NodeJS

LABA

WHAT IS GIT

VCS: Version Control System

Git is distributed

Records changes made over time in a special database

Allows easily transitioning between code versions

Scalable to large codebases

HISTORY OF GIT

The first version of Git was created by Linus Torvalds (Linux creator), in just 10 days.

It was made with the workflow of sending patches through e-mail in mind.

COMMON GIT COMMANDS

```
git status
git add
git commit
git push
git pull
git log
```

Solvd
NodeJS LABA

LESS COMMON COMMANDS

```
git apply
git commit-tree
git hash-object
```

CONFIGURATION

Git has a configuration both at global level, and the repository (project) level.

```
git config –get user.name
git config –get user.email
```

If they aren’t set, you can set them as follows

```
git config –set user.name "your name"
git config –set user.email "your@email.com"
```

Solvd NodeJS LABA

REPOSITORIES

The first step of any project is to setup a git repository. A “repository” (or “repo”) represents a single project. You’ll usually have one repository for each project you work on.

A repo is essentially just a directory. The only difference is that it also contains a hidden `.git` directory, where git stores all its tracking and versioning information about the project.

REPOSITORIES

Creating a repository is done by using the `git init` command on a directory

```
mkdir my_project
cd my_project
git init
```

STATUS

Files can be in several states in a repository. Some importante ones are:

- `untracked`
- `staged`
- `committed`

The git status command shows you the current state of your repository. It will tell you untracked, staged and committed files.

STATUS

The git status command shows you the current state of your repository. It will tell you untracked, staged and committed files.

`untracked`

`staged`

`committed`

STAGING

The `README.md` file was created, but it is still `untracked`. We need to stage it (add to the “index”) with the `git add` command.

Without staging, every file would need to be included on ever commit, but that’s not what we want to achieve.

```
git add README.md
```

COMMIT

A commit is a snapshot of the repository at some given point in time. It is a way to save the state of the repository and is how git keeps track of changes to the project over time.

To commit all of your staged files, you can run:

```
git commit -m "commit message goes here"
```

ALMOST ALL OF IT...

This is half of the git you'll be using on your day to day…

But there is still a lot more to git than what we covered up to now.

GIT LOG

A repository is a list (often very long) of commits, where each one represents at the full state of the repository at a point in time.

`git log` command shows the history of the commits in a repository. And allows you to see who made a commit, when was the commit made, and what was changed in the commit.

COMMIT HASH

A commit hash is a unique identifier for a commit. You'll notice that even though we have the same contents on our repository, my commit hash is different than yours. That is because git commit hashes are derived from the content, but also from other things. For example:

- Commit message
- The author's name and email
- The Date and time

GOING DEEPER

We’ve been using git in a very simple manner, and despite that being all you’re going to do most of the times. Git is much more than that.

ITS JUST FILES

All the data in the git repository is stored in the hidden .git directory. That includes all the commits, branches, tags and every other object

Git is made of objects, and a commit is just one of the types of objects

WHATS INSIDE OF IT?

Let’s take a look at what’s inside this commit object file...

Solvd
NodeJS LABA

WHATS INSIDE OF IT?

Well… That was just a mess…

But luckily, git already had us covered. We can run the command `cat-file` to view this object contents

```
git cat-file -p <hash>
```

TREES AND BLOBS

Lets go a little more deeper into git concepts. Some terms to know are:

- Tree: git’s way of storing a directory
- Blob: git’s way of storing a file

TREES AND BLOBS

Here’s what I got when I inspected my last commit:

```
git cat-file -p a20da52
tree fa5e6af79e30fc26ab4acbd96388fde22b4c2f36
author Willians Faria <dev.willians.faria@gmail.com> 1748209736 -0300
committer Willians Faria <dev.willians.faria@gmail.com> 1748209736 -0300

creating readme
```

Notice we can see the tree, author and commit message

SECOND COMMIT

Lets create a new file `todo.md`, add the following content to it and create our second commit.

