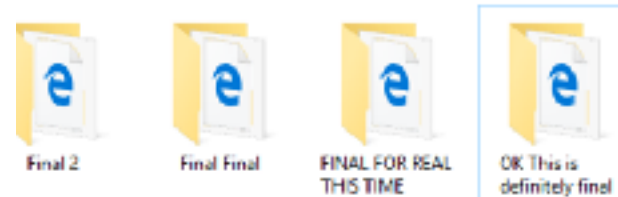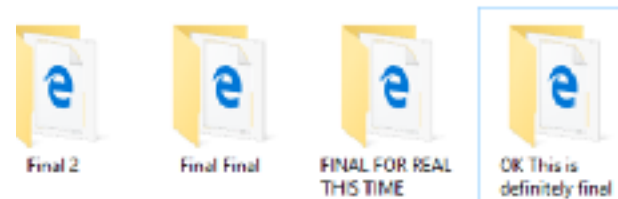# GIT: Push 'n Run

Erik Regla

ＧＩＴで何？

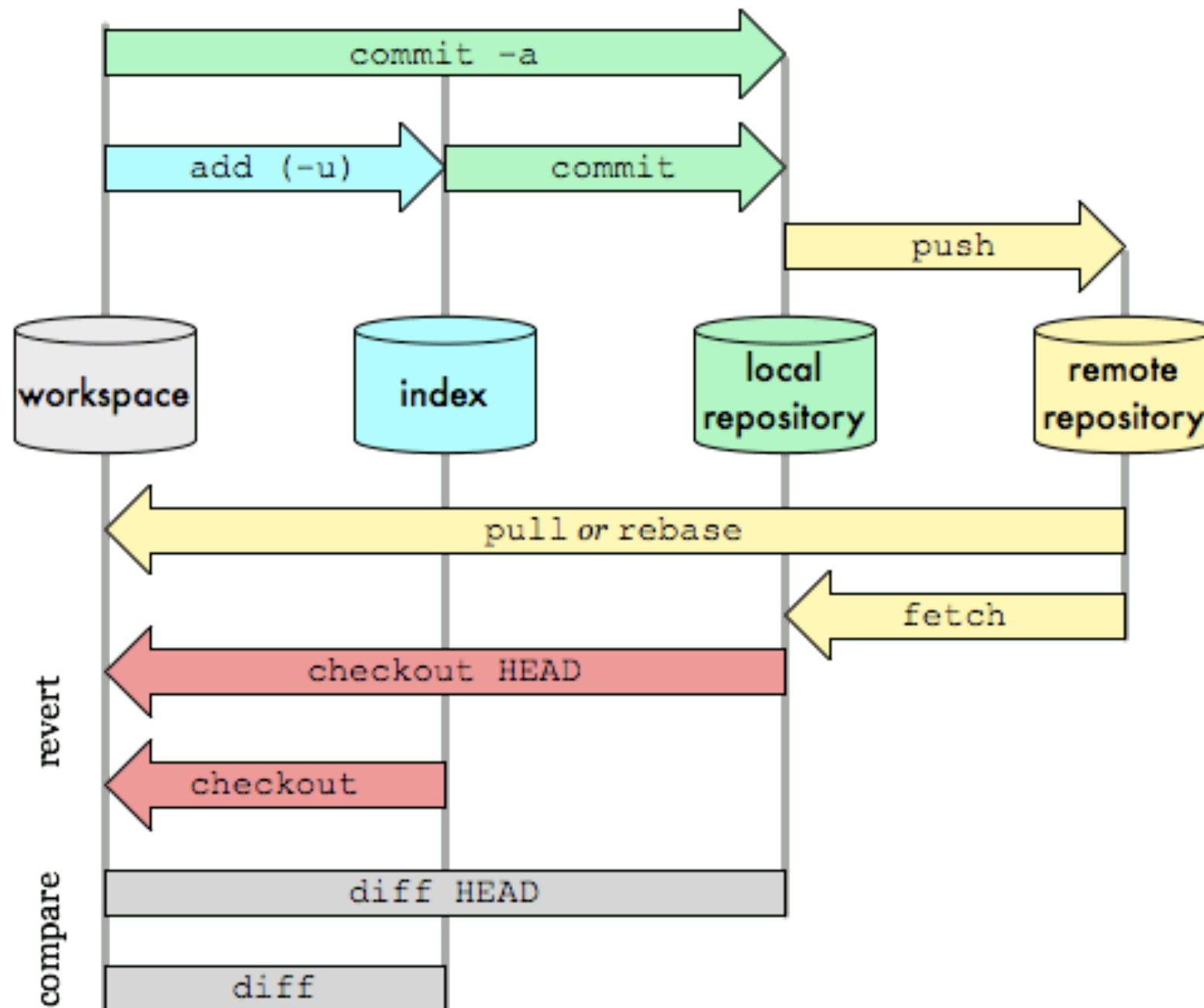- Gitとは、分散型バージョン管理システムの一つで、もともとLinuxのソースコードを効果的に管理するために開発されました。

What is GIT?

- Git is a distributed version control system, originally developed to effectively manage Linux source code.

GIT Architecture Overview

GIT/VersionControl basic terms

- A set of changes over the **repository tracked** files is called a **commit.**

- The users can **push** changes from the local repository onto the remote repository and **pull** changes from remote repo to local.

- Users can also **merge** contents of two commits in a single one. This is commonly used when performing **branching**, which can be seen as separate 'workspaces' for performing version control.

- Sometimes, when the commits involved in a merge operation changes the same lines or the same blocks, it's said to be a **conflict** between those two commits, because git is not able to resolve which commit has precedence on each conflicted line.

