# Understanding Merge Types

ADVANCED GIT

**Amanda Crawford-Adamo**
Software and Data Engineer

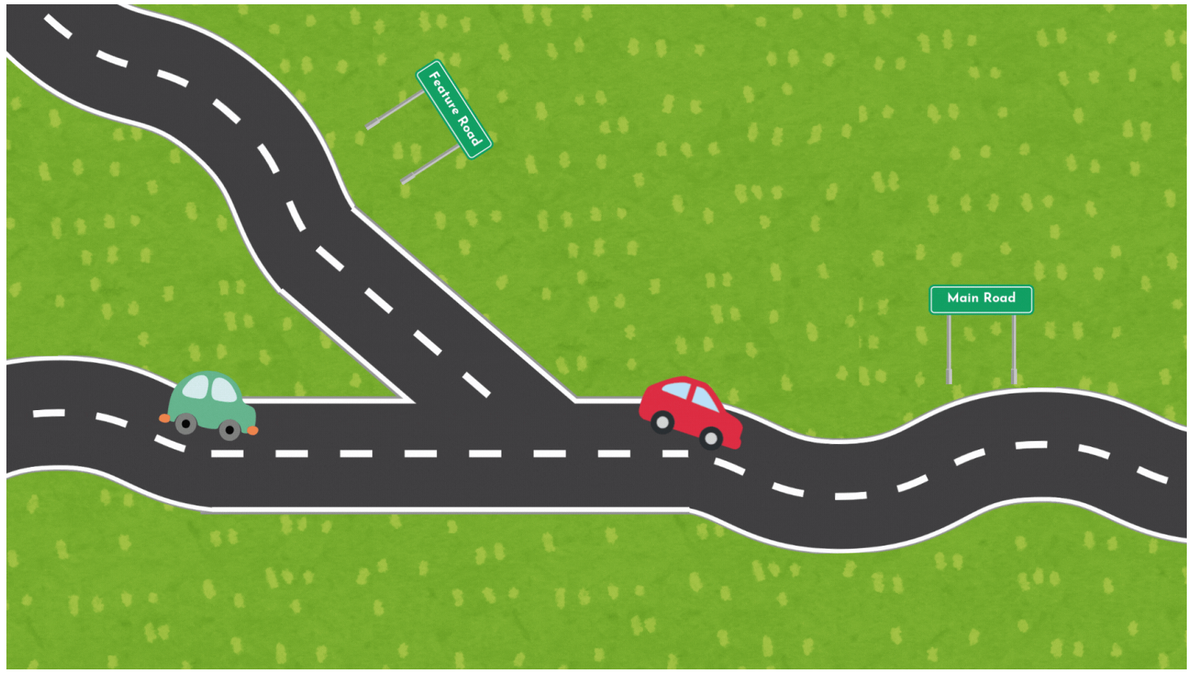
# I'm Amanda

# What is git merge?

**Git Merge Command:**

```
git merge
```

- Combines changes from one branch into another
- Finds the common base between two branches
- Use different merge strategies



# Fast-forward merge

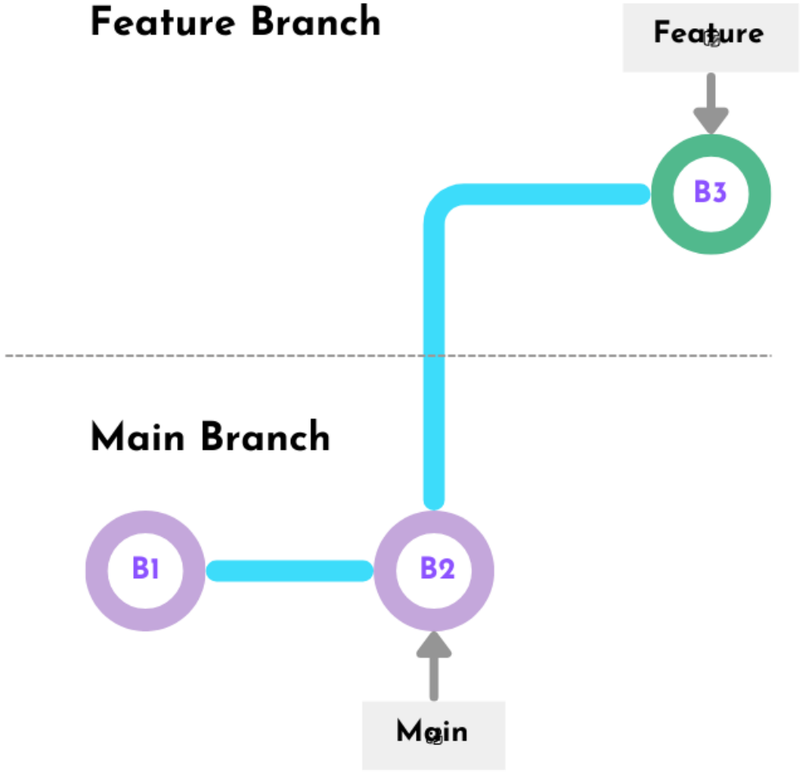
**What is fast forward merge?**

- Keep a simple, straight history
- Ideal for short-lived branches with simple changes

**When not to use**

- Need to preserve branch history
- Complex feature development in long-lived branches
- Merge conflict between branches - fast forward fails immediately!

