would result in numerous time consuming conversations and courses, in order to apprehend the workflow of a codebase with differentiations at each of its sections.

This operation consumes the invaluable time which could be invested into effectively *reviewing* and *enhancing* the codebase, therefore results in insufficient outcomes.

The above-mentioned illustrations are the results of the lack of consistent regulations between the individuals thus they become even more prominent when a large number of entities are involved in a project and tasks are broken into countless fragments.

---

[5]Languages that are generally used in the NexusWide corporation for generating documentation files and automating tasks.

Fortunately, all of the aforementioned dilemmas can be prevented by decreasing the amount of creativity arrayed into the programming style of each person by developing a default guide for the populace in the organization to adhere, so that amount of creativity can be directed into the actual development process of the working industry.

By endorsing this accuracy, we have conducted our own programming style guides with the aim of avoiding the referred considerations, which can be read at the following pages.

4.2 List of Rules

⚠   All programmers who use python, MUST follow the PEP8 style guide.

⚠   Programmers who use other languages, MUST follow their language style guide, but MUST still adhere to the blank line rules of PEP8.

⚠   All codes MUST be reformatted by the psf/black formatter before being merged to the main branch.

⚠   All codes MUST have *at least* 80 percent of unit test coverage acceptance before being merged to the master (Main.Project branch)