GIT/VersionControl basic commands

- **Initialise repository:** `git init`

- **track file:** `git add <file>`

- **commit:** `git commit -a -m <commit_message>`

- **clone:** `git clone <repo_address>`

- **push:** `git push`

- **pull:** `git pull`

- **create new branch:** `git checkout -b new_branch`

- **change to new branch:** `git checkout new_branch`

- **check status:** `git status`

- **merge other branch into current:** `git merge <branch_to_merge>`
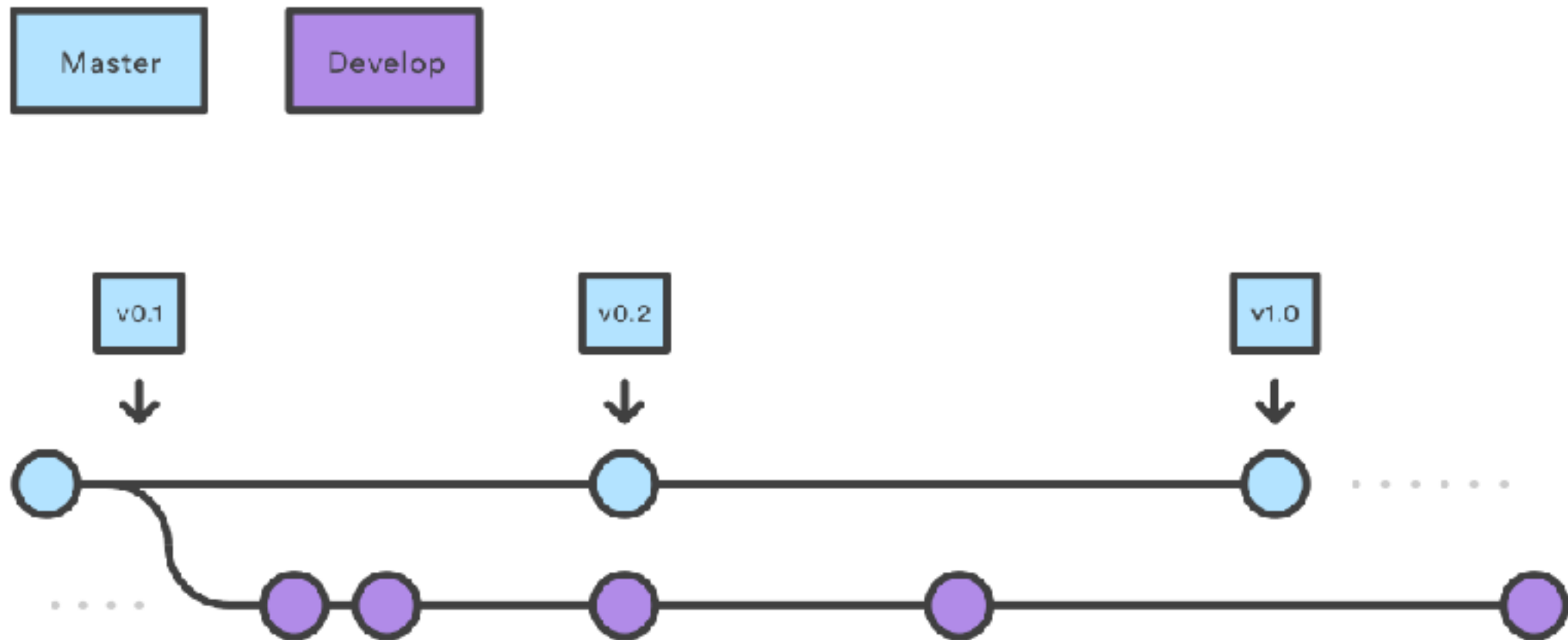
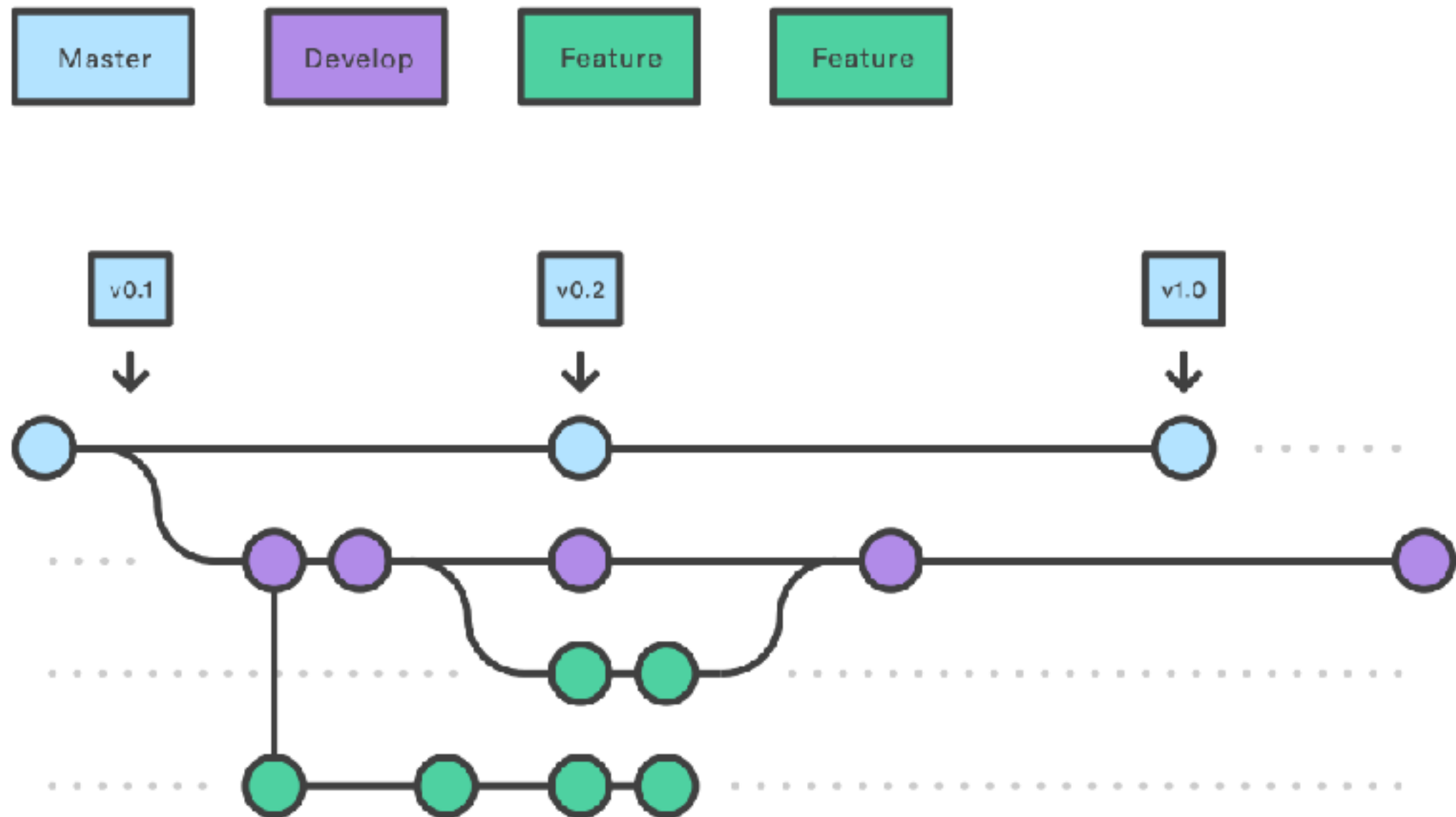# Demo