# Fast-forward merge (before)




Data Team

```
$ git log --oneline
abc1234 (main) Add data ingestion module
def5678 Initial commit

$ git log --oneline feature-branch
ghi9101 (feature-branch) Implement data validation
abc1234 Add data ingestion module
def5678 Initial commit
```

# Fast-forward merge syntax

### Git Merge Fast Forward Default

```
git checkout main
git merge feature_branch
```

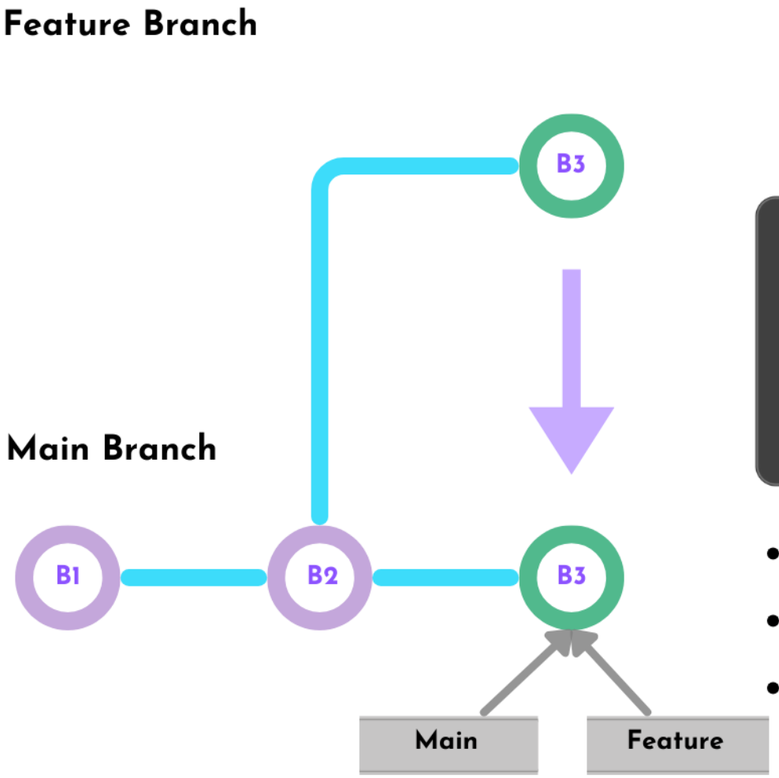
### Force Git Merge Fast Forward

```
git merge <branch> --ff-only
```

### Example

```
git checkout main
git merge feature_branch --ff-only
```

# Fast-forward merge (after)




Data Team

```
$ git log --oneline
ghi9101 (HEAD -> main, feature-branch) Implement data validation
abc1234 Add data ingestion module
def5678 Initial commit
```

- No additional commit is on main.
- Feature branch commits easily added to main.
- The main branch history remains linear.

# Recursive merge

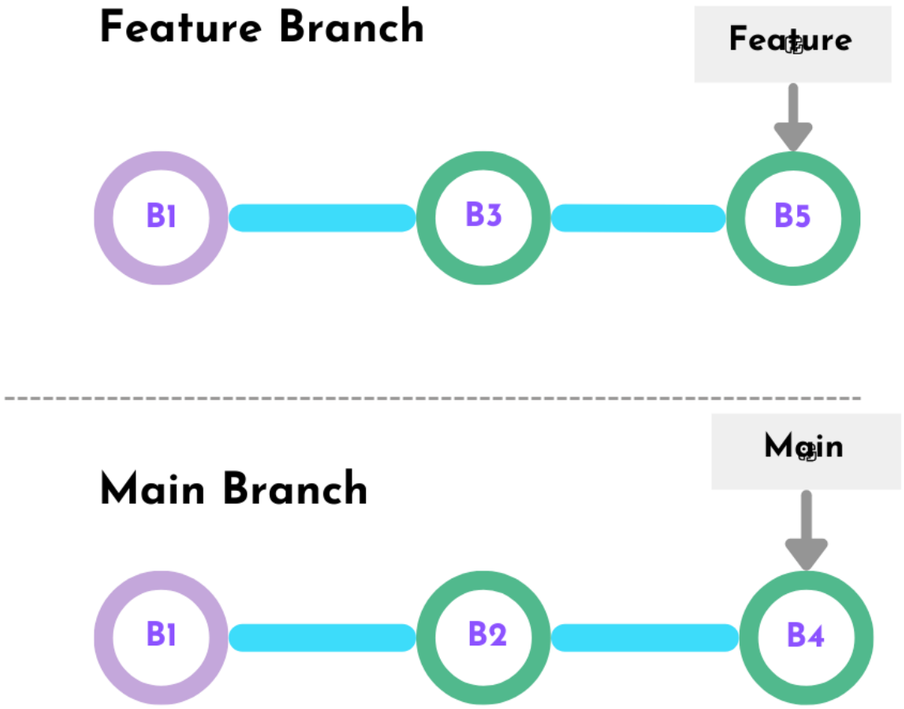
## What is a recursive merge?

