

# Configuring Git

## INTRODUCTION TO GIT

**George Boorman**

Curriculum Manager, DataCamp

# Why do we need to configure our settings?

- Git has customizable settings to speed up or improve how we work!



<sup>1</sup> Image credit: <https://unsplash.com/@schmaendels>

# Levels of settings

- `git config --list`
- Git has three levels of settings:
  1. `--local` : settings for one specific project
  2. `--global` : settings for all of our projects
  3. `--system` : settings for every users on this computer

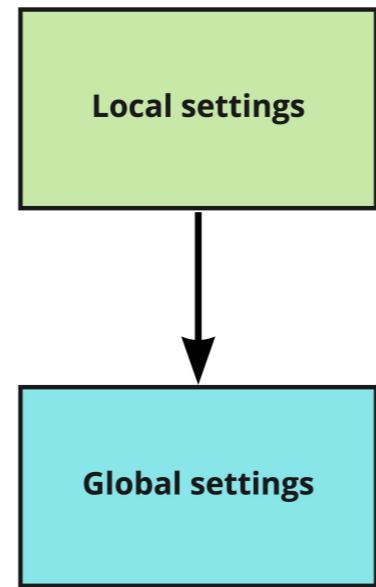
# Local settings

## Setting Levels

Local settings

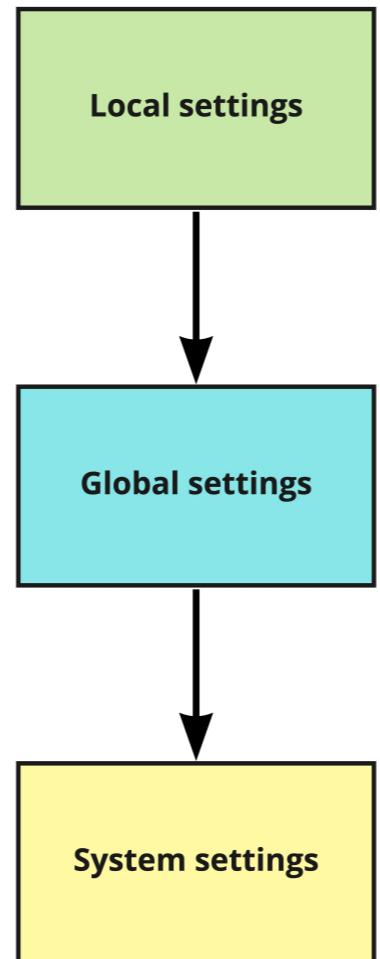
# Global settings

## Setting Levels



# System settings

## Setting Levels



# What can we configure?

```
git config --list
```

```
user.email=repl@datacamp.com
user.name=Rep Loop
core.editor=nano
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
```

- `user.email` and `user.name` are needed by some commands, so setting these saves time!
- `user.email` and `user.name` are **global** settings

# Changing our settings

```
git config --global setting value
```

- Change email address to **johnsmith@datacamp.com**:

```
git config --global user.email johnsmith@datacamp.com
```

- Change username to John Smith:

```
git config --global user.name 'John Smith'
```

- If we don't use `' '` and our `user.name` has a space:
  - Git would save `user.name` as `John`

# Using an alias

- Set up an alias through global settings
- Typically used to shorten a command
- To create an alias for committing files by executing `ci`:

```
git config --global alias.ci 'commit -m'
```

- Again, we use `''` so Git processes characters after the space
- We can now commit files by executing:

```
git ci
```

# Creating a custom alias

- We can create an alias for any command
- If we often unstage files:

```
git config --global alias.unstage 'reset HEAD'
```

- Be careful not to overwrite existing commands!

# Tracking aliases

.gitconfig file

```
git config --global --list
```

Output format: alias.aliasname=command

```
alias.ci=commit -m
```

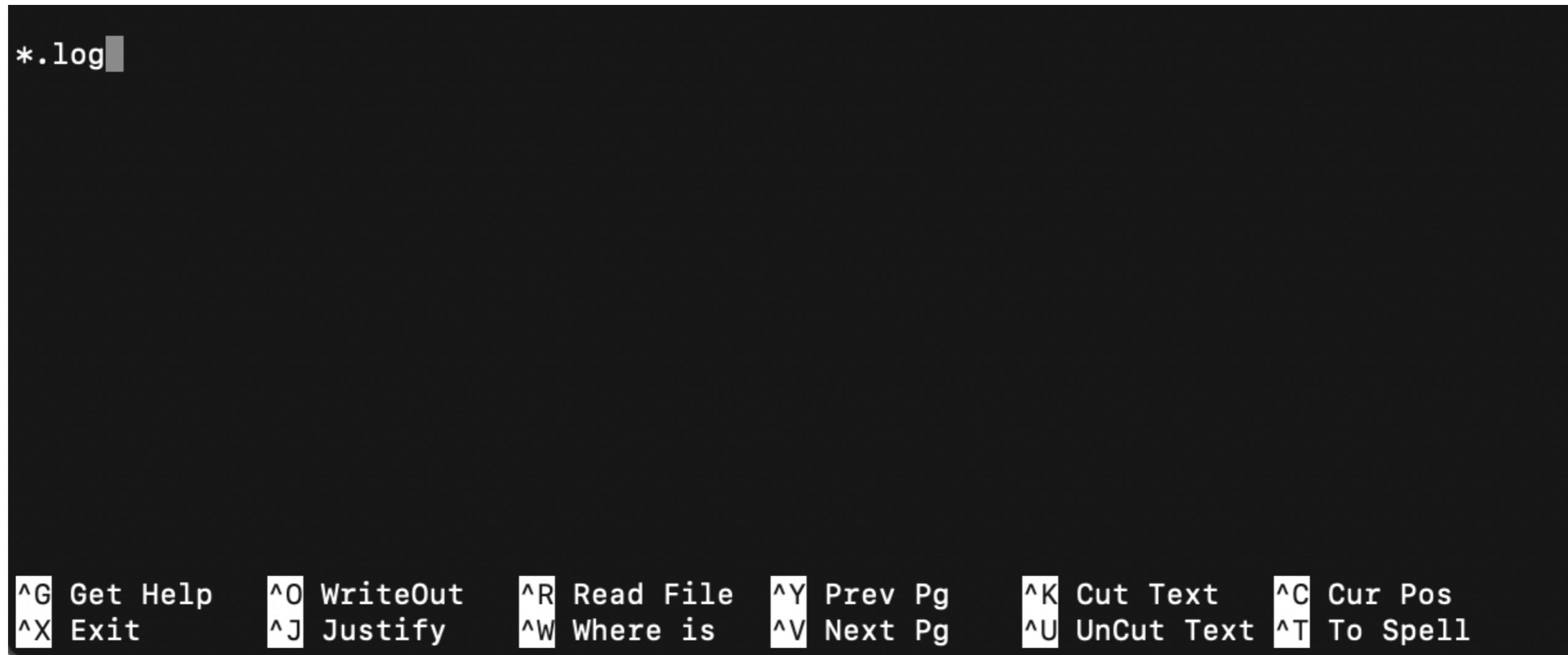
```
alias.unstage=reset HEAD
```

# Ignoring specific files

```
nano .gitignore
```

# Ignoring specific files

```
*.log
```



```
^G Get Help      ^O WriteOut     ^R Read File    ^Y Prev Pg      ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where is      ^V Next Pg      ^U UnCut Text    ^T To Spell
```