How branching is supposed to work
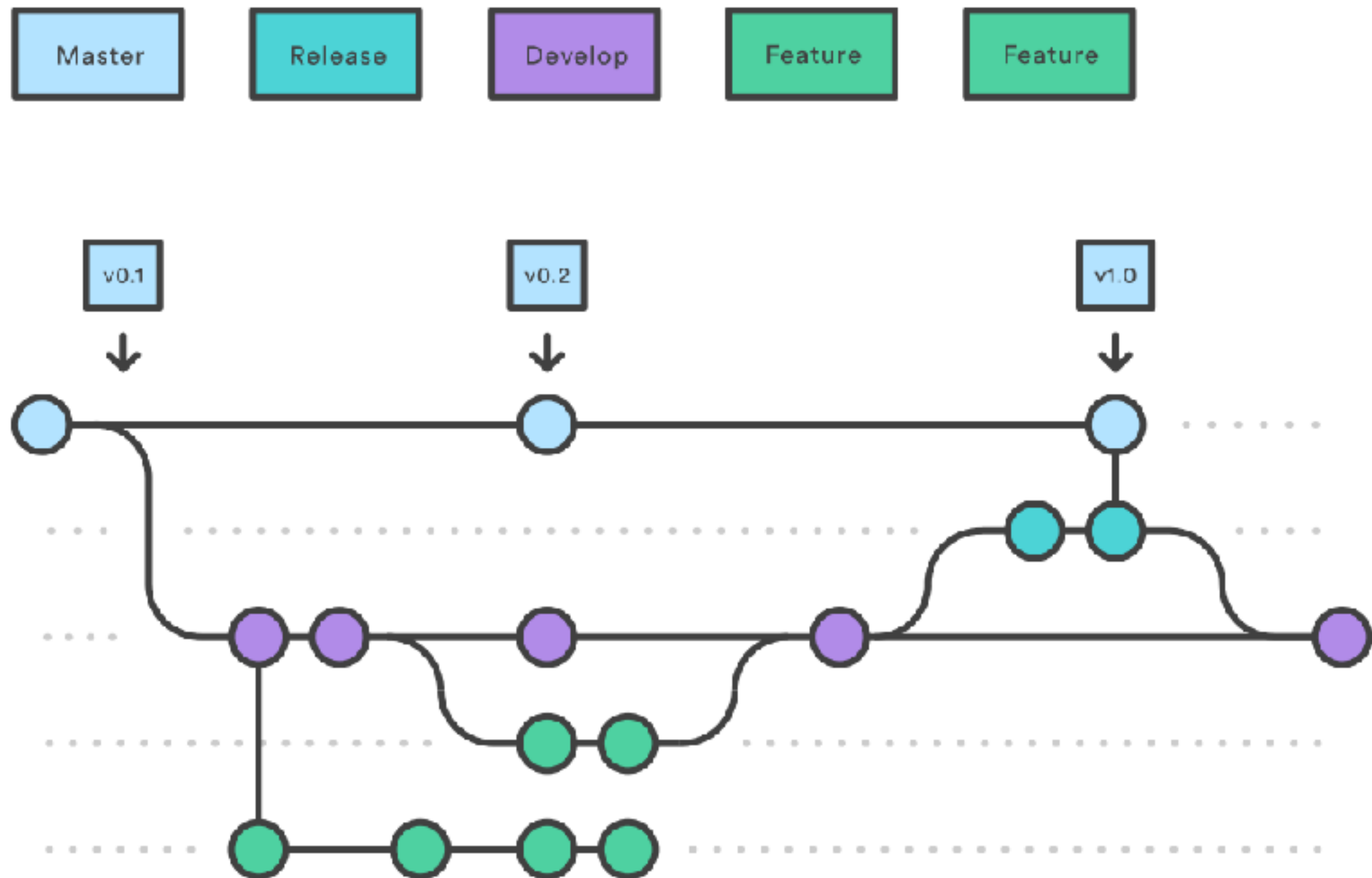
How branching works in practice
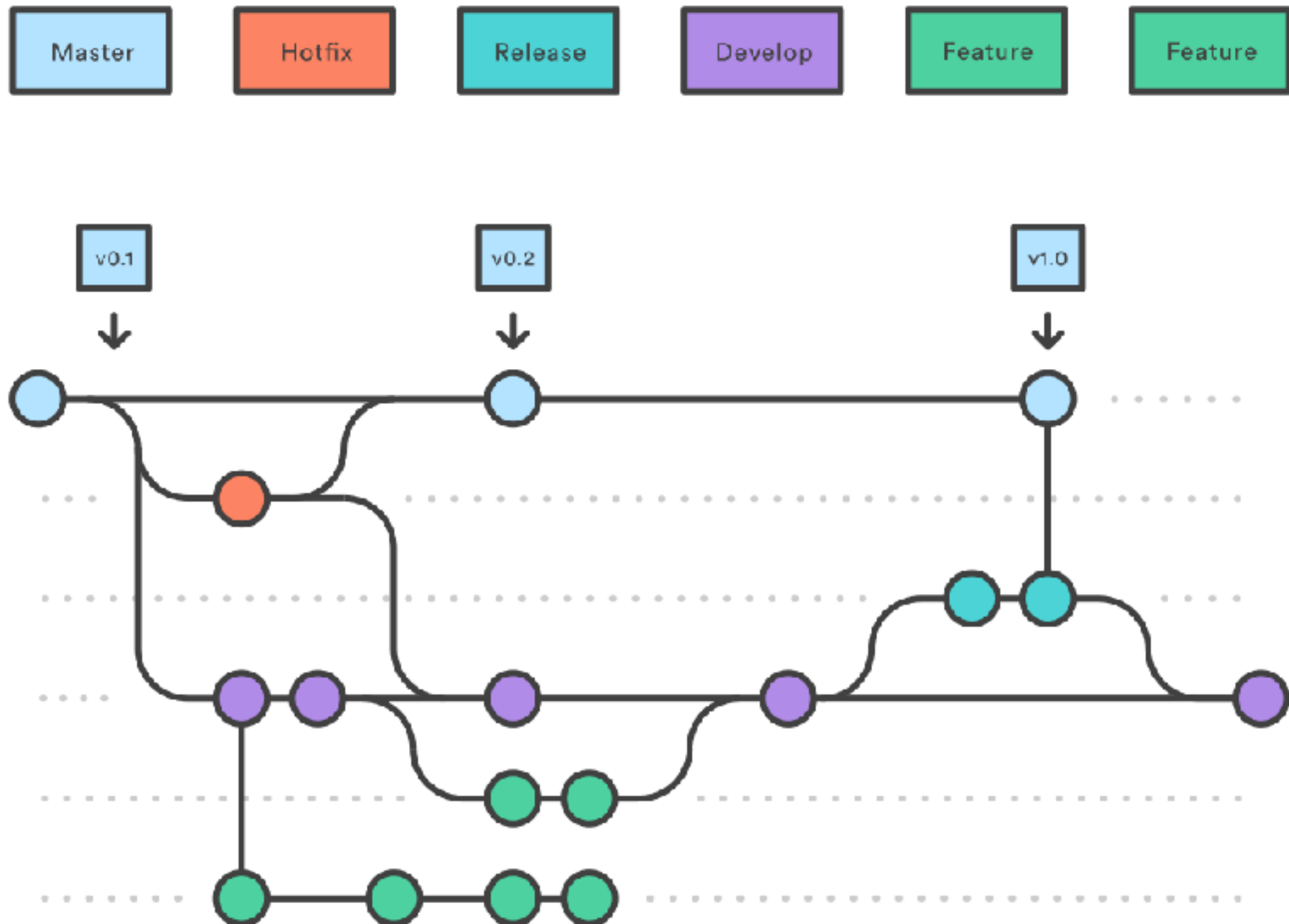
GIT-Flow: Historical Branches

GIT-Flow: Future Branches

GIT-Flow: Release Branches

GIT-Flow: Maintenance Branches

Managing Conflicts

```c
#include <stdio.h>

int sum(int a, int b){
    // someone has to implement this.
    return 0;
}

int main(){
    int a = 0, b = 0;
    printf("Ingrese a: ");
    scanf("%d\n", &a);
    printf("Ingrese b: ");
    scanf("%d\n", &b);
    int result = sum(a, b);
    printf("Resultado a+b: %d", result);
    return 0;
}
```
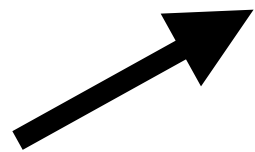
master

Managing Conflicts

```c
#include <stdio.h>

int sum(int a, int b){
    // someone has to implement this.
    return 0;
}

int main(){
    int a = 0, b = 0;
    printf("Ingrese a: ");
    scanf("%d\n", &a);
    printf("Ingrese b: ");
    scanf("%d\n", &b);
    int result = sum(a, b);
    printf("Resultado a+b: %d", result);
    return 0;
}
```

master

Managing Conflicts

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      // someone has to im
5      return 0;
6  }
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: "
11     scanf("%d\n", &a);
12     printf("Ingrese b: "
13     scanf("%d\n", &b);
14     int result = sum(a,
15     printf("Resultado a+
16     return 0;
17  }
```

master

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      return a+b;
5  }
6
7  int main(){
8      int a = 0, b = 0;
9      printf("Ingrese a: ");
10     scanf("%d\n", &a);
11     printf("Ingrese b: ");
12     scanf("%d\n", &b);
13     int result = sum(a, b);
14     printf("Resultado a+b: %d", result);
15     return 0;
16  }
```