- Creates a merge commit with two parents
- Preserve the entire project history
- Ideal for long-lived branches
- Maintain branching structure

## When not to use

- When you want to maintain a simple and linear history
- Quick and minor changes

# Recursive merge (before)




Data Team

```
$ git log --oneline main
abc1234 (main) Update data schema
def5678 Add data ingestion module
ghi9101 Initial commit

$ git log --oneline feature-branch
jkl2345 (feature-branch) Implement data transformation
mno6789 Add data validation
ghi9101 Initial commit
```

# Recursive merge syntax

### Recursive Merge Command:

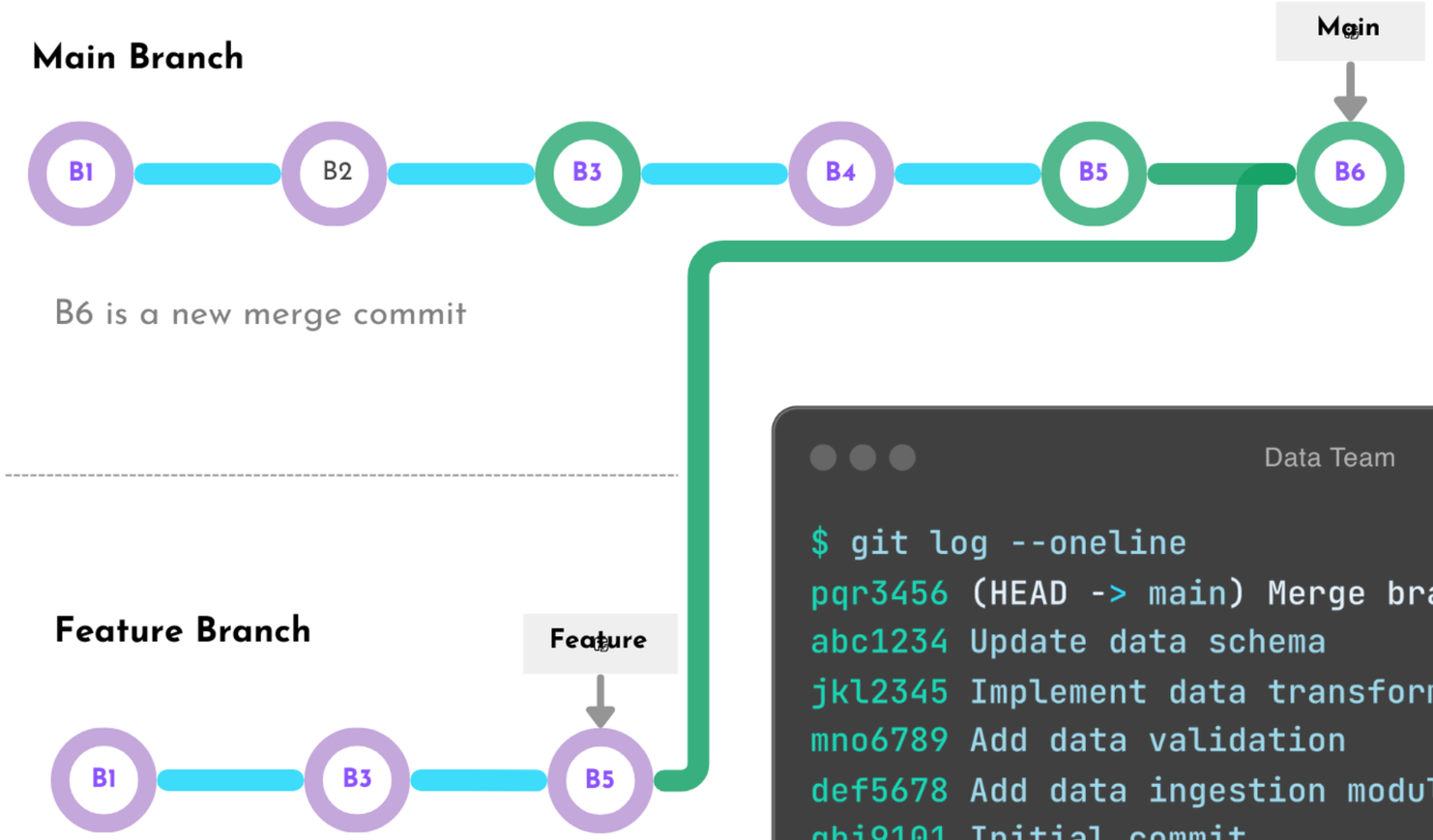
```
git merge --no-ff <branch>
```

### Example

```
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

$ git merge --no-ff feature_branch
Merge made by the 'recursive' strategy.
...
```

# Recursive merge (after)




Data Team

```
$ git log --oneline
pqr3456 (HEAD -> main) Merge branch 'feature-branch'
abc1234 Update data schema
jkl2345 Implement data transformation
mno6789 Add data validation
def5678 Add data ingestion module
ghi9101 Initial commit
```

# Summary

- Fast forward merges keeps history simple and linear
- Recursive merges preserves historical context
- Recursive merges are better for more complex development

### Fast Forward Merge Commands

```
git merge <branch_name> # default command
git merge --ff-only <branch_name> # force fast forward merge
```