- \* = Wildcard
- Commonly ignored files: APIs, credentials, system files, software dependencies

# **Let's practice!**

**INTRODUCTION TO GIT**

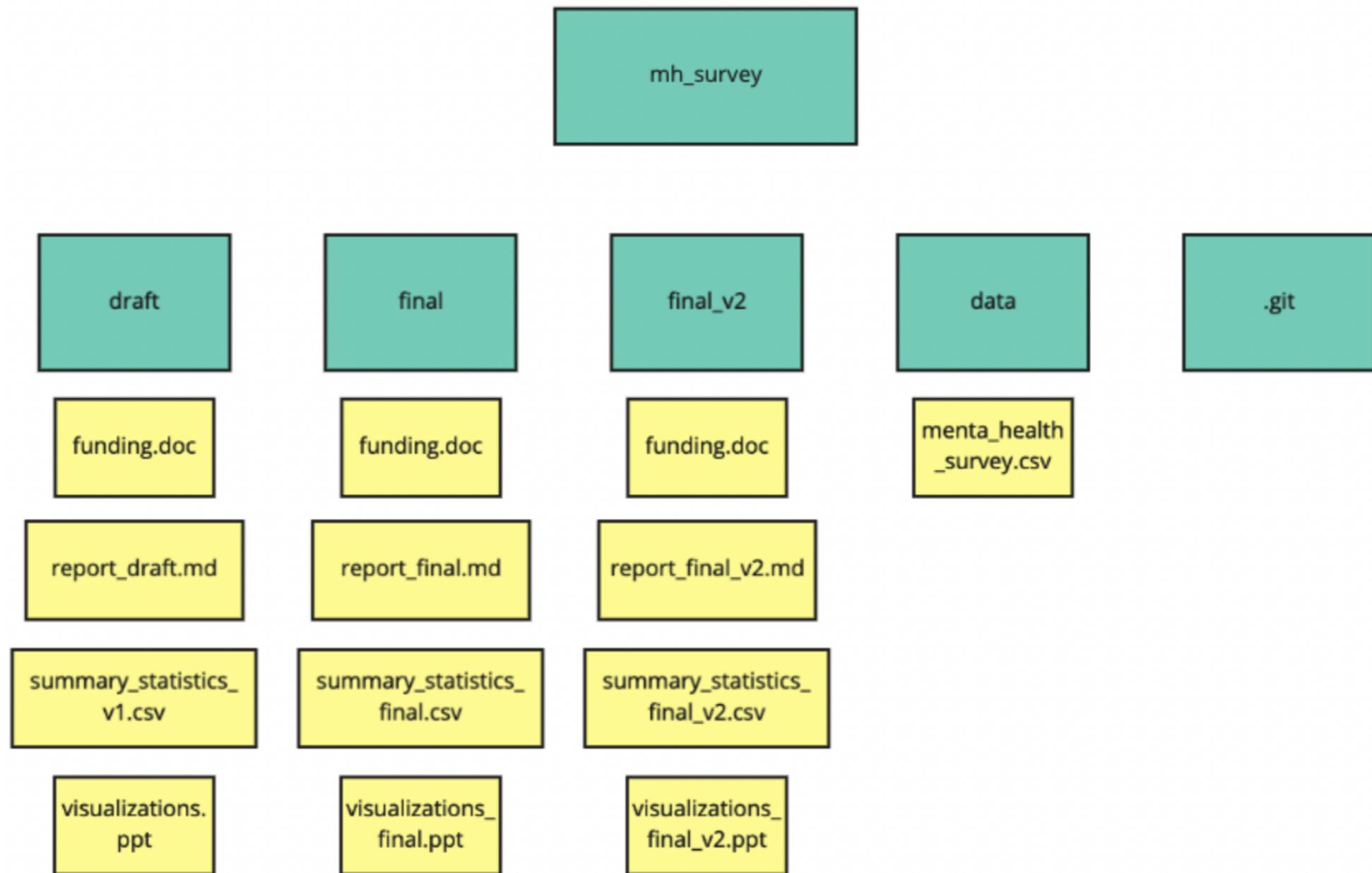
# Branches

INTRODUCTION TO GIT

**George Boorman**

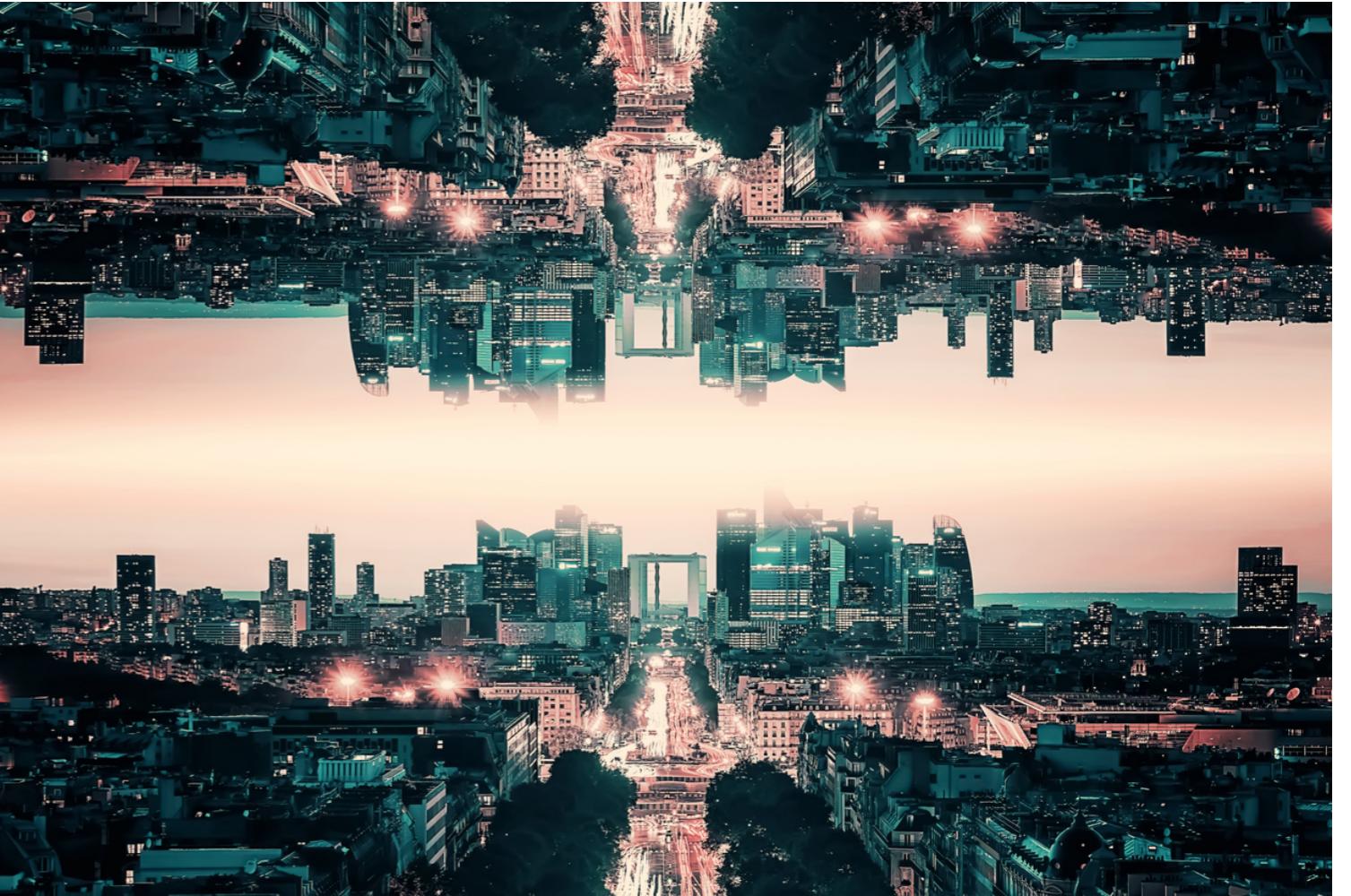
Curriculum Manager, DataCamp

