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