developer_1

Managing Conflicts
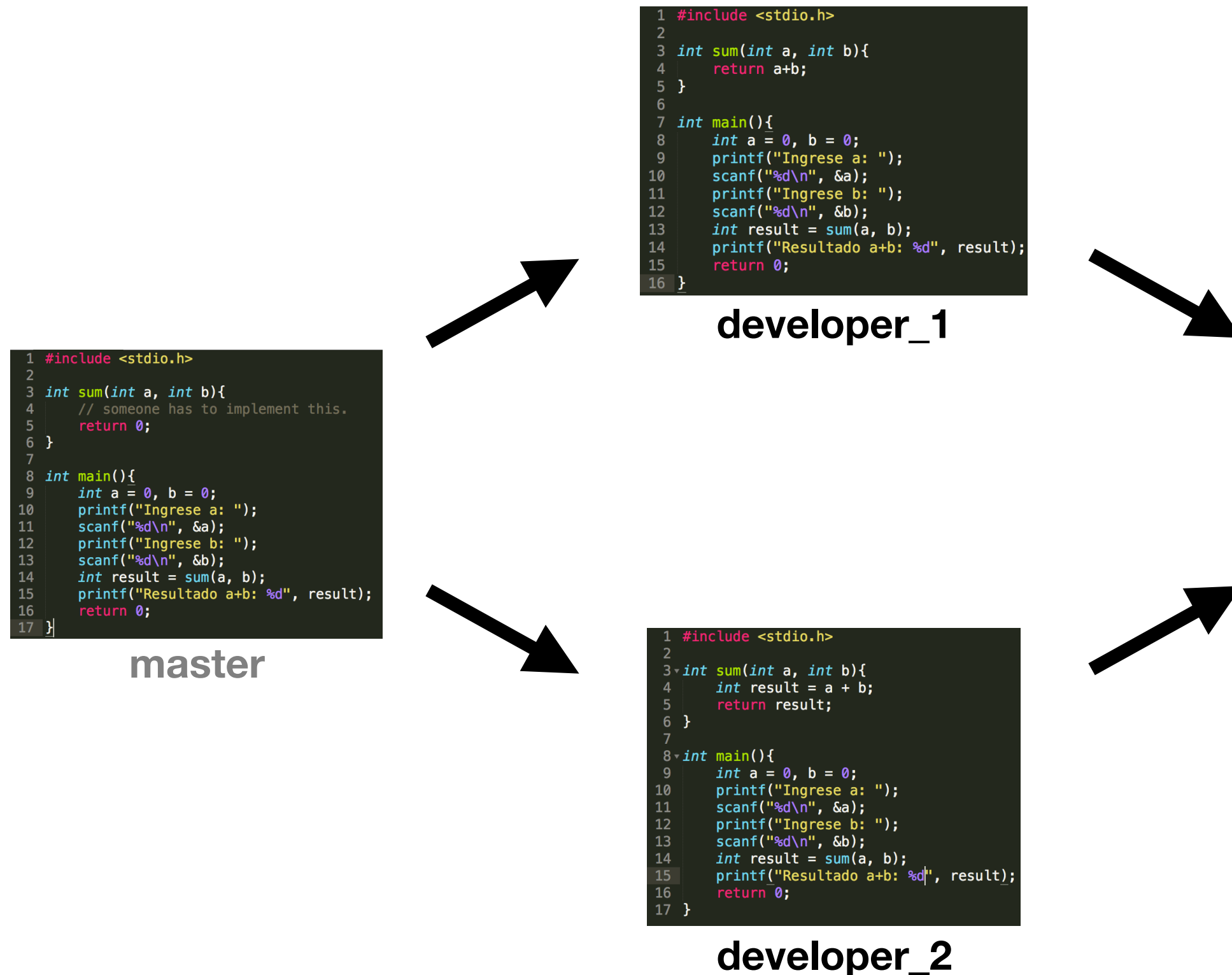


```
 1  #include <stdio.h>
 2
 3  int sum(int a, int b){
 4      return a+b;
 5  }
 6
 7  int main(){
 8      int a = 0, b = 0;
 9      printf("Ingrese a: ");
10      scanf("%d\n", &a);
11      printf("Ingrese b: ");
12      scanf("%d\n", &b);
13      int result = sum(a, b);
14      printf("Resultado a+b: %d", result);
15      return 0;
16  }
```

developer_1

```
 1  #include <stdio.h>
 2
 3  int sum(int a, int b){
 4      // someone has to implement this.
 5      return 0;
 6  }
 7
 8  int main(){
 9      int a = 0, b = 0;
10      printf("Ingrese a: ");
11      scanf("%d\n", &a);
12      printf("Ingrese b: ");
13      scanf("%d\n", &b);
14      int result = sum(a, b);
15      printf("Resultado a+b: %d", result);
16      return 0;
17  }
```

master

Managing Conflicts

```
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      int result = a + b;
5      return result;
6  }
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: ");
11     scanf("%d\n", &a);
12     printf("Ingrese b: ");
13     scanf("%d\n", &b);
14     int result = sum(a, b);
15     printf("Resultado a+b: %d", result);
16     return 0;
17 }
```

master

```
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      // someone has to i
5      return 0;
6  }
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: "
11     scanf("%d\n", &a);
12     printf("Ingrese b: "
13     scanf("%d\n", &b);
14     int result = sum(a,
15     printf("Resultado a+
16     return 0;
17 }
```
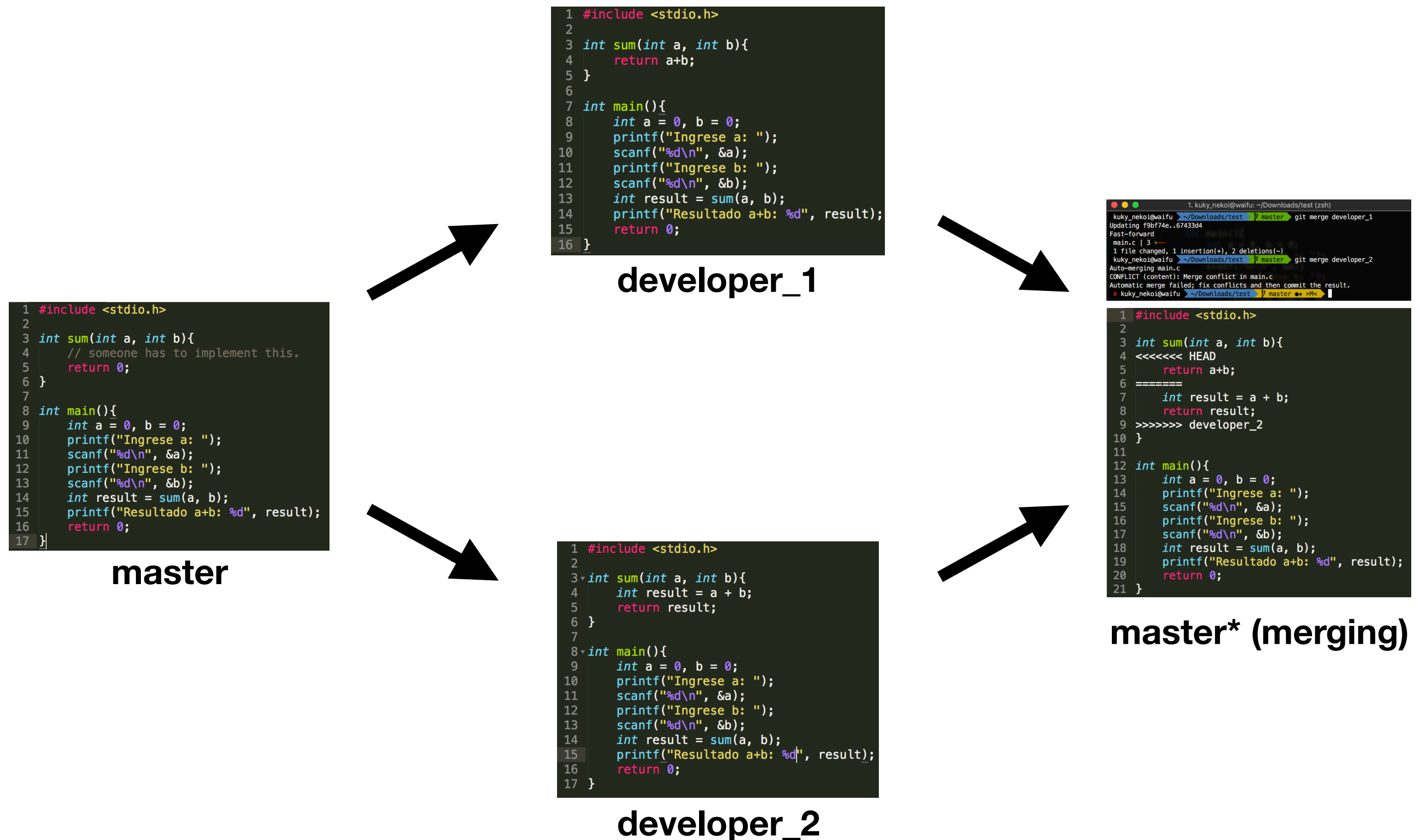
developer_2