# Too many subdirectories

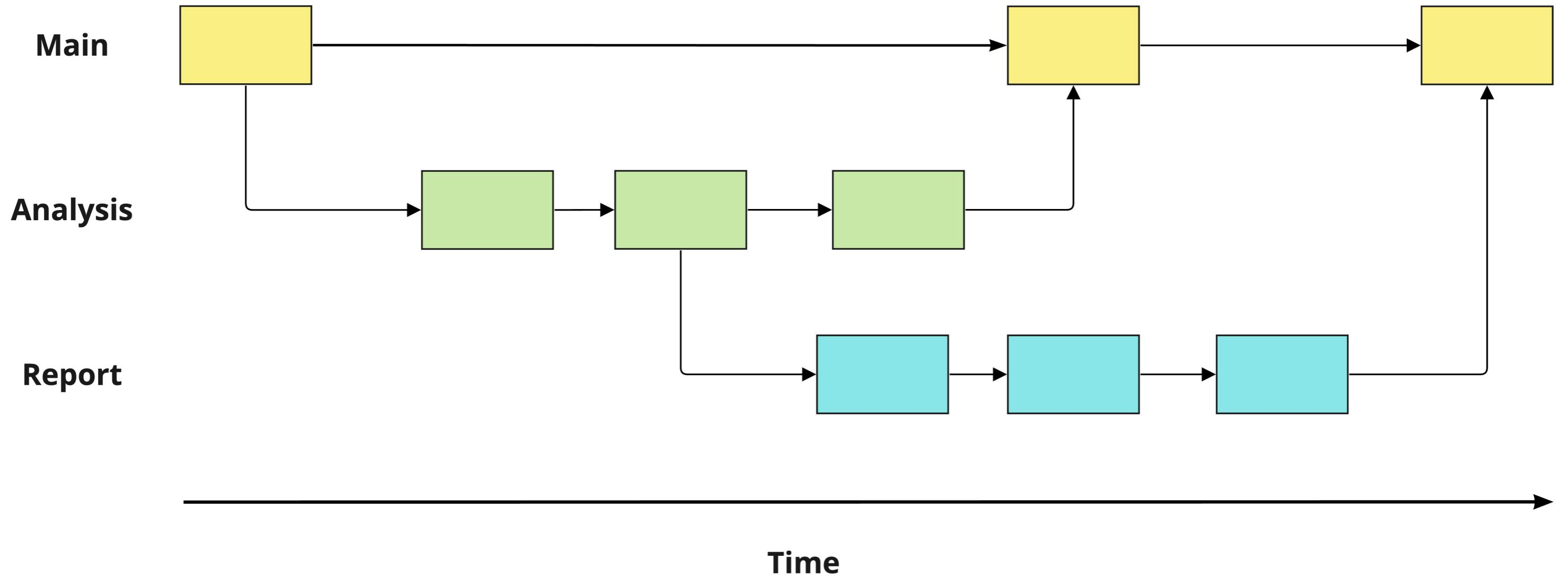


# Branches to the rescue!

- Git uses **branches** to systematically track multiple versions of files
- In each branch:
  - Some files might be the same
  - Others might be different
  - Some may not exist at all