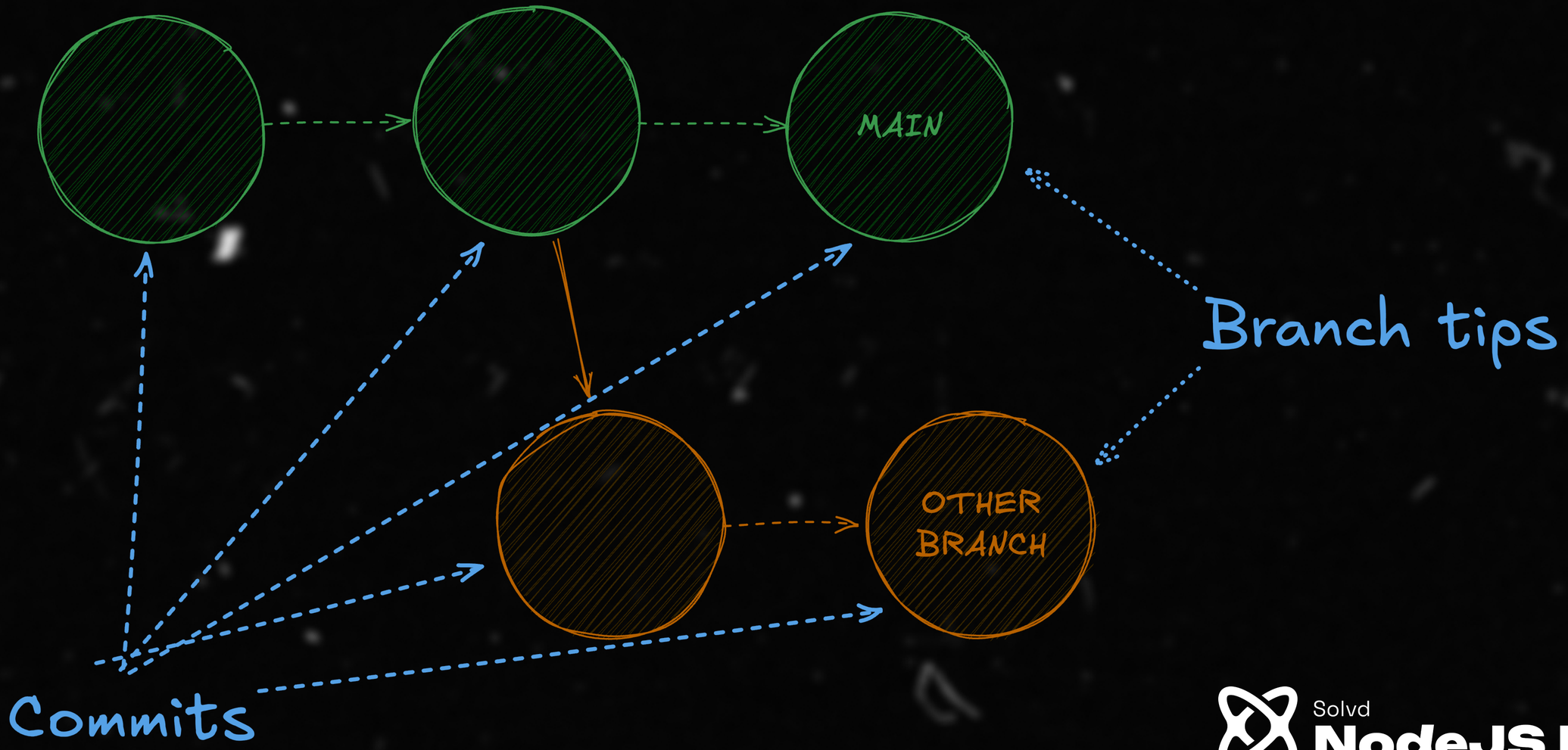
```
# TODO

*  Create a second commit
*  Inspect how it changes our tree!
```

GIT STORING DATA

Git stores an entire snapshot of files per-commit level. And not only the changes that commit introduced.

But while this is true, git also uses some optimization techniques. Git compresses and packs files. Git deduplicates files that are the same across commits. And this is one very important detail.