Managing Conflicts

developer_1

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      return a+b;
5  }
6
7  int main(){
8      int a = 0, b = 0;
9      printf("Ingrese a: ");
10     scanf("%d\n", &a);
11     printf("Ingrese b: ");
12     scanf("%d\n", &b);
13     int result = sum(a, b);
14     printf("Resultado a+b: %d", result);
15     return 0;
16 }
```

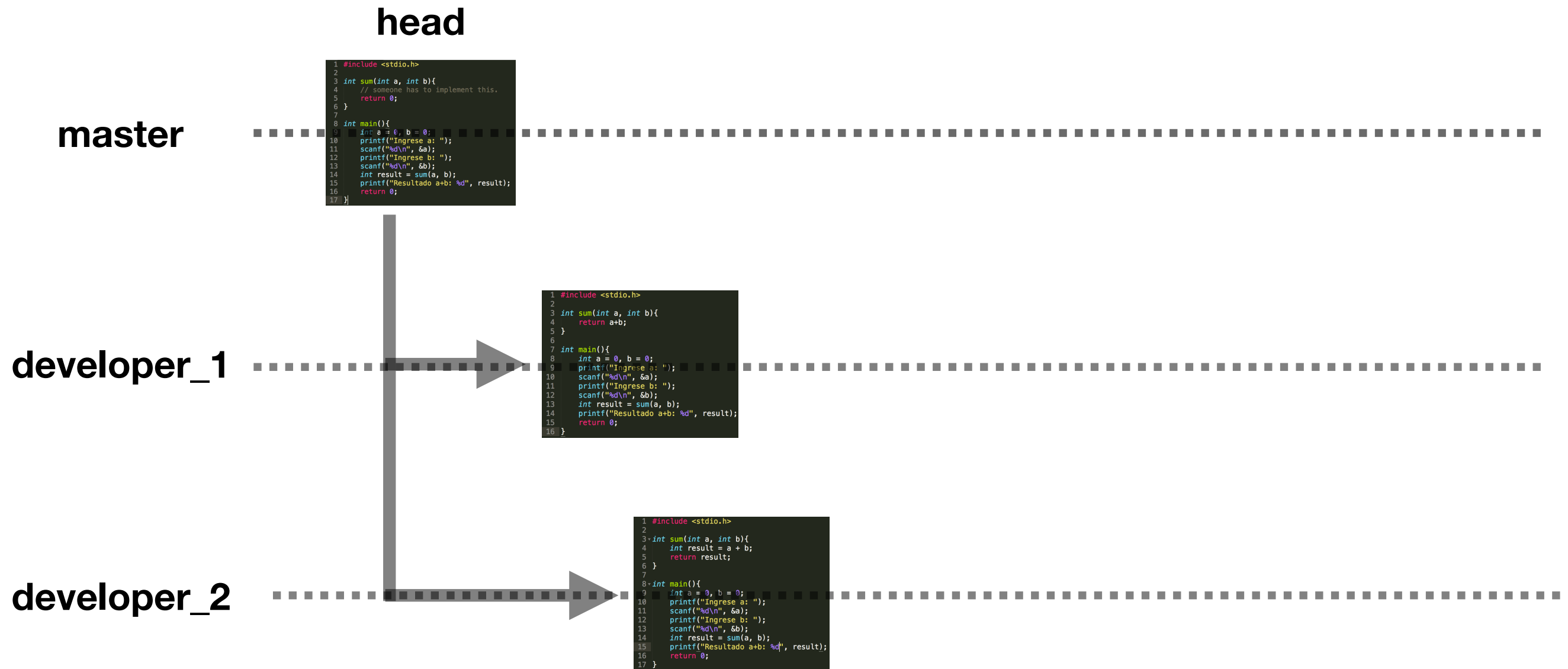**master**

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      // someone has to implement this.
5      return 0;
6  }
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: ");
11     scanf("%d\n", &a);
12     printf("Ingrese b: ");
13     scanf("%d\n", &b);
14     int result = sum(a, b);
15     printf("Resultado a+b: %d", result);
16     return 0;
17 }
```
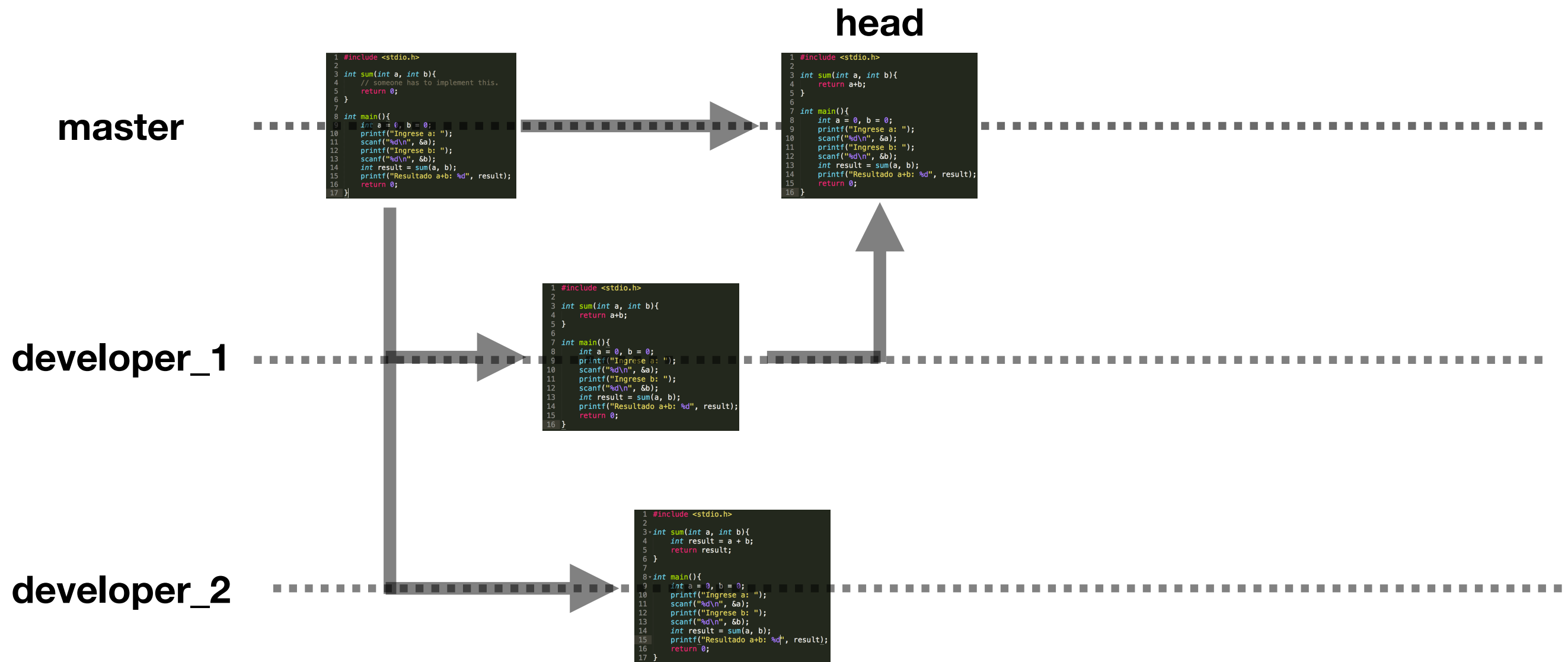
developer_2