# Visualizing branches



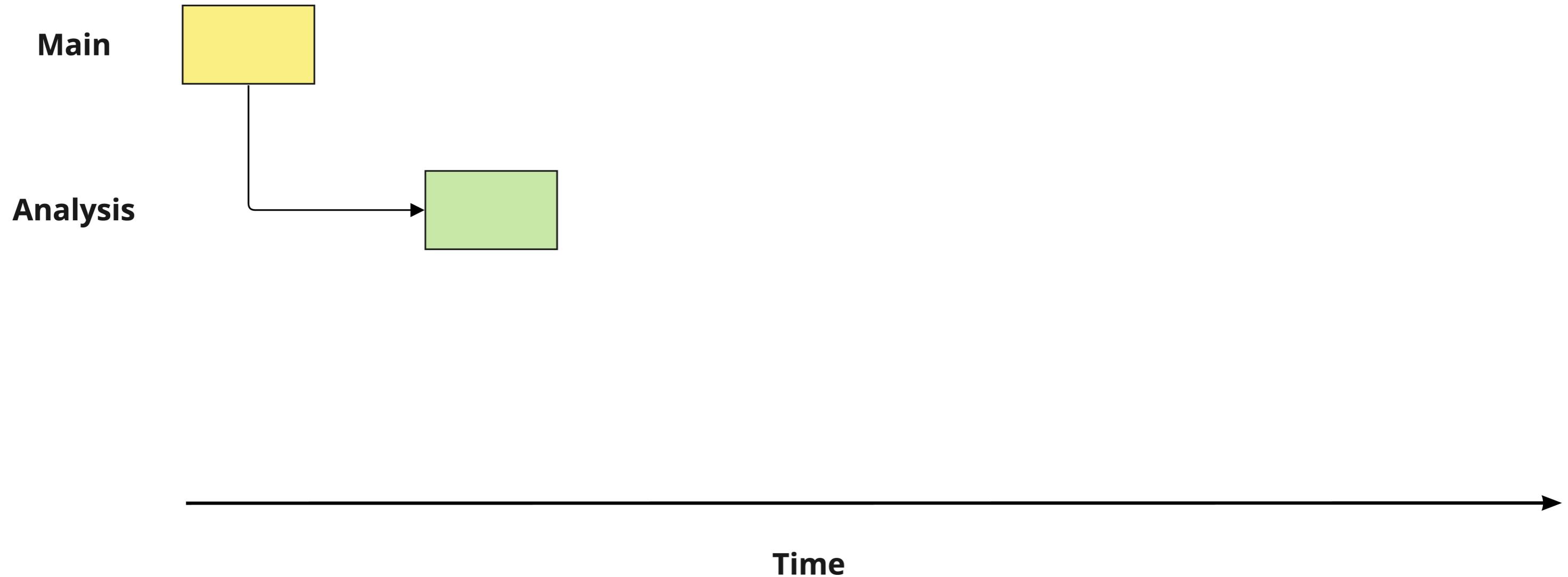
# Main branch

Main

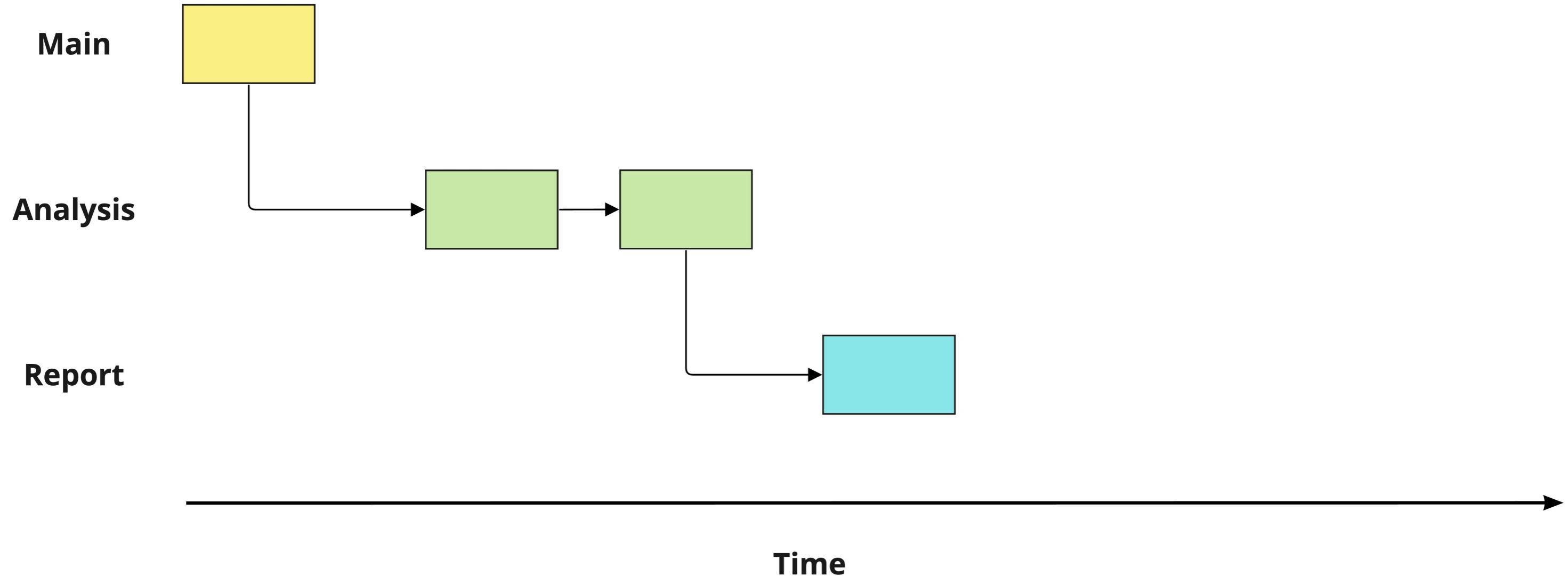


Time

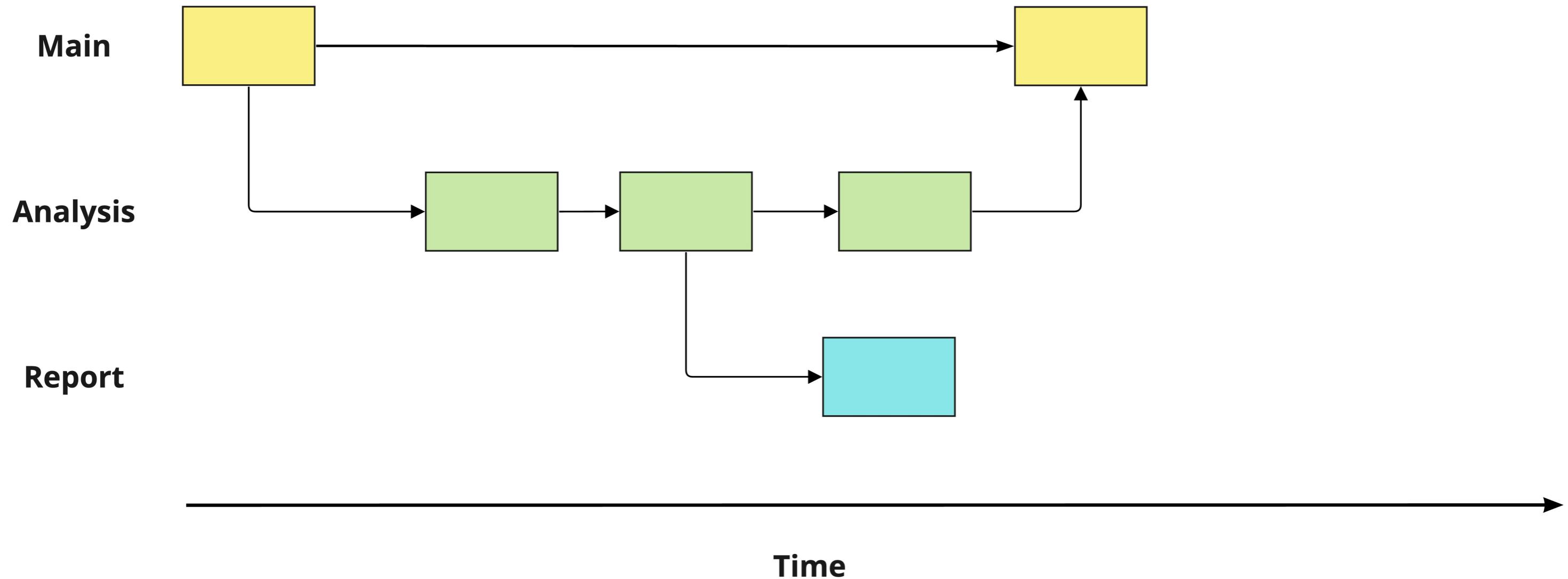
# Analysis branch



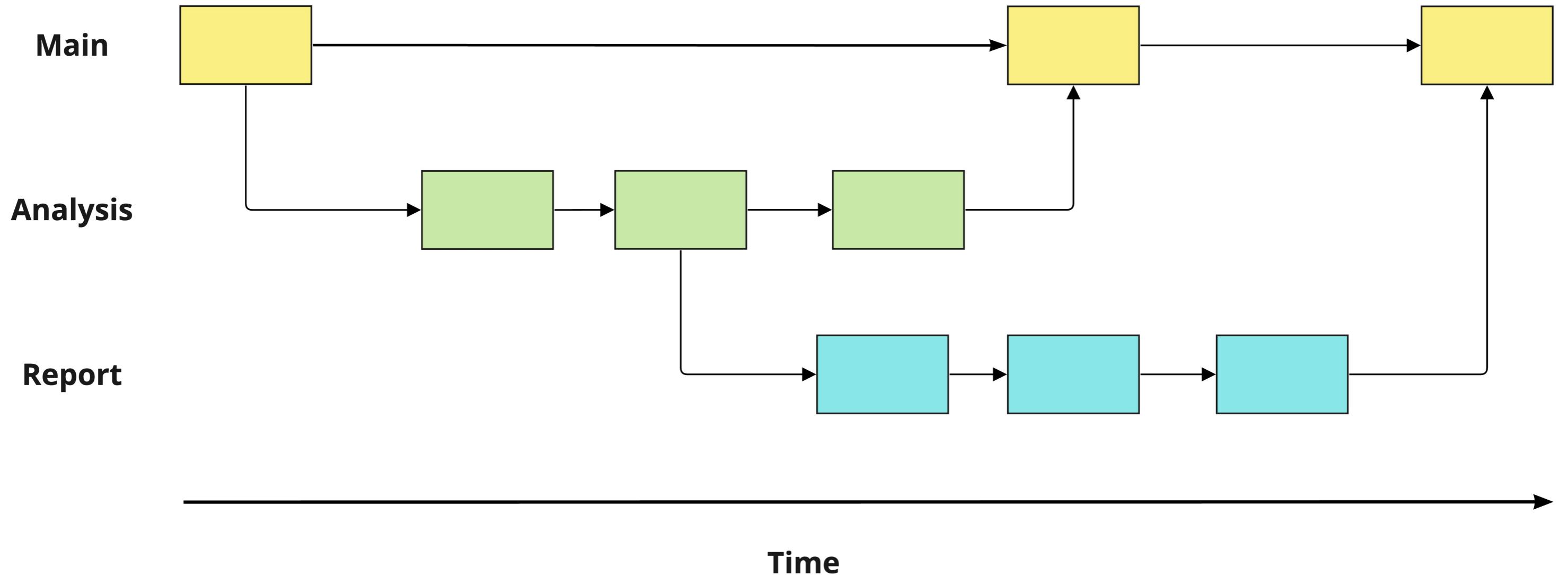
# Report branch



# Merging analysis into main

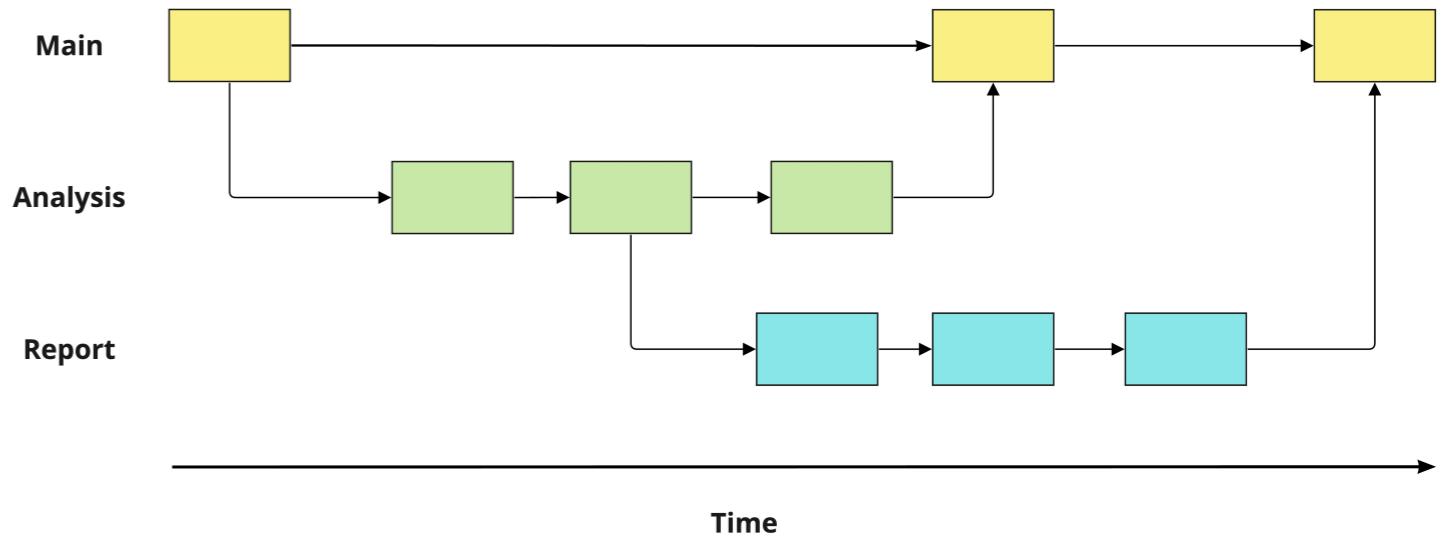


# Merging report into main



# Source and destination

- When merging two branches:
  - the commits are called parent commits
  - **source** —the branch we want to merge **from**
  - **destination** —the branch we want to merge **into**
- When merging **Analysis** into **Main**:
  - **Analysis** = **source**
  - **Main** = **destination**



# Benefits of branches

- Avoiding endless subdirectories
- Multiple users can work simultaneously
- Everything is tracked
- Minimizes the risk of conflicting versions

# Identifying branches

```
git branch
```

```
alter-report-title  
main  
* summary-statistics
```

- \* = current branch

# Creating a new branch

```
git checkout -b report
```

```
Switched to a new branch 'report'
```

```
git branch
```

```
alter-report-title
main
summary-statistics
* report
```

# The difference between branches

```
git diff main summary-statistics
```

# Comparing branches

```
diff --git a/bin/summary b/bin/summary
new file mode 100755
index 000000..9d6e2fa
--- /dev/null
+++ b/bin/summary
@@ -0,0 +1,44 @@
+Summary statistics
+
+Age:
+count: 49.00
+mean: 31.82
+std: 6.72
+min: 18.00
+25%: 28.00
+50%: 31.00
+75%: 35.00
+max: 46.00
+
```