### Recursive Merge Command

```
git merge --no-ff <branch_name>
```

# Let's practice!

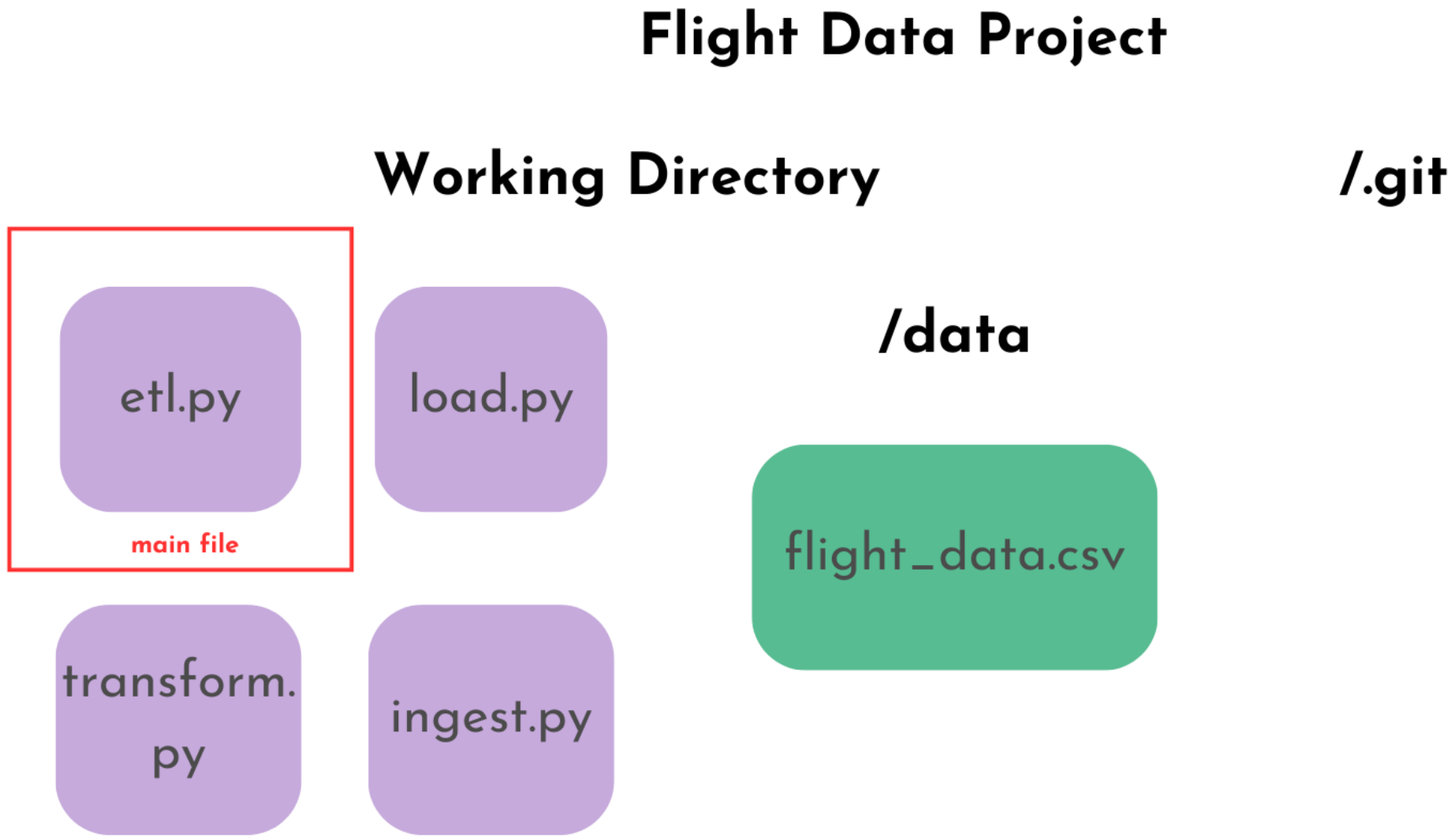
ADVANCED GIT

# Complex Merge Scenarios

ADVANCED GIT

**Amanda Crawford-Adamo**
Software and Data Engineer

# Flight travel data pipeline repo



# Git squash merging

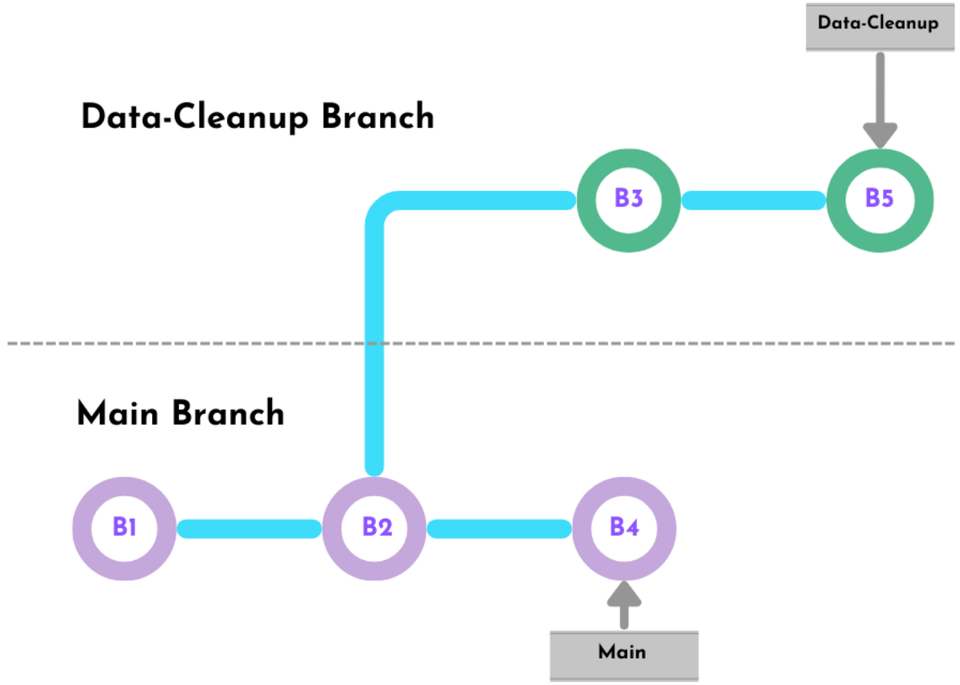
## Functionality

- Creates a single new commit on the target branch
- Combines all changes from the source branch
  - Adds a regular commit with one parent (unlike recursive strategy)
- Added to target branch (**not** feature branch)
  - Doesn't preserve the detailed commit history of the source branch

