

git

Packages:

```
sudo apt install git
```

To login via GitHub install gh:

```
sudo apt install gh
```

Configuring gh and git

Login to GitHub via (just follow the prompts)

```
gh auth login
```

Config username and email: → add `--global` option to just configure the login for one specific repository/directory

```
git config user.email "<email-address>"
```

```
git config user.name "<github-username>"
```

Replace `email-address` with your GitHub email and `github-username` with your username.

Initialize directory for git

Change to the highest directory of your project, then initialize repository:

```
git init
```

Add the remote:

```
git remote add origin '<url>'
```

In case of GitHub `<url>` for example can be [https://github.com/<username>/<repository>/](https://github.com/<username>/<repository>/.).

Pushing changes to GitHub

Add files to commit:

```
git add /path/to/file(s)
```

View status of changes:

git status

Before starting to edit, sync external changes:

```
git fetch <remote-repo>
```

```
git pull origin <branch>
```

Create commit:

```
git commit -vs
```

Or use the more simpler command:

```
git commit -m <message>
```

Push changes:

```
git push origin <branch>
```

Replace branch with the concerning branch, e.g. main.

Reset to previous commit (deletes all changes, use stash before to save changes and only delete the commit itself):

```
git reset --hard <commit>
```

To then push changes/reset:

```
git push --force
```

git merge and git rebase

Merge:

```
git merge
```

Rebase:

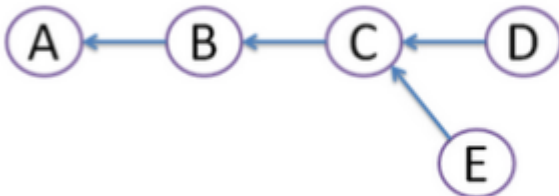
```
git rebase
```

Differences:

Suppose originally there were 3 commits, **A**, **B**, **C**:

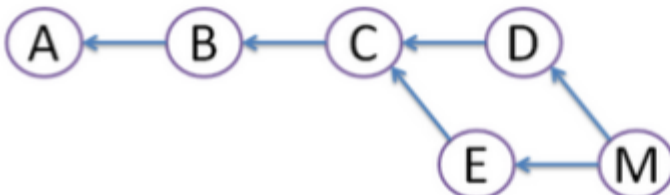


Then developer Dan created commit **D**, and developer Ed created commit **E**:



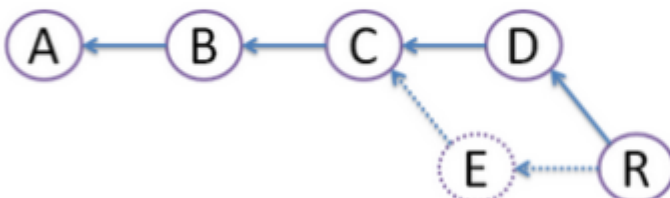
Obviously, this conflict should be resolved somehow. For this, there are 2 ways:

MERGE:



Both commits **D** and **E** are still here, but we create merge commit **M** that inherits changes from both **D** and **E**. However, this creates *diamond* shape, which many people find very confusing.

REBASE:



We create commit **R**, which actual file content is identical to that of merge commit **M** above. But, we get rid of commit **E**, like it never existed (denoted by dots - vanishing line). Because of this obliteration, **E** should be local to developer Ed and should have never been pushed to any other repository. Advantage of rebase is that *diamond* shape is avoided, and history stays nice straight line - most developers love that!

Sourced from: stackoverflow.com - [git merge](#) & [git rebase](#)

From:

<http://fixes.brecht-schule.hamburg/> - Fixes | Public BIT Wiki

Permanent link:

<http://fixes.brecht-schule.hamburg/linux/command-collection/git?rev=1780382070>

Last update: **2026/06/02 08:34**