# **Let's practice!**

**INTRODUCTION TO GIT**

# Working with branches

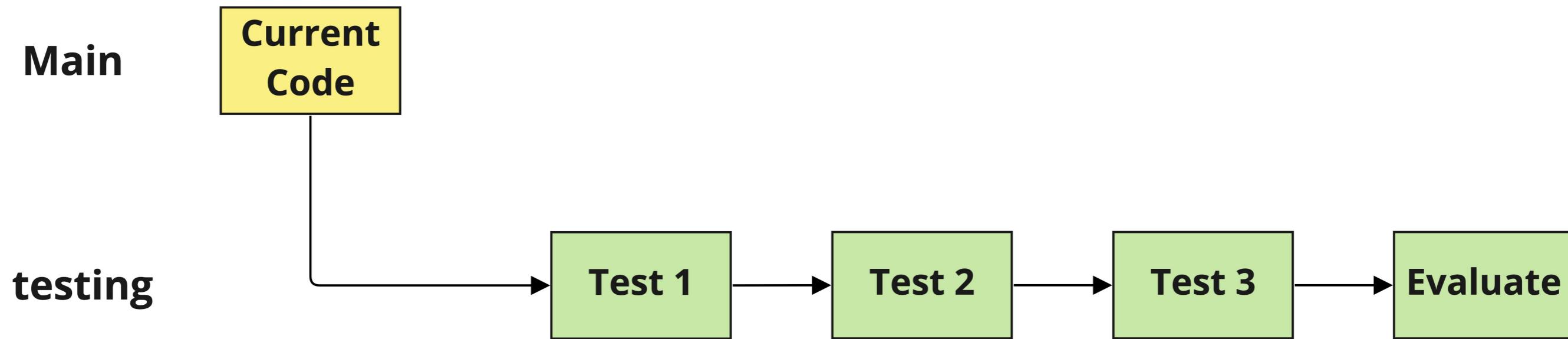
INTRODUCTION TO GIT

**George Boorman**

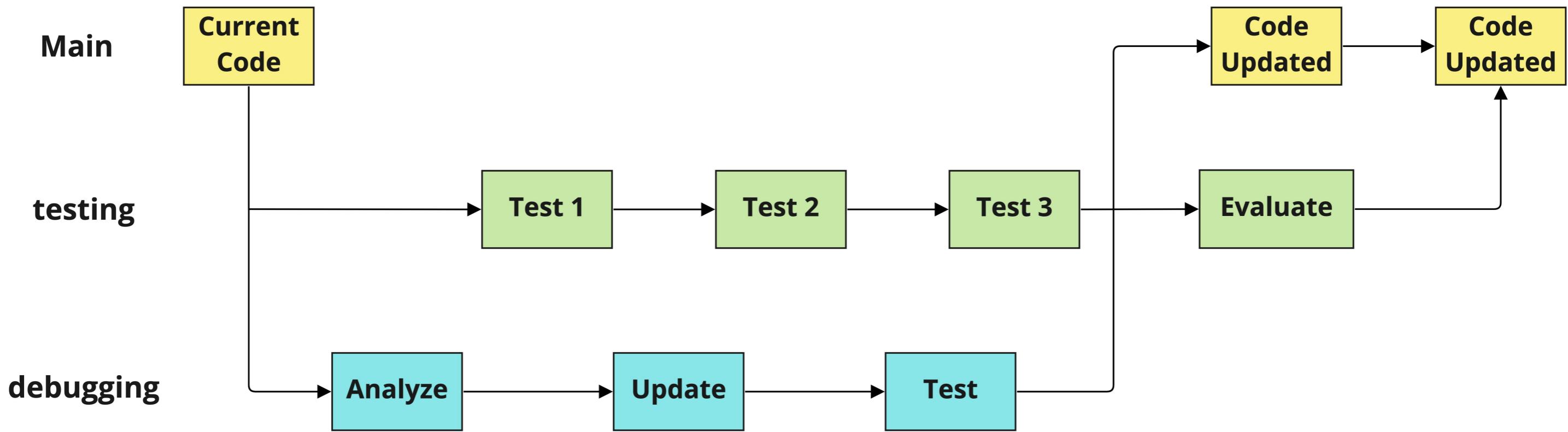
Curriculum Manager, DataCamp

# Why do we need to switch branches?

- Common to work on different components of a project simultaneously
- Branches allow us to keep making progress concurrently



# Why do we need to switch branches?



# How do we switch branches?

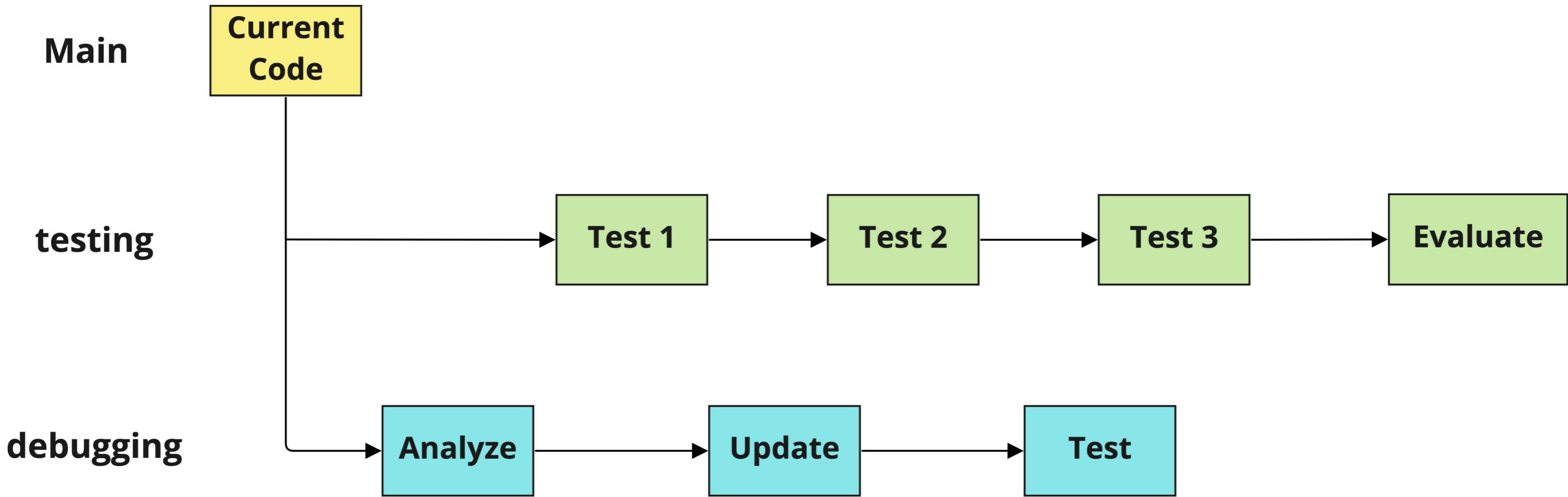
- `git checkout -b new_branch` to create a new branch

```
git checkout debugging
```