## Advantages

- Clean and linear history
- Simplifies code review for large features
- Easier to revert the entire feature

# Merge squash example




Data Team

```
$ git log --oneline main
abc1234 (main) Update data schema
def5678 Add data ingestion module
gh19101 Initial commit

$ git log --oneline data-cleanup
jkl2345 (data-cleanup) Optimize data cleaning
mno6789 Fix bug in data cleaning
pqr0123 Implement data cleaning
ghi9101 Initial commit
```

# Merge squash process

1. Checkout the main branch

```
$ git checkout main
```

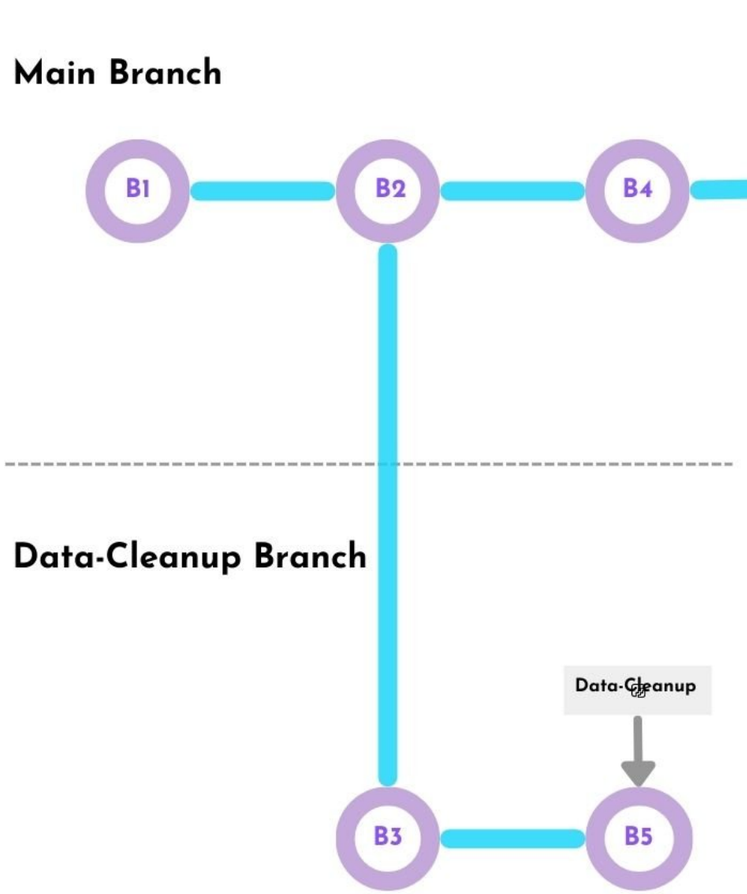
2. Create a squash commit of all data-cleanup changes

```
$ git merge --squash data-cleanup
```

3. Commit the squash commit to main branch history

```
$ git commit -m "Implement and optimize data cleanup"
```

# Merge squash result




Data Team

```
$ git log --oneline
stu3456 (HEAD -> main) Implement and optimize data cleaning
abc1234 Update data schema
def5678 Add data ingestion module
ghi9101 Initial commit
```

# Git octopus merge

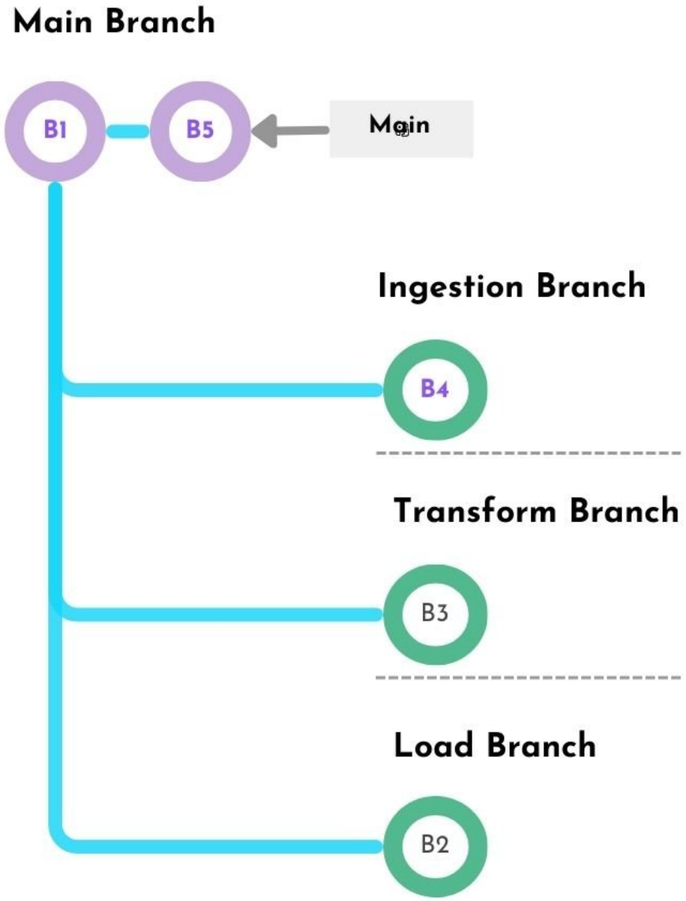
## Functionality

- Merges three or more branches at once
- Creates a single merge commit with multiple parents
- Best used when branches don't conflict with each other