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      int result = a + b;
5      return result;
6  }
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: ");
11     scanf("%d\n", &a);
12     printf("Ingrese b: ");
13     scanf("%d\n", &b);
14     int result = sum(a, b);
15     printf("Resultado a+b: %d", result);
16     return 0;
17 }
```
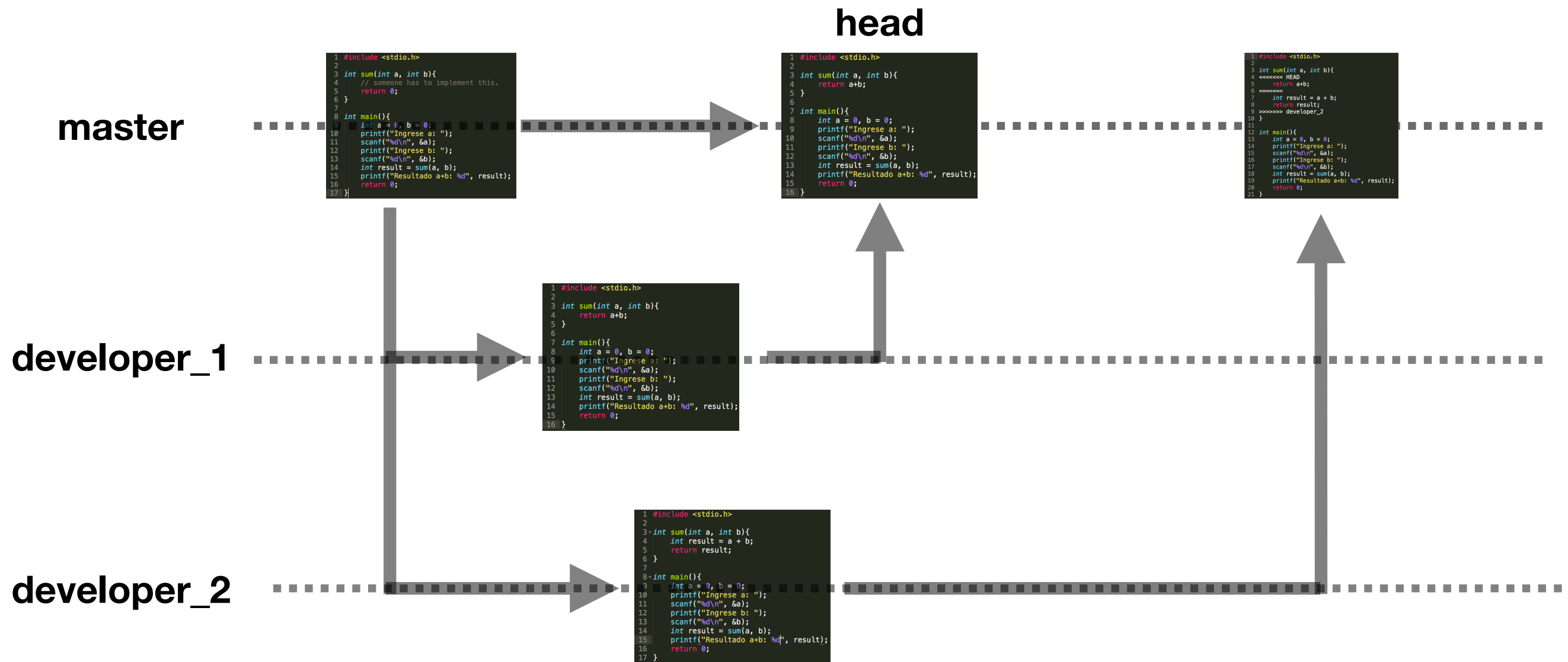
Managing Conflicts
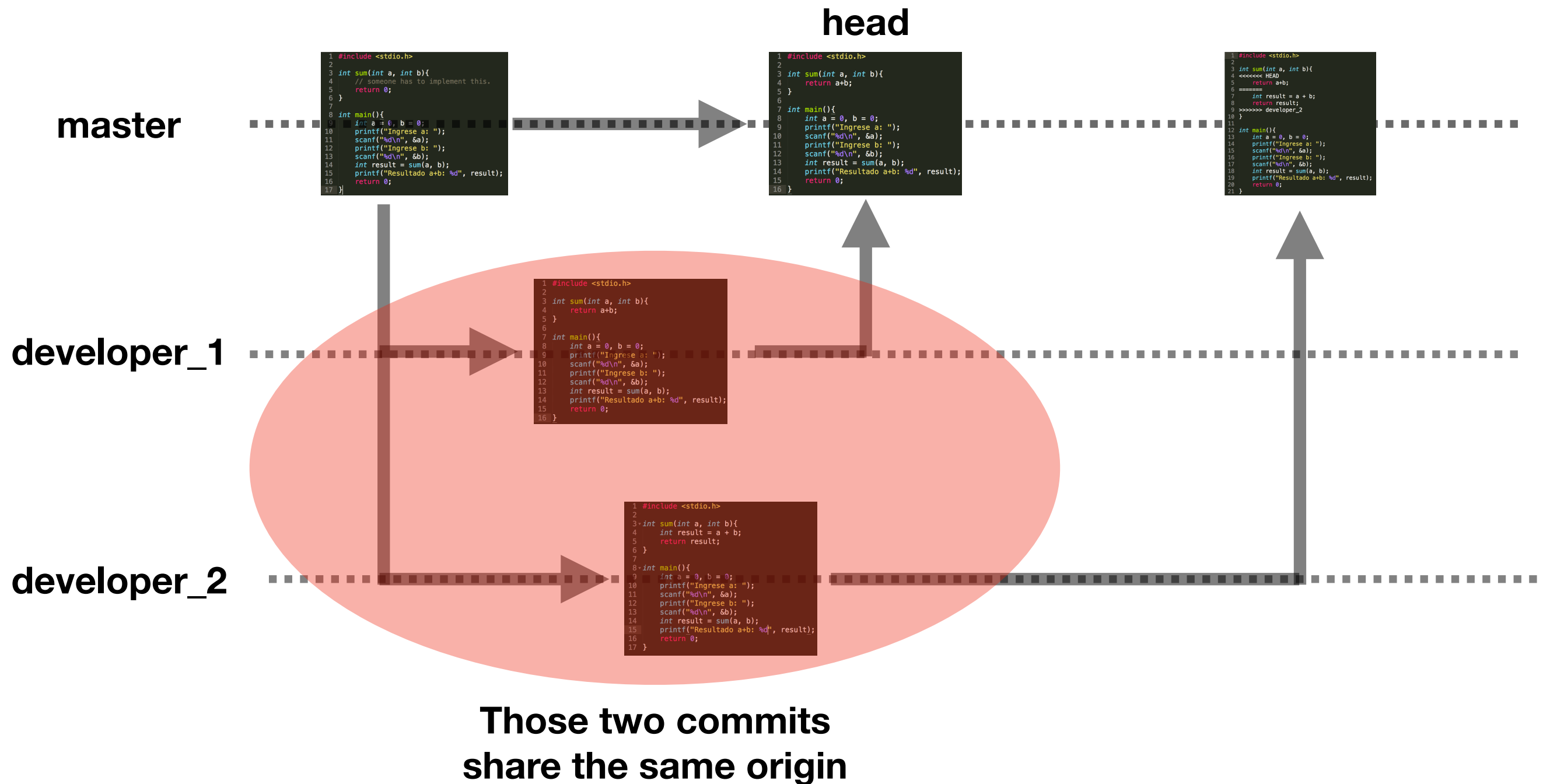