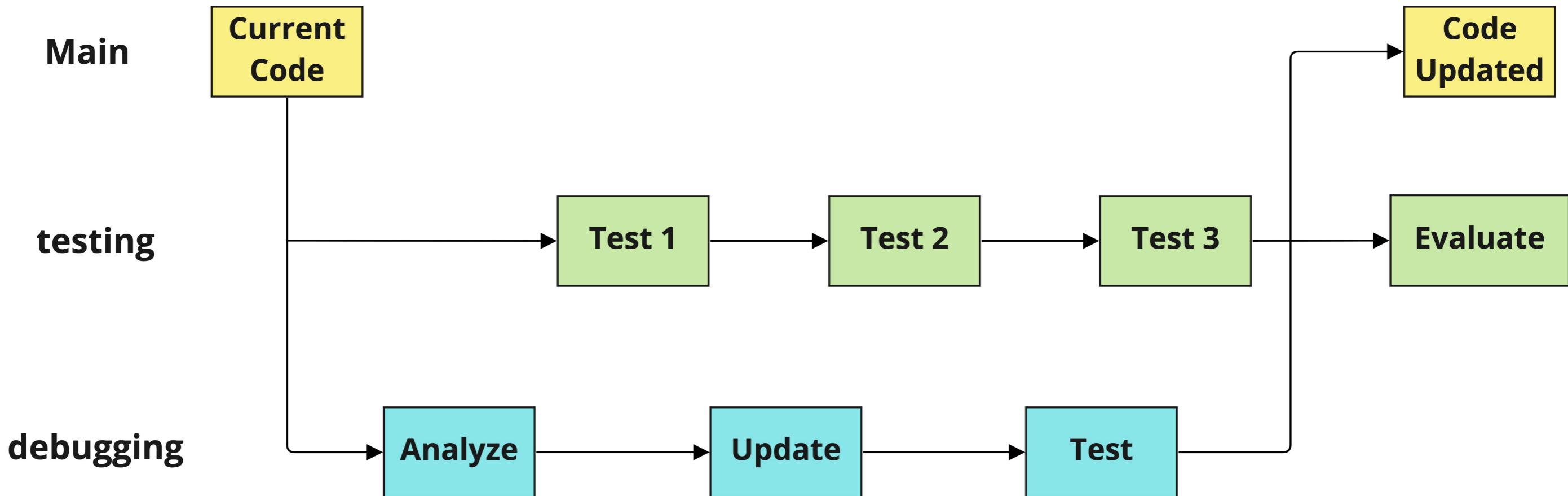
```
git branch
```

```
* debugging  
  main  
  summary-statistics
```

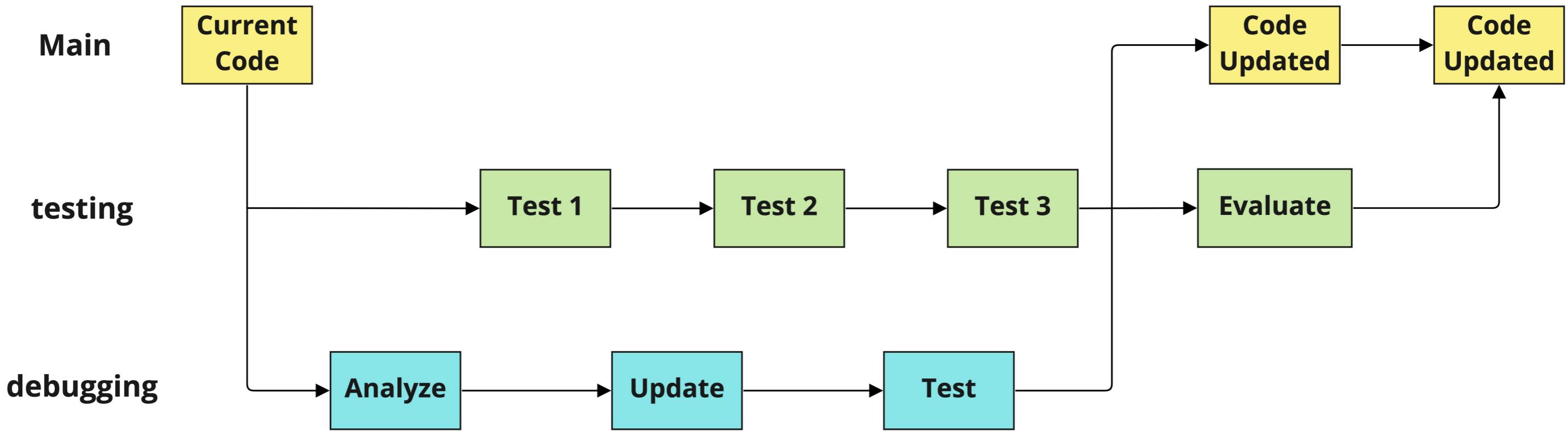
# Next step: merge



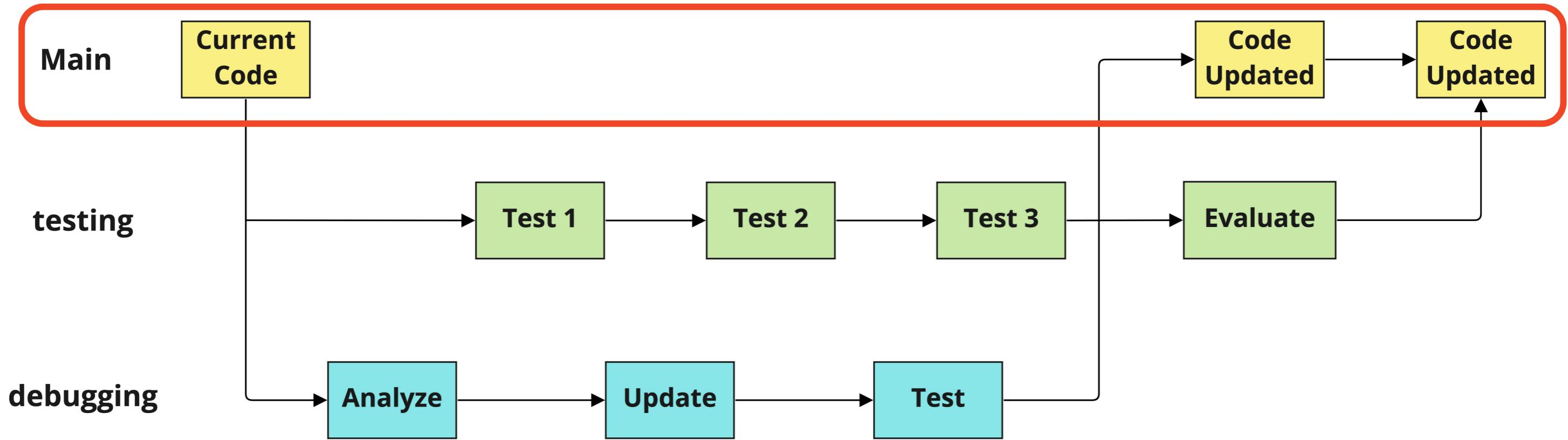
# Next step: merge



# Next step: merge



# Why do we merge branches?



# Why do we merge branches?

- `main` = ground truth
- Each branch should be for a specific task
- Once the task is complete we should merge our changes into `main`
  - to keep it up to date and accurate