## Advantages

- Useful for integrating multiple independent features simultaneously
- Used for synchronizing several release branches for different versions of a project

# Octopus merge example




Data Team

```
$ git log --oneline main
abc1234 (main) Update main pipeline
def5678 Initial commit

$ git log --oneline ingestion
ghi9101 (ingestion) Implement data ingestion
def5678 Initial commit

$ git log --oneline transformation
jkl2345 (transformation) Implement data transformation
def5678 Initial commit

$ git log --oneline load
mno6789 (load) Implement data loading
def5678 Initial commit
```

# Octopus merge commands

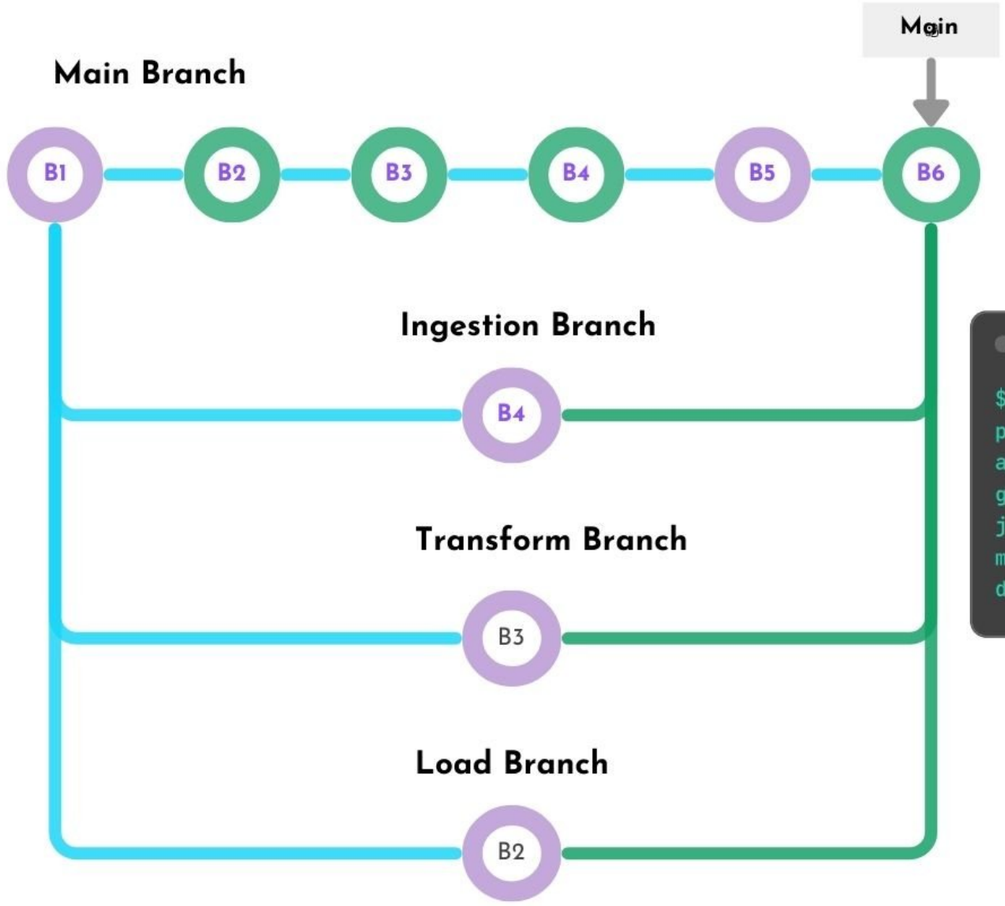
## Git Octopus Merge Command

```
git merge -s octopus
```

## Example

```
$ git merge -s octopus ingest transform load
Trying simple merge with ingest
Trying simple merge with transform
Trying simple merge with load
Merge made by the 'octopus' strategy.
...
```

# Octopus merge result




Data Team

```
$ git log --oneline
pqr0123 (HEAD -> main) Merge branches 'ingestion', 'transformation' and 'load'
abc1234 Update main pipeline
ghi9101 Implement data ingestion
jkl2345 Implement data transformation
mno6789 Implement data loading
def5678 Initial commit
```

# Summary

### Squash merge

- Simplifies history, combines multiple commits into one
- Use squash merges for a clean, simplified history

```
git merge --squash <source_branch>
```

### Octopus merge

- Preserves branch structure
- Merges multiple branches simultaneously
- Efficiently integrating multiple parallel developments

```
git merge -s octopus <branch 1> <branch 2> <branch 3>
```