BRANCHING



Solvd
NodeJS LABA

BRANCHING

You can check in which branch you’re currently in using the command:

```
git branch
```

MAKING A NEW BRANCH

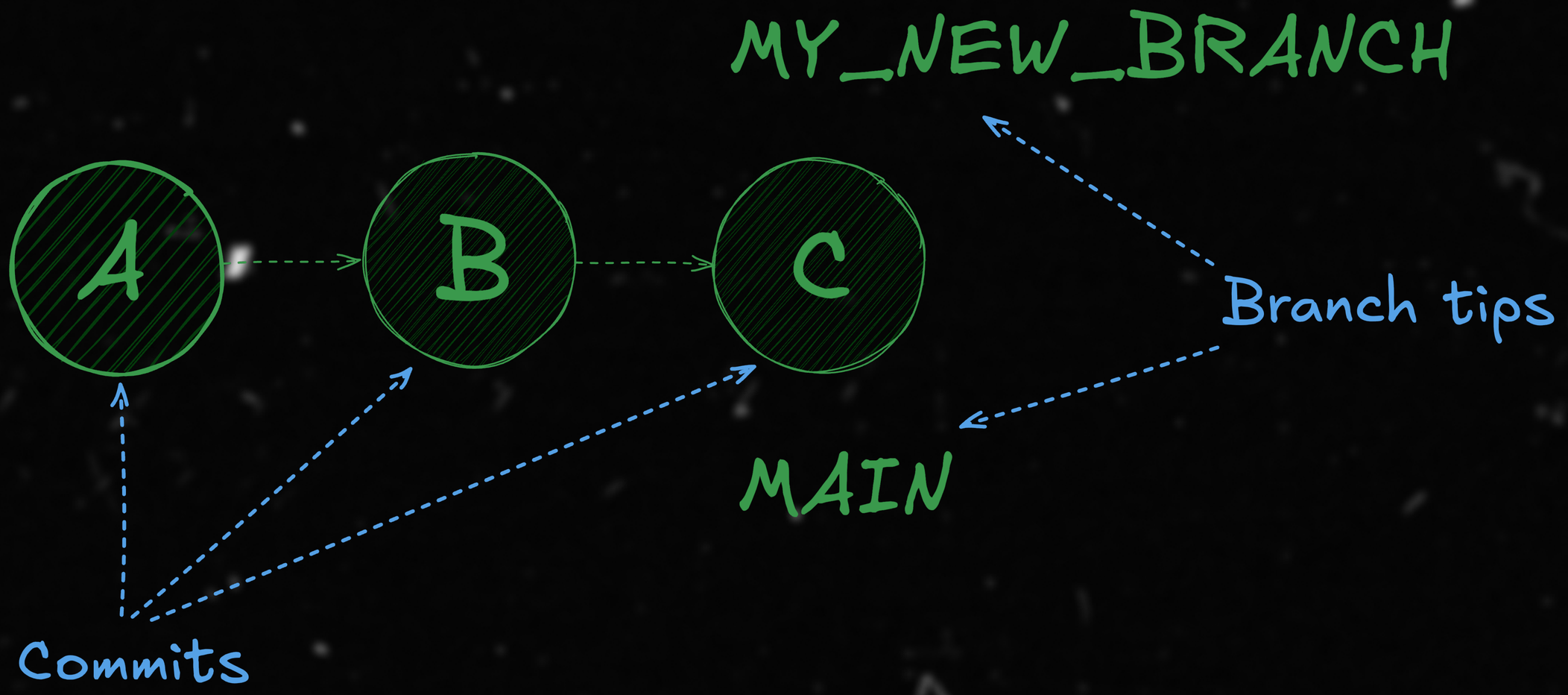
You can make a new branch using the command:

```
git branch my_new_branch
```

But usually, when you make a new branch you also want to switch to it. You can do so by using:

