Working with Git CLI

Set-up Git locally with an existing remote repository

```
git init --initial-branch=main
git remote add origin *insert your git repository's ssh link here*
git pull origin main
```

Set-up Git remotely with an existing local repository / local files

Step 1:

Follow the instructions on the platform for the remote repository (Github/Gitlab) on making a new repository.

Step 2 (if git repo is not initialized):

```
git init --initial-branch=main
```

Step 3:

```
git remote add origin *insert your git repository's ssh link here*
```

Step 4:

Follow the regular workflow to push your current files to the remote repository

Simple workflow

Step 1:

To add changes in the working directory to the staging area:

```
git add <filename>
```

or in case you want to stage all files in the working directory:

```
git add *
```

Step 2:

Checking whether the staged files are the ones desired to be committed:

```
git status
```

Step 3:

Commit the staged files:

```
git commit -m "your commit message"
```

Step 4:

Push the commited changes to the remote repository if set-up:

```
git push origin main
```

Note: main can be any branch

Git-branches

Making a new git branch:

```
git checkout -b 'branchname'
```

Git checkout switches you to another branch, by using -b you will create a new branch. You can now work inside this specific git branch.

Commits can be pushed to the remote repository with:

```
git push origin 'branchname'
```

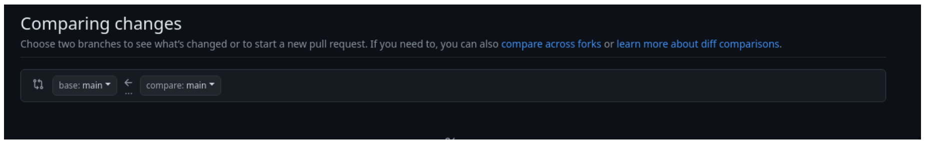
Pull requests

If you are working with a team that uses pull requests to maintain the main branch:

1. If you think the features on your branch are ready to be merged into the main branch.
2. Request a pull request via the platform your team is using (e.g. Github or Gitlab).
3. The ones reviewing the pull requests will check whether the request does not contain mistakes and is suitable to be merged into the main branch.
4. The pull request is either accepted or rejected with suggestions that should be added.

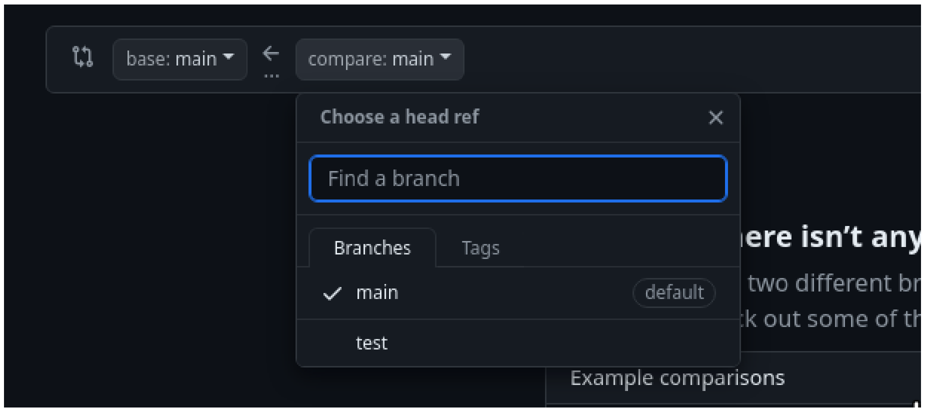
Pull requests on Github

Each repository on Github has a 'Pull requests' tab, with a button to create new pull requests.



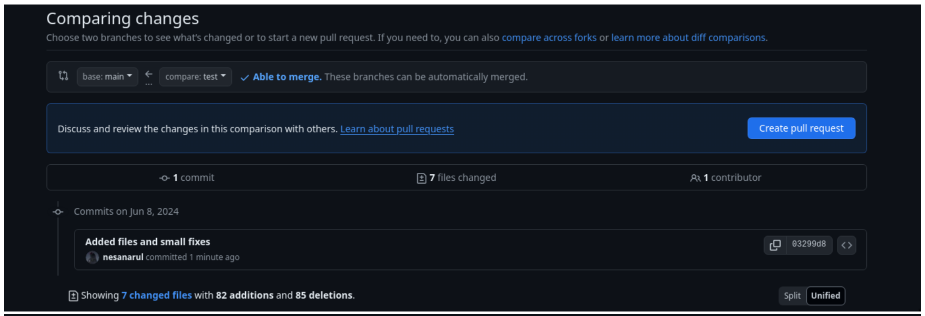
Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.



Step 1:

Pick the branch you were working on and want to create a pull request for as shown in the pictures above.



Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.

base: main ← compare: test ✓ **Able to merge.** These branches can be automatically merged.