# Let's practice!

ADVANCED GIT

# Advanced Branch Integration: Git Rebasing

ADVANCED GIT

**Amanda Crawford-Adamo**
Software and Data Engineer

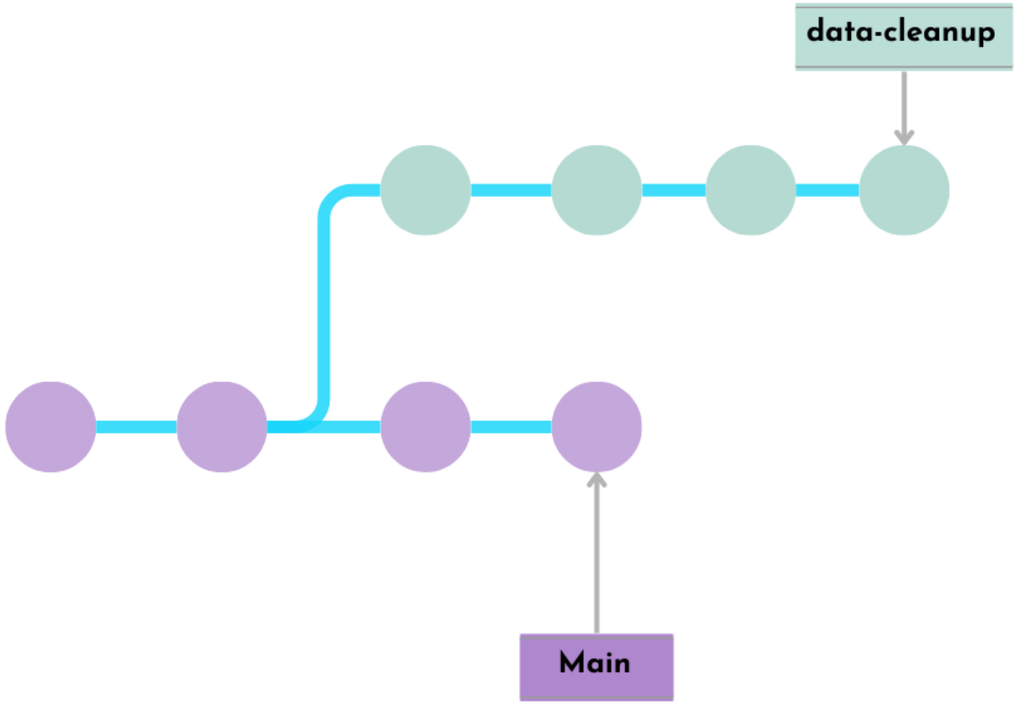
# Git rebase

- Method to integrate changes
- Different from merge
- Removes merge commits for a cleaner history
- Maintains a linear commit graph for clarity

**Git Rebase Command:**

```
git rebase <branch_name>
```

# Rebase example - before




Data Team

```
$ git log --oneline data-cleanup
abc1234 Optimize validation performance
def5678 Fix validation bug
ghi9101 Add data validation function
jkl2345 Initial commit
```

# Rebase process

1. Checkout your `data-cleanup` feature branch.

```
git checkout data-cleanup
```

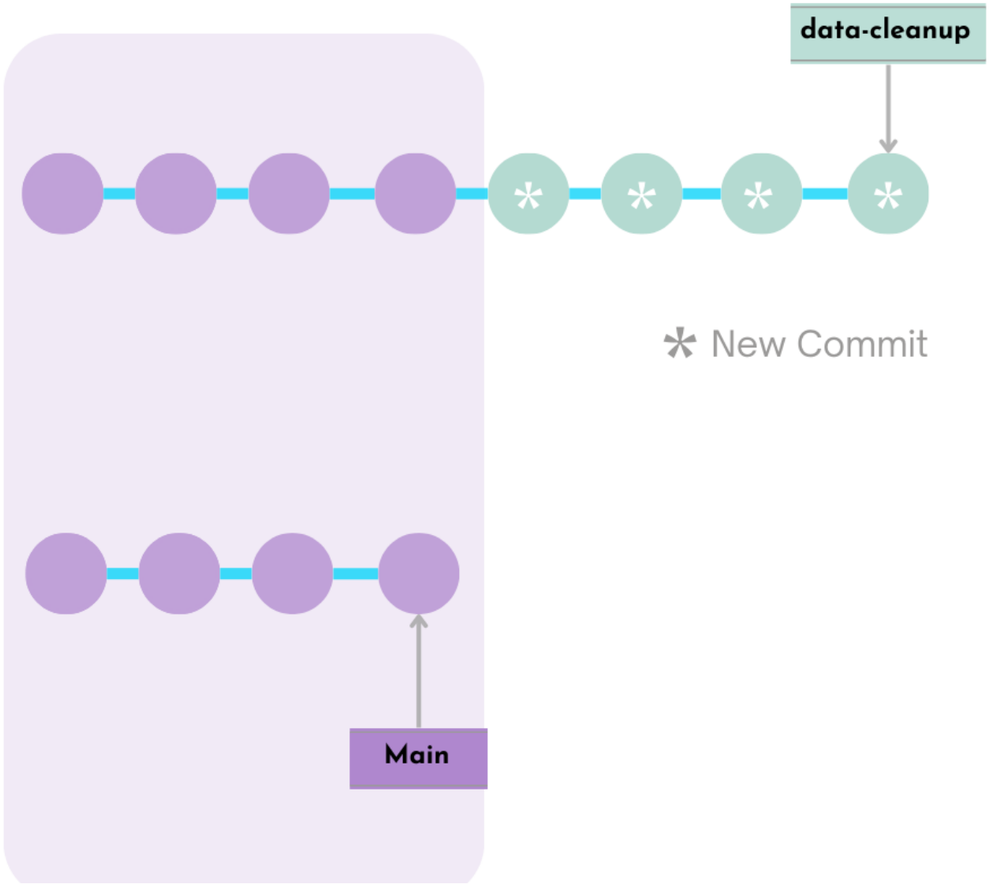
2. Rebase `main` branch onto `data-cleanup` branch.

```
git rebase main
```