# Merging branches

```
git merge source destination
```

- To merge `summary-statistics` into `main`

```
git merge summary-statistics main
```

# Merge output

## Commit hashes



```
Updating dc9d8fa..cef5ad8
Fast-forward
  bin/summary      | 44 ++++++=====
  results/summary.txt | 0
  2 files changed, 44 insertions(+)
  create mode 100755 bin/summary
  create mode 100644 results/summary.txt
```

# Merge output

Type of merge →

```
Updating dc9d8fa..cef5ad8
Fast-forward
  bin/summary      | 44 ++++++=====
  results/summary.txt | 0
  2 files changed, 44 insertions(+)
  create mode 100755 bin/summary
  create mode 100644 results/summary.txt
```

# Merge output

**Number of lines changed**

```
Updating dc9d8fa..cef5ad8
Fast-forward
  bin/summary      | 44 ++++++++|||||||||||||||||||||+
  results/summary.txt | 0
  2 files changed, 44 insertions(+)
  create mode 100755 bin/summary
  create mode 100644 results/summary.txt
```

# Merge output

Files modified →

```
Updating dc9d8fa..cef5ad8
Fast-forward
  bin/summary      | 44 ++++++=====
  results/summary.txt | 0
  2 files changed, 44 insertions(+)
create mode 100755 bin/summary
create mode 100644 results/summary.txt
```

# **Let's practice!**

**INTRODUCTION TO GIT**

# Handling conflict

## INTRODUCTION TO GIT

**George Boorman**

Curriculum Manager, DataCamp

# What is a conflict?

- A) Write report.
- B) Submit report.

```
git add todo.txt
```

```
git commit -m "Add todo list"
```

```
git checkout -b update
```

- A) Write report.
- B) Submit report.
- C) Submit expenses.

# What is a conflict?

```
git add todo.txt  
git commit -m "Reminder to submit expenses"  
git checkout main
```

c) Submit expenses.

# Conflict

Main branch todo.txt

c) Submit expenses.

Update branch todo.txt

A) Write report.  
B) Submit report.  
C) Submit expenses.

# Attempting to merge a conflict

```
git merge update main
```

```
CONFLICT (add/add): Merge conflict in todo.txt
```

```
Auto-merging todo.txt
```

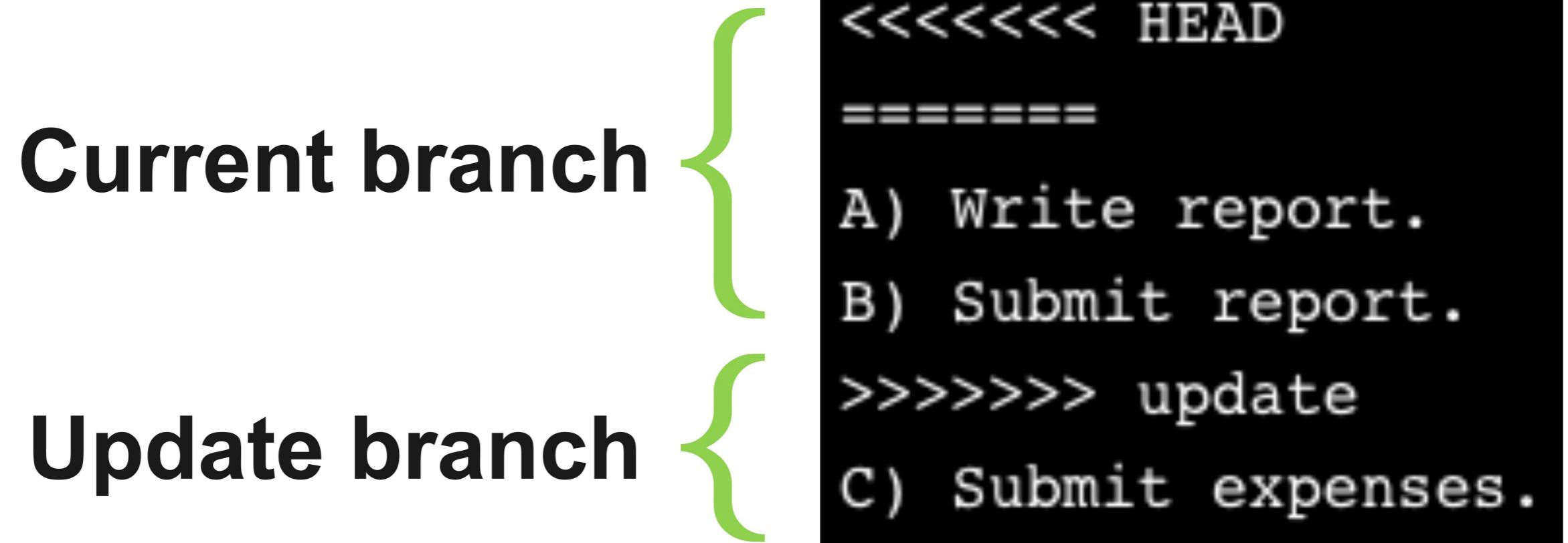
```
Automatic merge failed; fix conflicts and then commit the result.
```

# Git conflicts

```
nano todo.txt
```

```
<<<<< HEAD
=====
A) Write report.
B) Submit report.
>>>>> update
C) Submit expenses.
```

# Git conflicts



# Conflict web editor

```
<<<<< HEAD  
=====  
A) Write report.  
B) Submit report.  
>>>>> update  
C) Submit expenses.
```

**^G** Get Help

**^X** Exit

**^O** Write Out

**^R** Read File

**^W** Where Is

**^V** Replace

**^K** Cut Text

**^U** Uncut Text

**^J** Justify

**^T** To Spell

# Merging the branches

```
git add todo.txt
```

```
git commit -m "Resolving todo.txt conflict"
```

```
git merge update main
```

```
Already up to date.
```

- Large conflicts can be quite intimidating!

# How do we avoid conflicts?

- Prevention is better than cure!
- Use each branch for a specific task
- Avoid editing a file in multiple branches
- Doesn't guarantee we'll avoid conflicts
  - but it does reduce the risk

# **Let's practice!**

**INTRODUCTION TO GIT**