Discuss and review the changes in this comparison with others. Learn about pull requests

Create pull request

1 commit **7** files changed **1** contributor

Commits on Jun 8, 2024

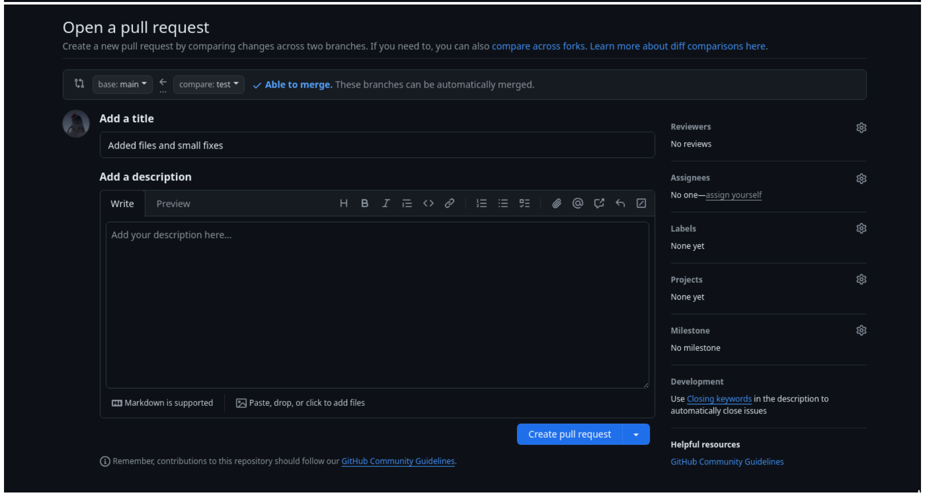
Added files and small fixes

nesanarul committed 1 minute ago

03299d8

Showing **7 changed files** with **82 additions** and **85 deletions**.

Split **Unified**



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base: main ← compare: test ✓ **Able to merge.** These branches can be automatically merged.

Add a title

Added files and small fixes

Add a description

Write Preview

Add your description here...

Markdown is supported Paste, drop, or click to add files

Create pull request

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Reviewers
No reviews

Assignees
No one—assign yourself

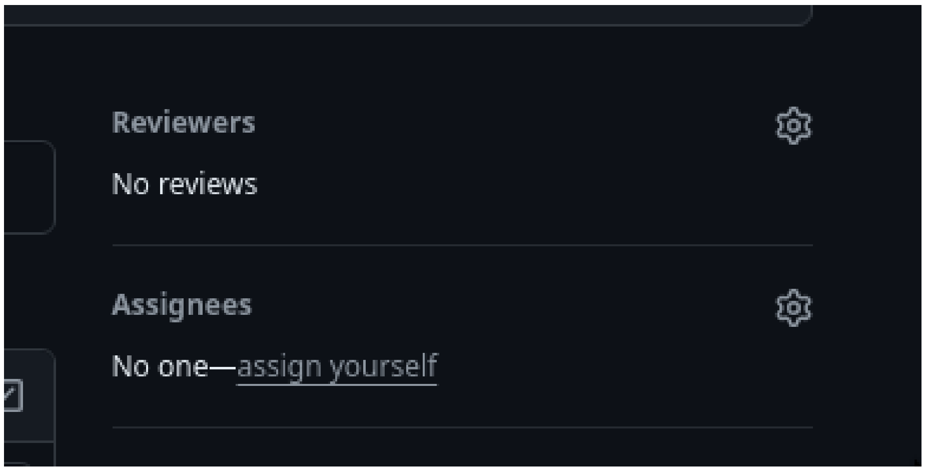
Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Use Closing keywords in the description to automatically close issues

Helpful resources
GitHub Community Guidelines


Reviewers

No reviews

Assignees

No one—assign yourself

Step 2:

Create the pull request, add a description if you want to add some extra details to the pull request. Afterwards, make sure you assign reviewers to review your pull request and submit the pull request when you have done this.

General workflow

If you are working with several people on the same branch, updating your local code with the newest content can be done using:

```
git pull
```

Note that this is the most simple command that can be used to do this, after all git pull is a combination of git fetch and git rebase/merge.

When working on files it is common to add them to the staging area after making some changes, this has to be done before any commits can be made. This is due to commits being based of the files in the staging area. The easiest command for this is (replace * with or use a .gitignore for more precise functionality):

```
git add *
```

After implementing a certain (part of a) feature it is recommended to create a commit, which essentially snapshots the current state of the project. The usual command for this is:

```
git commit -m "your commit message"
```

It is usually recommended to shortly describe the committed changes in your commit changes, although do avoid being too verbose in your message. Commits can be pushed onto the remote repository using:

```
git push origin 'branchname'
```

Other frequently used commands are:

```
git status
git checkout 'branchname'
```