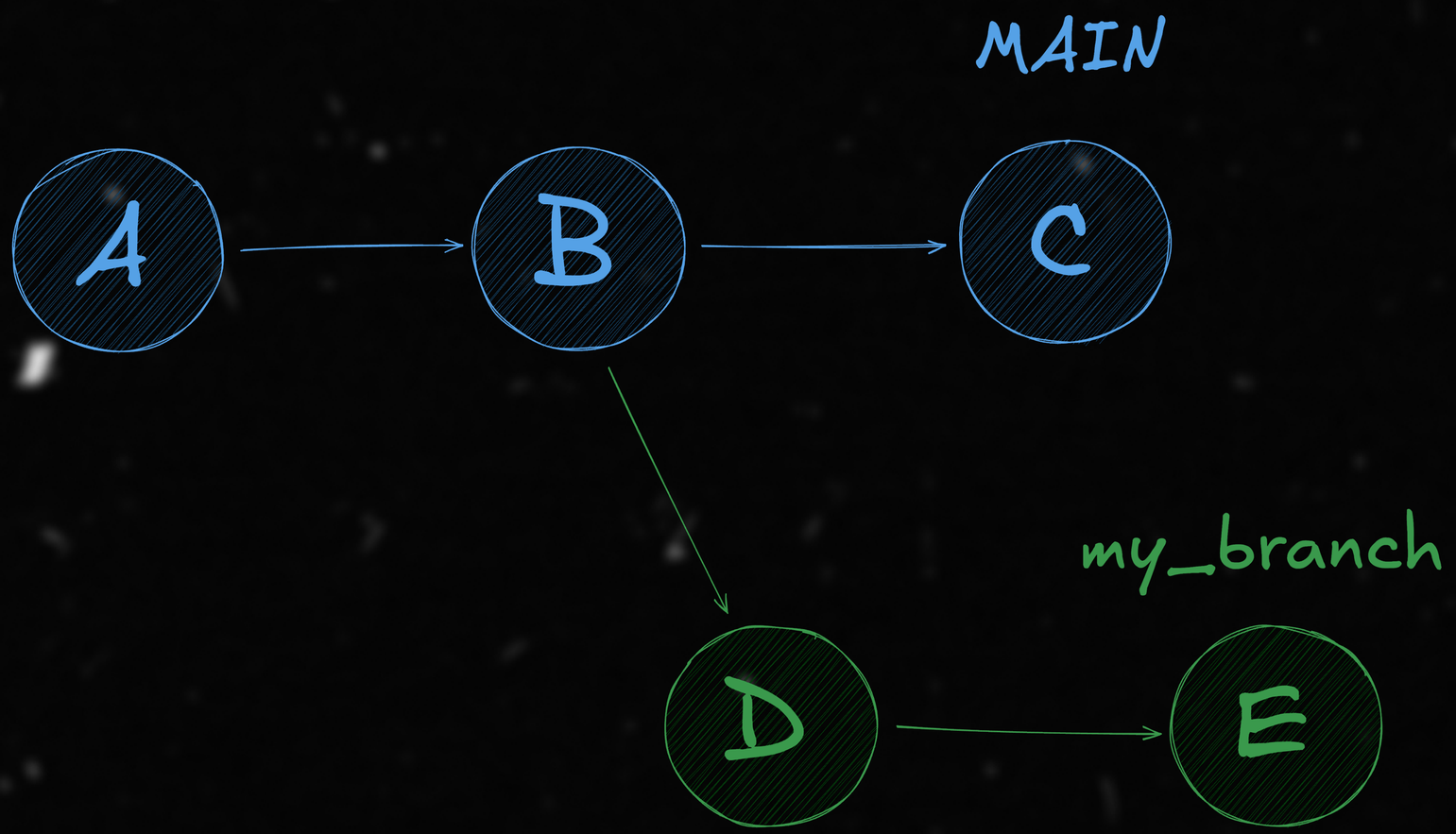
```
git switch -c my_new_branch
```

MAKING A NEW BRANCH



Solvd
NodeJS LABA

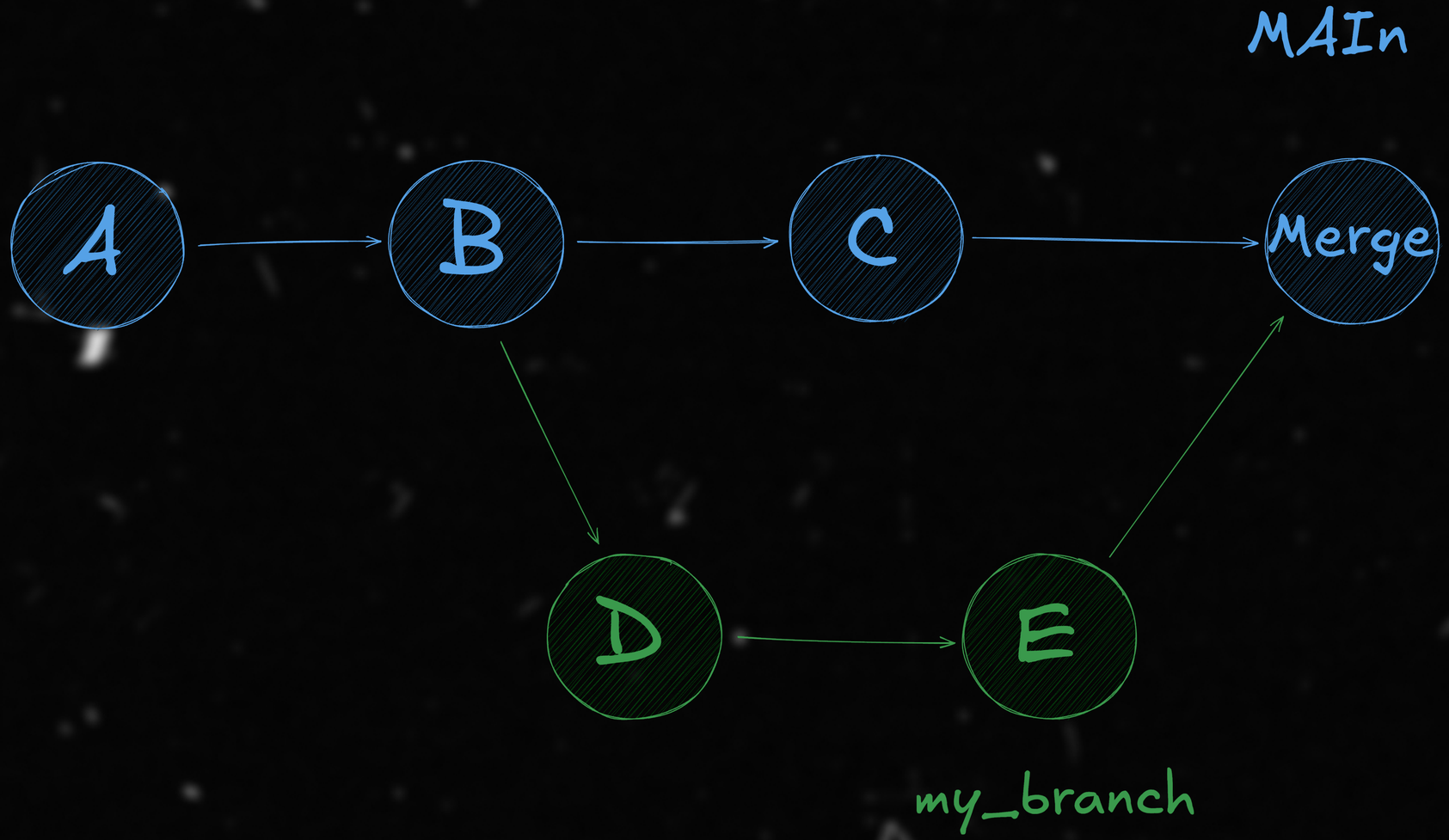
MERGING




```
git merge <branch>
```

Solvd NodeJS LABA

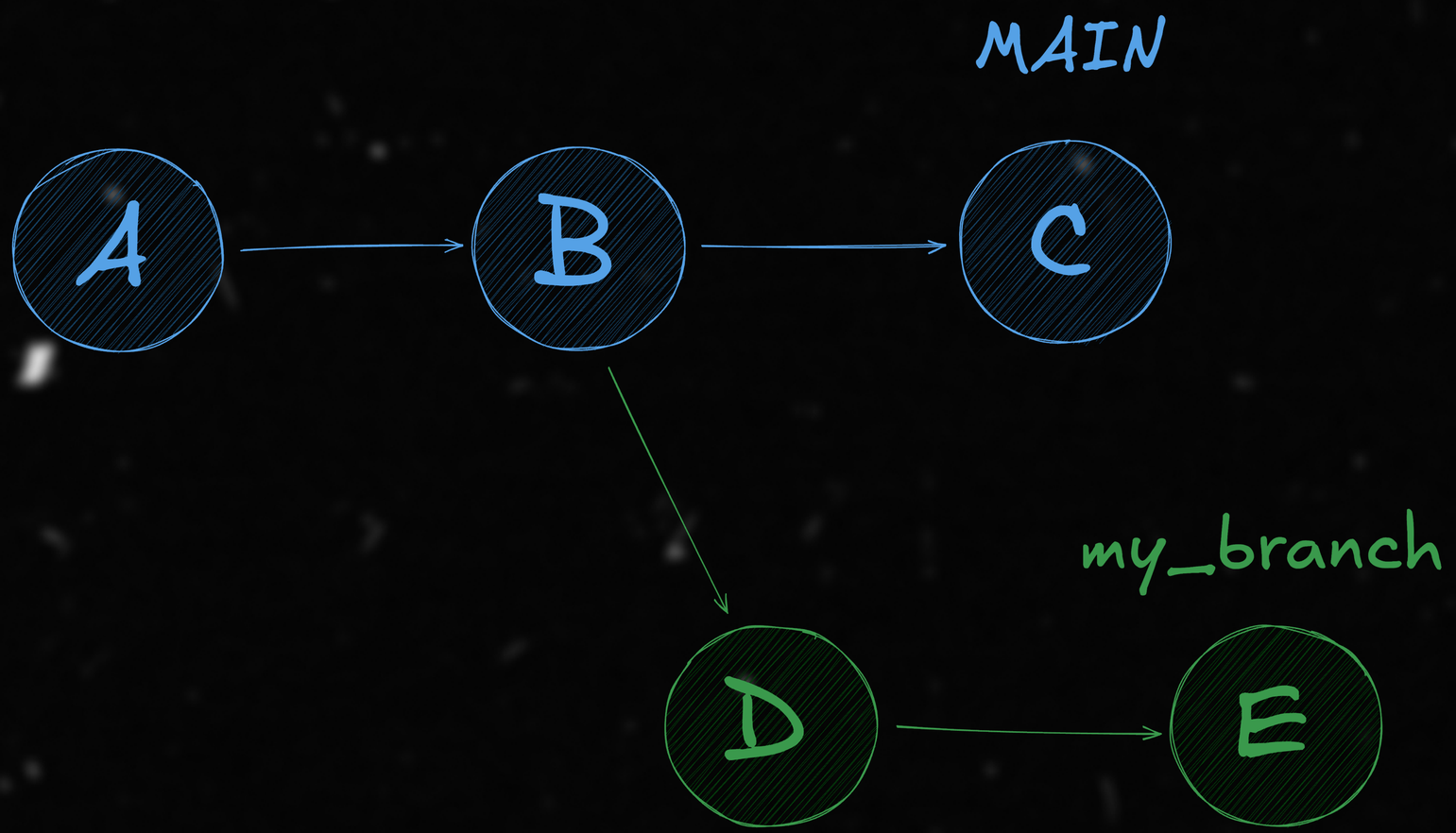