```c
#include <stdio.h>

int sum(int a, int b){
    return a+b;
}

int main(){
    int a = 0, b = 0;
    printf("Ingrese a: ");
    scanf("%d\n", &a);
    printf("Ingrese b: ");
    scanf("%d\n", &b);
    int result = sum(a, b);
    printf("Resultado a+b: %d", result);
    return 0;
}
```
developer_1

```c
#include <stdio.h>

int sum(int a, int b){
    // someone has to implement this.
    return 0;
}

int main(){
    int a = 0, b = 0;
    printf("Ingrese a: ");
    scanf("%d\n", &a);
    printf("Ingrese b: ");
    scanf("%d\n", &b);
    int result = sum(a, b);
    printf("Resultado a+b: %d", result);
    return 0;
}
```
master

```c
#include <stdio.h>

int sum(int a, int b){
    int result = a + b;
    return result;
}

int main(){
    int a = 0, b = 0;
    printf("Ingrese a: ");
    scanf("%d\n", &a);
    printf("Ingrese b: ");
    scanf("%d\n", &b);
    int result = sum(a, b);
    printf("Resultado a+b: %d", result);
    return 0;
}
```
developer_2

Managing Conflicts

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      return a+b;
5  }
6
7  int main(){
8      int a = 0, b = 0;
9      printf("Ingrese a: ");
10     scanf("%d\n", &a);
11     printf("Ingrese b: ");
```

```
kuky_nekoi@waifu  ~/Downloads/test  ⎇ master  git merge developer_1
Updating f9bf74e..67433d4
Fast-forward
 main.c | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
kuky_nekoi@waifu  ~/Downloads/test  ⎇ master  git merge developer_2
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then commit the result.
✗ kuky_nekoi@waifu  ~/Downloads/test  ⎇ master ●+ >M<
```

1. kuky_nekoi@waifu: ~/Downloads/test (zsh)

```c
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: ");
11     scanf("%d\n", &a);
12     printf("Ingrese b: ");
13     scanf("%d\n", &b);
14     int result = sum(a, b);
15     printf("Resultado a+b: %d", result);
16     return 0;
17 }
```

developer_2

```c
#include <stdio.h>

int sum(int a, int b){
<<<<<<< HEAD
    return a+b;
=======
    int result = a + b;
    return result;
>>>>>>> developer_2
}

int main(){
    int a = 0, b = 0;
    printf("Ingrese a: ");
    scanf("%d\n", &a);
    printf("Ingrese b: ");
    scanf("%d\n", &b);
    int result = sum(a, b);
    printf("Resultado a+b: %d", result);
    return 0;
}
```

kuky_nek
Updating
Fast-forw
 main.c |
 1 file c
 kuky_nek
Auto-merg
CONFLICT
Automatic
 ✗ kuky_n

developer_2

Managing Conflicts

developer_1

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      return a+b;
5  }
6
7  int main(){
8      int a = 0, b = 0;
9      printf("Ingrese a: ");
10     scanf("%d\n", &a);
11     printf("Ingrese b: ");
12     scanf("%d\n", &b);
13     int result = sum(a, b);
14     printf("Resultado a+b: %d", result);
15     return 0;
16  }
```

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      // someone has to implement this.
5      return 0;
6  }
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: ");
11     scanf("%d\n", &a);
12     printf("Ingrese b: ");
13     scanf("%d\n", &b);
14     int result = sum(a, b);
15     printf("Resultado a+b: %d", result);
16     return 0;
17  }
```

master

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      int result = a + b;
5      return result;
6  }
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: ");
11     scanf("%d\n", &a);
12     printf("Ingrese b: ");
13     scanf("%d\n", &b);
14     int result = sum(a, b);
15     printf("Resultado a+b: %d", result);
16     return 0;
17  }
```

developer_2

```
1. kuky_nekoi@waifu: ~/Downloads/test (zsh)
kuky_nekoi@waifu  ~/Downloads/test  master  git merge developer_1
Updating f9bf74e..67433d4
Fast-forward
main.c | 3 +--
1 file changed, 1 insertion(+), 2 deletions(-)
kuky_nekoi@waifu  ~/Downloads/test  master  git merge developer_2
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then commit the result.
kuky_nekoi@waifu  ~/Downloads/test  master  >M<
```

```c
1  #include <stdio.h>
2
3  int sum(int a, int b){
4  <<<<<<< HEAD
5      return a+b;
6  =======
7      int result = a + b;
8      return result;
9  >>>>>>> developer_2
10  }
11
12  int main(){
13     int a = 0, b = 0;
14     printf("Ingrese a: ");
15     scanf("%d\n", &a);
16     printf("Ingrese b: ");
17     scanf("%d\n", &b);
18     int result = sum(a, b);
19     printf("Resultado a+b: %d", result);
20     return 0;
21  }
```

master* (merging)

Managing Conflicts

Managing Conflicts

head

```
1   #include <stdio.h>
2
3   int sum(int a, int b){
4       // someone has to implement this.
5       return 0;
6   }
7
8   int main(){
9       int a = 0, b = 0;
10      printf("Ingrese a: ");
11      scanf("%d\n", &a);
12      printf("Ingrese b: ");
13      scanf("%d\n", &b);
14      int result = sum(a, b);
15      printf("Resultado a+b: %d", result);
16      return 0;
17  }
```

master

Managing Conflicts

head

```
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      // someone has to implement this.
5      return 0;
6  }
7
8  int main(){
9      int a = 0, b = 0;
10     printf("Ingrese a: ");
11     scanf("%d\n", &a);
12     printf("Ingrese b: ");
13     scanf("%d\n", &b);
14     int result = sum(a, b);
15     printf("Resultado a+b: %d", result);
16     return 0;
17 }
```

master

developer_1

