guidelines

Author: Adrian DC

Description: Guidelines applicable to all RadianDevCore projects and applicable projects

Date: 28/02/2025

Version: `1.0.0-17-g70ae3d4`

▸ Usage

Guidelines shared for all RadianDevCore projects and for applicable projects.

Documentation: https://radiandevcore.gitlab.io/wiki/guidelines

---

Table of contents

Documentation

Guidelines are available for the following topics:

- **Git:**
  - **Commands:** Important Git commands reference guide
  - **Commits:** Conventional Commits variant reference guide
- **GitLab:**
  - **GitLab Issues:** Issues labels and issue boards guide
- **Terminals:**
  - **.bashrc:** Bash configurations and reference guide

Compatibility

Projects compatible with RadianDevCore `guidelines` can use this badge to ease things for developers, both as an indicator and a documentation shortcut button :

> 
>

Badge in Markdown

```
[![radiandevcore-guidelines](https://img.shields.io/badge/radiandevcore-guidelines-brightgreen?logo=gitlab)]
(https://radiandevcore.gitlab.io/wiki/guidelines)
```

Badge in HTML

```
<a href="https://radiandevcore.gitlab.io/wiki/guidelines"><img src="https://img.shields.io/badge/randiandevcore-
guidelines-brightgreen?logo=gitlab" alt="radiandevcore-guidelines" style="max-width:100%;"></a>
```

Dependencies

- commitizen: Simple commit conventions for internet citizens
- pre-commit: A framework for managing and maintaining pre-commit hooks
- pre-commit-crocodile: Git hooks intended for developers using pre-commit

---

References

- .gitlab-ci.yml: GitLab CI/CD Pipeline Configuration Reference
- conventionalcommits: Conventional Commits specification for commit messages
- gcil: Launch .gitlab-ci.yml jobs locally
- git-cliff: CHANGELOG generator
- mkdocs: Project documentation with Markdown
- mkdocs-exporter: Exporter plugin for mkdocs documentation
- mkdocs-material: Material theme for mkdocs documentation

guidelines

Author: Adrian DC

Description: Guidelines applicable to all RadianDevCore projects and applicable projects

Date: 28/02/2025

Version: `1.0.0-17-g70ae3d4`

▸ Commands

Creation & Clone

Clone an existing remote repository locally:

```
git clone protocol://user@domain/repo.git
```

Create a empty local repository:

```
git init
```

Local Changes

Display modified files in the local repository:

```
git status
```

Display changes made to tracked files:

```
git diff
```

Display changes summary of tracked files:

```
git diff --name-status
```

Display changes made to a single file only:

```
git diff <FILE>
```

Add files to the staging area:

```
git add -v <FILE1> [FILE2 ...]
```

Add all modified files to the staging area (tracked + untracked):

```
git add -Av
```

Add only a portion of changes from a file to the staging area:

```
git add -p <FILE>
```

- **y :** Yes, add the selected chunk to the staging area
- **n :** No, do not add
- **a :** Yes, add all remaining chunks in the current file
- **d :** No, do not add any remaining chunks in the current file
- **s :** Split the modification group into individual changes
- **e :** Manually edit the selected chunk
- **q :** Quit adding

Commits

Commit the contents of the staging area:

```
git commit
```

Commit the contents of the staging area with a sign-off signature:

```
git commit -s
```

Amend current commit with staged changes and modify the message:

```
git commit --amend
```

Amend current commit with staged changes without message edition:

```
git commit --amend --no-edit
```

Stash

Stash the changes made to sources and reset to HEAD:

```
git stash
```

Manually stash changes chunks made to sources:

```
git stash -p
```

- **y :** Yes, stash this hunk
- **n :** No, do not stash this hunk
- **q :** Quit, do not stash any remaining hunk
- **a :** All, stash this hunk and all later hunks in the file
- **d :** Deny, do not stash the remaining hunks in the file
- **e :** Edit, manually edit the current hunk

Restore the last stash of changes to the working directory:

```
git stash pop
```

Remove all stashed entries:

```
git stash clear
```

Commits History

Display commits detailed history:

```
git log
```

Display commits simple history:

```
git log --pretty=oneline --abbrev-commit
```

Display modifications history for a single file (`--` if the file is missing)

```
git log -p -- <FILE>
```

Display authors of changes to a file:

```
git blame <FILE>
```

Show local commits history in visual tools:

```
git fetch --all
tig --all --
gitk --all -- &
```

Branches & Tags

Display all existing branches and their HEAD:

```
git branch --all --verbose
```

Extract and point HEAD to a branch:

```
git checkout <BRANCH>
```

Create a new local branch from current HEAD or a specific commit:

```
git branch <NEW_BRANCH>
git branch -f <BRANCH> <SHA1>
```

Delete a local branch:

```
git branch -d <BRANCH> # Use -D if the branch is not merged
```

Create a tag pointing to current HEAD commit:

```
git tag -m '<TAG_MESSAGE>' <TAG_NAME>
```