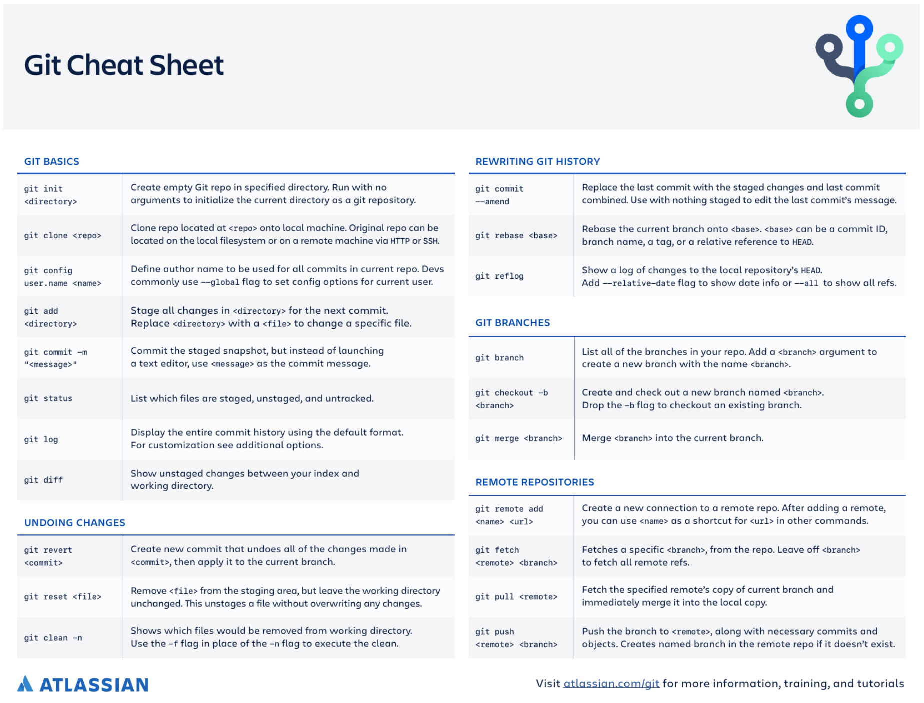
Git status shows the current status of the local repository, and git checkout 'branchname' moves your working directory to the mentioned branch.

Merge conflicts

Git can often resolve differences between branches and merge them automatically. However, if there are conflicting changes, manual intervention will be required. This is called a merge conflict. These will frequently occur when attempting to merge a pull request. To succesfully merge a pull request, one has to resolve all the conflicts that are present. This can be done through the CLI, or through the environment on platforms such as Github.

Extra

This document has been written by Arul, if you notice errors or unclear statements please contact me.



Git Cheat Sheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

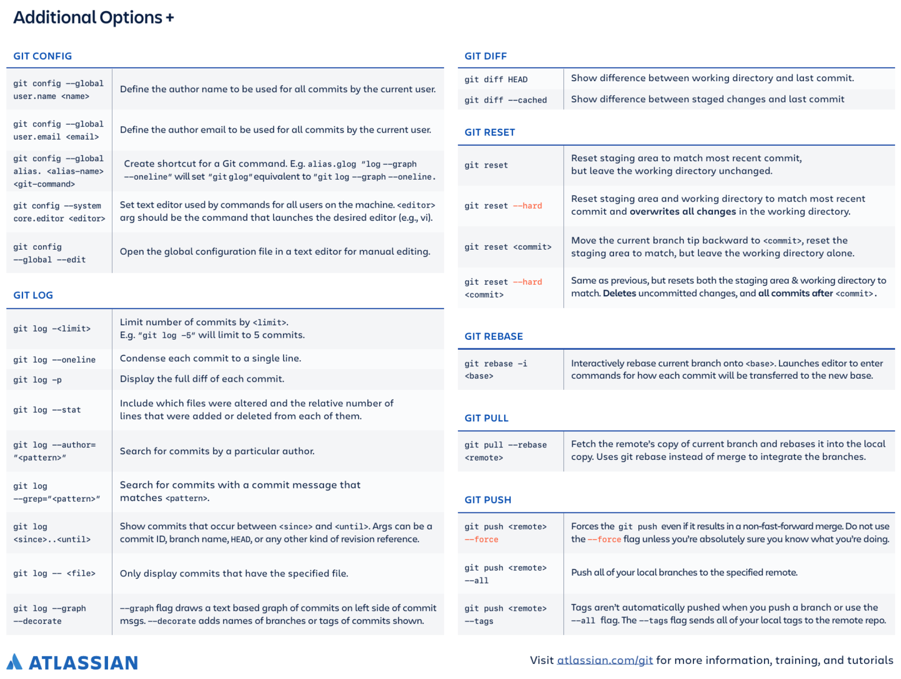
REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

ATLASSIAN

Visit atlassian.com/git for more information, training, and tutorials

Figure 1: Cheatsheet



Additional Options +

GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline`.
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT LOG

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `"git log -5"` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author= "<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, `HEAD`, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

GIT PUSH

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all` flag. The `--tags` flag sends all of your local tags to the remote repo.

ATLASSIAN

Visit atlassian.com/git for more information, training, and tutorials

Figure 2: Cheatsheet