MERGING




```
git merge <branch>
```

Solvd NodeJS LABA

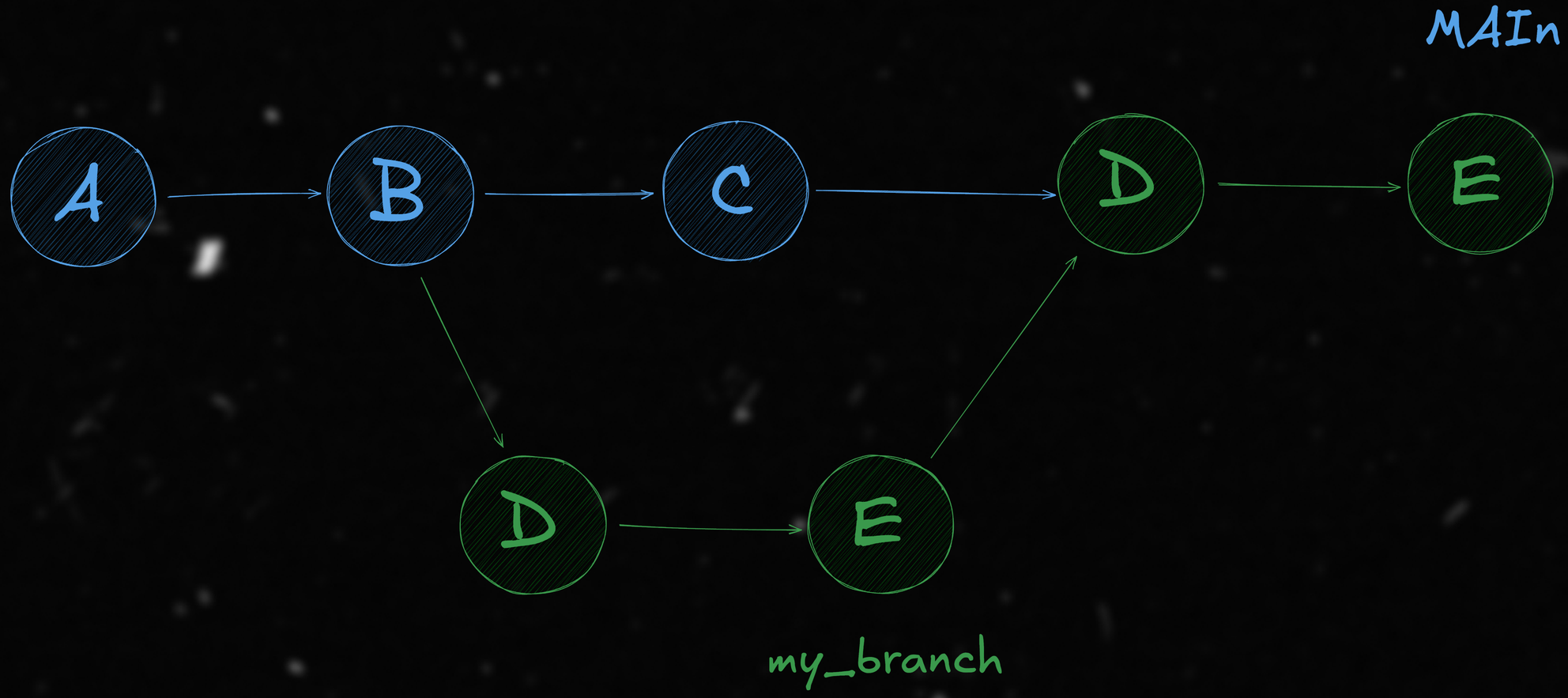
REBASING




```
git rebase <branch>
```

Solvd
NodeJS LABA

REBASING




```
git rebase <branch>
```

Solvd
NodeJS LABA

REBASING

Rebase essentially makes the branch you're currently working on point to a another commit on the merge base.

This allows for a fast-forward merge in the future, which removes any "merge commits".

This has some nuances, but it is very powerful and allows for the use of more advanced git features.

Solvd NodeJS LABA

CONFLICTS

Git conflicts happen when multiple branches change the same lines of code, making git unable to decide which changes should be kept.

This requires manual actual from the programmer, and is a process you'll do very often.