Remotes

List existing remotes:

```
git remote -v
```

List information of a specific remote:

```
git remote show <REMOTE>
```

Add a new remote:

```
git remote add <SHORT_NAME> <URL>
```

Update & Publish

Fetch all changes from a remote without changing the local repo:

```
git fetch --all <REMOTE>
```

Fetch changes from a remote branch and merge them locally:

```
git pull <REMOTE> <BRANCH>
```

Fetch changes from a remote branch and rebase on them locally:

```
git pull -r <REMOTE> <BRANCH>
```

Publish the content of a local branch to a remote:

```
git push <REMOTE> <BRANCH>
```

Publish the content of HEAD to a remote branch:

```
git push <REMOTE> HEAD:refs/heads/<BRANCH>
```

Overwrite the content of a remote branch to HEAD:

```
git push -f <REMOTE> HEAD:refs/heads/<BRANCH>
```

Delete a branch on a remote:

```
git branch -dr <REMOTE/BRANCH>
```

Rebase & Merge

Rebase HEAD onto a branch:

```
git rebase <BRANCH>
```

Abort an active rebase:

```
git rebase --abort
```

Continue a rebase after resolving conflicts:

```
git rebase --continue
```

Skip a commit without resolving conflicts:

```
git rebase --skip
```

Choose the commits that should be rebased interactively:

```
git rebase -i <BRANCH>
```

- **p, pick :** Use commit
- **r, reword :** Use commit, but edit the commit message
- **e, edit :** Use commit, but stop for amending
- **s, squash :** Use commit, but merge into previous one
- **f, fixup :** Use commit, but patch into previous one
- **d, drop or `# ...` :** Remove the commit

Merge a branch into the HEAD:

```
git merge <BRANCH>
```

Open the merge tool configured in git (if configured first):

```
git mergetool
```

Undo

Reset the content of the staging area:

```
git reset
```

Reset the project to the state of the current commit:

```
git reset --hard HEAD
```

Reset the project to the previous commit's state or a specific commit:

```
git reset --hard HEAD^
git reset --hard <SHA1>
```

Reset the project to the state of the last fetched commit:

```
git reset --hard FETCH_HEAD
```

Reset a file in the project to its state in a previous commit:

```
git checkout HEAD <FILE>
```

Reset a file in the project to its state in a previous commit:

```
git checkout <SHA1> <FILE>
```

Revert a commit's content to cancel its changes:

```
git revert <SHA1>
```

Submodules

Add a submodule to the project:

```
git submodule add <URL>
```

List all submodules of the project:

```
git submodule status
```

Initialize all submodules recursively:

```
git submodule update --init --recursive
```

Update all submodules recursively:

```
git fetch --recurse-submodules
git submodule sync --quiet --recursive
git submodule update --init --recursive
```

guidelines

Author: Adrian DC

Description: Guidelines applicable to all RadianDevCore projects and applicable projects

Date: 28/02/2025

Version: `1.0.0-17-g70ae3d4`

▸ Commits

Guidelines shared for all RadianDevCore projects and for applicable projects.

Table of contents

Commits commands

- `git commit` **:** Create a commit with the staged changes
- `git commit -s` **:** Create a signed commit with the staged changes

Conventional Commits - Customized specification

The commit contains the following structural elements:

Commit type

- `fix` **: Bug** or **issues fixes**
- `feat` **:** New or extended **feature**
- `chore` **:** Changes to **non-production** code
- `docs` **: Documentation** only changes
- `style` **: Cosmetic** changes only (white-space, formatting, etc)
- `refactor` **:** Code changes that **neither** fix a **bug** nor add a **feature**
- `perf` **:** Code change that **improve performance**
- `security` **:** Changes resolve **security** related **issues**
- `test` **:** Adding missing or correcting existing **tests**
- `build` **:** Changes to the **build** system or dependencies (**example:** `makefile, pip, docker`)
- `ci` **:** Changes to **CI** configuration files and scripts (**example:** `gitlab-ci`)
- `improvement` **:** Changes that **improve** sources without any impact
- `wip` **: Temporary** commits meant for **local** developments and rebases

Commit scope

A scope has to be provided to a commit's type, to provide additional contextual information, such as the **changed file**, the containing folder or a globally **modified feature**, and is contained **within parenthesis**, e.g. `feat(parser): add ability to parse arrays`.

Commit breaking changes

`BREAKING CHANGE:` A commit with `!` before the `:` or that has the text `BREAKING CHANGE:` at the beginning of its optional body or footer section introduces a breaking API change (correlating with MAJOR in **semantic versioning**).

A `BREAKING CHANGE` can be part of commits of any type.

Commit syntax

Notice these types are not mandated by the **conventional commits** specification, and have no implicit effect in **semantic versioning** (unless they include a `BREAKING CHANGE`).

```
<type>(mandatory scope)[optional !]: <description>

[optional body]

[optional footer]
```

Commit example

```
feat(scope): implement feature ABC in sources

Issue: #123
Details: Details about the changes
---

Signed-off-by: Firstname LASTNAME <firstname.lastname@example.com>
```

guidelines

Author: Adrian DC

Description: Guidelines applicable to all RadianDevCore projects and applicable projects

Date: 28/02/2025

Version: `1.0.0-17-g70ae3d4`

▸ GitLab Issues

Guidelines shared for all RadianDevCore projects and for applicable projects.