   - `data-cleanup` commits are recreated after `main` 's latest commit
   - Rebased `data-cleanup` commits get new hashes

**Note:** If there are any conflicts, these would need to be resolved manually.

# Rebase example - after




Data Team

```
$ git log --oneline data-cleanup
mno6789 Optimize validation performance
pqr0123 Fix validation bug
stu3456 Add data validation function
vwx7890 Main Branch Commit
jkl2345 Initial commit
```

# Interactive rebase

```
git rebase -i <commit_hash>
```

**Functionality**

- Allows developer to make granular changes to multiple commits
- Opens an editor

**Warning!**

- Rebasing public branches can disrupt workflows
  - Do not use rebase on public branches, like `main`
- Establish team rules on when to use rebase

# Interactive rebase - before

Here is the **data-validation branch history** before we edit the commit history.

```
$ git log --oneline data-validation
abc1234 Optimize validation performance
def5678 Fix validation bug
ghi9101 Add data validation function
xyz1234 New Main Branch Commit
vwx7890 Main Branch Commit
jkl2345 Initial commit
```

# Interactive rebase editor

Here's how we edit the commit history using **pick** and **fixup** commands in the open editor.

```
$ git rebase -i HEAD~3
pick ghi9101 Add data validation function
fixup def5678 Fix validation bug
fixup abc1234 Optimize validation performance

# Rebase xyz1234..abc1234 onto HEAD~3(xyz1234) (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# f, fixup <commit> = like "squash", but discard this commit's log message
...
```

# Interactive rebase - after

### Before

```
$ git log --oneline data-validation
abc1234 Optimize validation performance
def5678 Fix validation bug
ghi9101 Add data validation function
xyz1234 New Main Branch Commit
vwx7890 Main Branch Commit
jkl2345 Initial commit
```

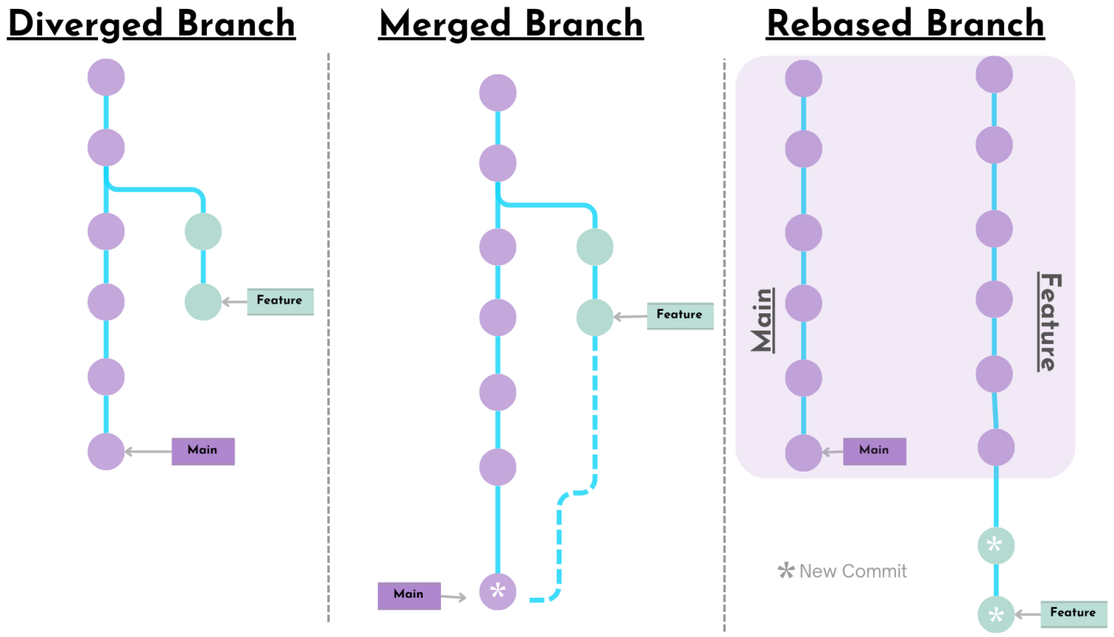
### After

```
$ git log --oneline data-validatoin
mno6789 Add data validation function
xyz1234 New Main Branch Commit
vwx7890 Main Branch Commit
jkl2345 Initial commit
```

The following commits are combined into one

- abc1234
- def5678
- ghi9101

# Merge versus rebase




## Merge

- Preserves full history and branch structure
- Maintains context of parallel development

## Rebase

- Replays commits on top of the target branch
- Creates a linear history
- Loses some contextual information

# When to use merge versus rebase

**Merge**

For integrating completed features, preserving development context

**Rebase**

For keeping feature branches updated with main, or for cleaning up before merging

**Remember**

- Rebase rewrites history - use it carefully

# Let's practice!

ADVANCED GIT