GitLab Issues - Labels

The following `state` common labels are available:

- `state/to-do` : Issues to start or in the backlog *(`#009966`)*
- `state/in-progress` : Issues currently in progress and being processed *(`#428bca`)*
- `state/in-review` : Issues that have been resolved and need reviewing *(`#dc8010`)*

The following `priority` common labels are available:

- `priority/urgent` : Urgent issues *(`#dc143c`)*
- `priority/wished` : Wished or low priority issues *(`#009966`)*
- `priority/blocked` : Blocked issues *(`#707070`)*
- `priority/evolutions` : Evolution issues *(`#6699cc`)*
- `priority/rejected` : Rejected issues *(`#330066`)*

The following `group` custom labels are available:

- `group/common` : Group common label, for issues affecting multiple group elements *(remove if wished, `#f0f0f0`)*
- `group/project` : Project specific group label, for issues affecting the current projet *(remove if wished, `#f0f0f0`)*
- `group/...` : Customized group labels, for example `documentation` , `c++` , `yocto` , ... *(create per need, `#f0f0f0`)*

GitLab Issues - Issue boards

The following issues boards are available:

- **Development:**
- **List** `Open` : New issues to analyze and to schedule
- **List** `state/to-do` : Issues to start or in the backlog
- **List** `state/in-progress` : Issues currently in progress and being processed
- **List** `state/in-review` : Issues that have been resolved and need reviewing
- **List** `Closed` : Issues that have been resolved, tested or rejected

guidelines

Author: Adrian DC

Description: Guidelines applicable to all RadianDevCore projects and applicable projects

Date: 28/02/2025

Version: `1.0.0-17-g70ae3d4`

▸ .bashrc

Guidelines shared for all RadianDevCore projects and for applicable projects.

~/.bashrc

The following guidelines are provided for `~/.bashrc`:

- Limiting changes to `~/.bashrc` is recommended to ease updates / system migration
- Creating `~/.bash_user.rc` and including it in `~/.bashrc` is recommended

Bind ~/.bash_user.rc

```
# As user
{
  cat >>~/.bashrc <<EOF

# User bash additions
if [ -f ~/.bash_user.rc ]; then
    . ~/.bash_user.rc
fi
EOF
}
```

~/.bash_user.rc

The following guidelines are provided for `~/.bash_user.rc`:

Prepare ~/.bash_user.rc

```
# As user
{
  cat >>~/.bash_user.rc <<EOF
# ~/.bash_user.rc
#
# Description: Import user additions from ~/.bashrc with the following lines
#
# # User bash additions
# if [ -f ~/.bash_user.rc ]; then
#     . ~/.bash_user.rc
# fi
EOF
}
```

Load SSH keychain in terminals

> ⚠️ / This section is optional, use if needed

```
# As user
{
  # Load SSH keychain
  cat >>~/.bash_user.rc <<EOF

# SSH keychain
if [[ \$- == *i* ]] && [ -x /usr/bin/keychain ]; then
  eval \$(keychain --eval --nogui -q ~/.ssh/id_rsa)
fi
EOF
}
```

Disable DBus access over SSH

> ⚠️ / This section is optional, use if needed

```
# As user
{
  # Disable DBus over SSH
  cat >>~/.bash_user.rc <<EOF

# DBus over SSH
if [ ! -z "\${SSH_CLIENT}" ] || [ ! -z "\${SSH_TTY}" ]; then
  export DBUS_SESSION_BUS_ADDRESS=''
fi
EOF
}
```

Show Git status in prompt

> ⚠️ / This section is optional, use if needed

```
# As user
{
  # Prepare Git prompt status
  TMP_PS1="${PS1}"
  if echo "${PS1}" | grep -q '__git_ps1'; then
    true
  elif echo "${PS1}" | grep -q '\\w'; then
    TMP_PS1="${TMP_PS1//\w/\w\\[\\033[00m\\]\\[\$(__git_ps1 \" (%s)\") \\]}"
  fi

  # Configure Git prompt status
  cat >>~/.bash_user.rc <<EOF

# Configure Git prompt status
source /etc/bash_completion.d/git-prompt
export GIT_PS1_SHOWCOLORHINTS=1
export GIT_PS1_SHOWCONFLICTSTATE=1
export GIT_PS1_SHOWDIRTYSTATE=1
export GIT_PS1_SHOWUNTRACKEDFILES=1
export GIT_PS1_SHOWUPSTREAM=1
export PS1='${TMP_PS1}'
EOF
}
```

Use nano as default editor

> ⚠️ / This section is optional, use if needed

```
# As user
{
  # Configure default editor
  cat >>~/.bash_user.rc <<EOF

# Default editor
export EDITOR='nano'
EOF
}
```

guidelines