```
1  #include <stdio.h>
2
3  int sum(int a, int b){
4      return a+b;
5  }
6
7  int main(){
8      int a = 0, b = 0;
9      printf("Ingrese a: ");
10     scanf("%d\n", &a);
11     printf("Ingrese b: ");
12     scanf("%d\n", &b);
13     int result = sum(a, b);
14     printf("Resultado a+b: %d", result);
15     return 0;
16 }
```

Managing Conflicts

Managing Conflicts

Managing Conflicts

head

Managing Conflicts

head

master

developer_1

developer_2

Those two commits
share the same origin

Pull request

Pull request

Pull request

Pull request

Forking repos

- Forks serve as a way to "remotely clone a repository".

- As they are "clones" of the original repo, users can also initiate "pull requests" from the forked repo to the original one.

- There is also a branching model for users who prefer forks to branches: **Forking Workflow.**
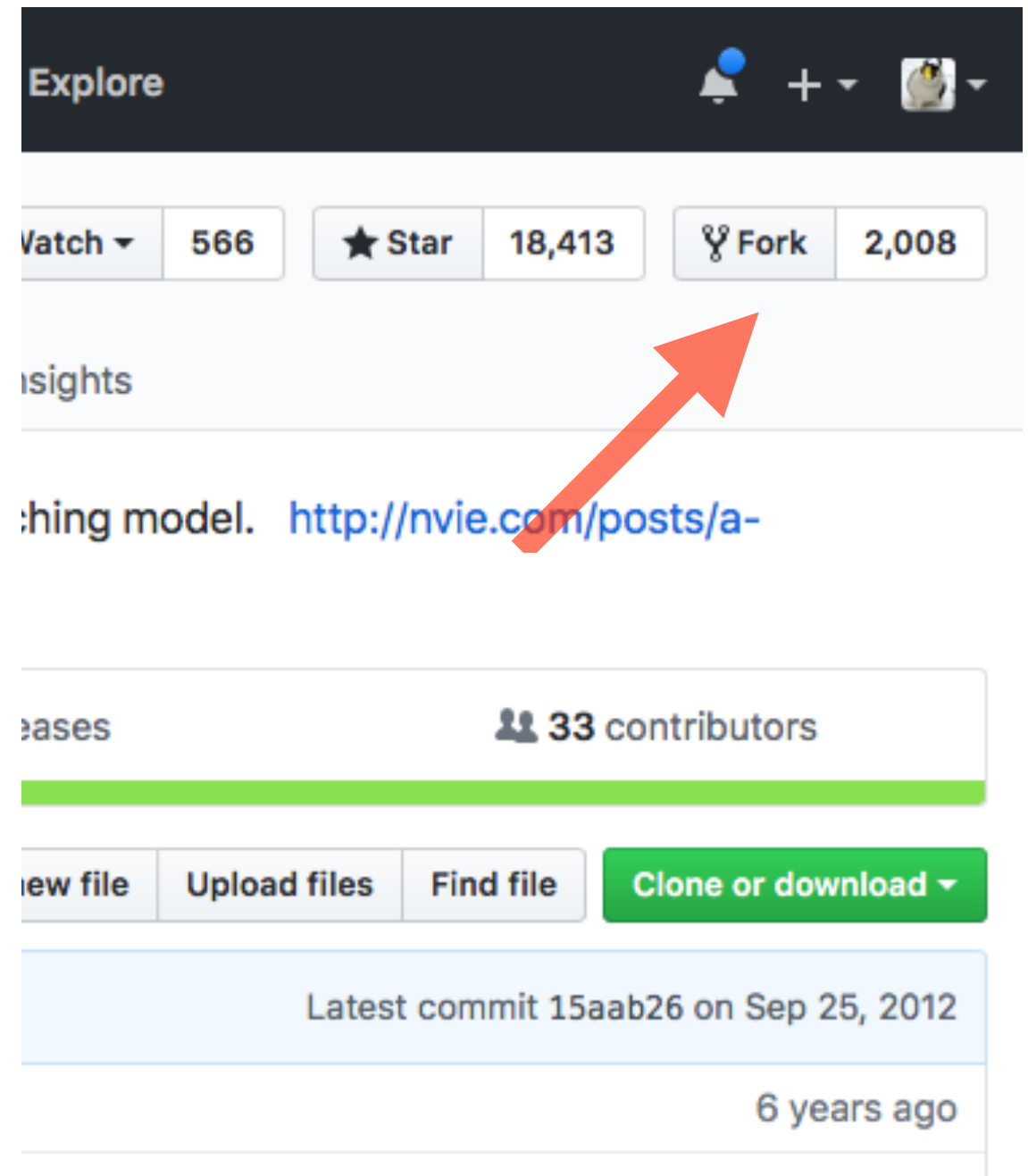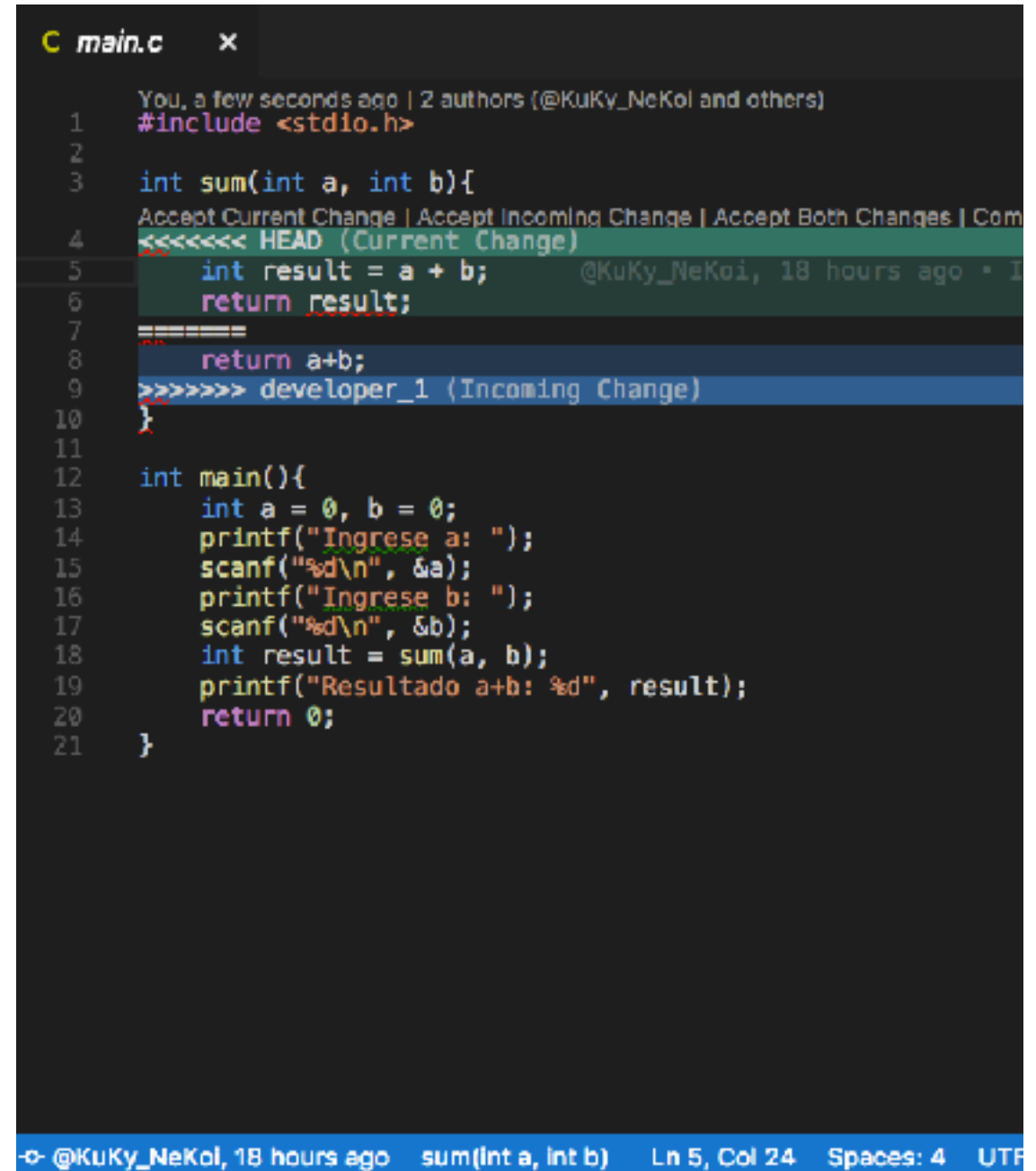
GIT Blame

- GIT also allows to track which person changed which line at a given moment (or to summarize all the changes).

- git-blame rarely gives useful results with this command, so git-log is prefered instead.

Useful tools

- **Visual Studio Code**: Has native integration with git tools as well as many useful plugins like git-lens which allow seamless integration with many git tools (like blame). It also identifies merge operations on the way allowing quick conflict resolution.
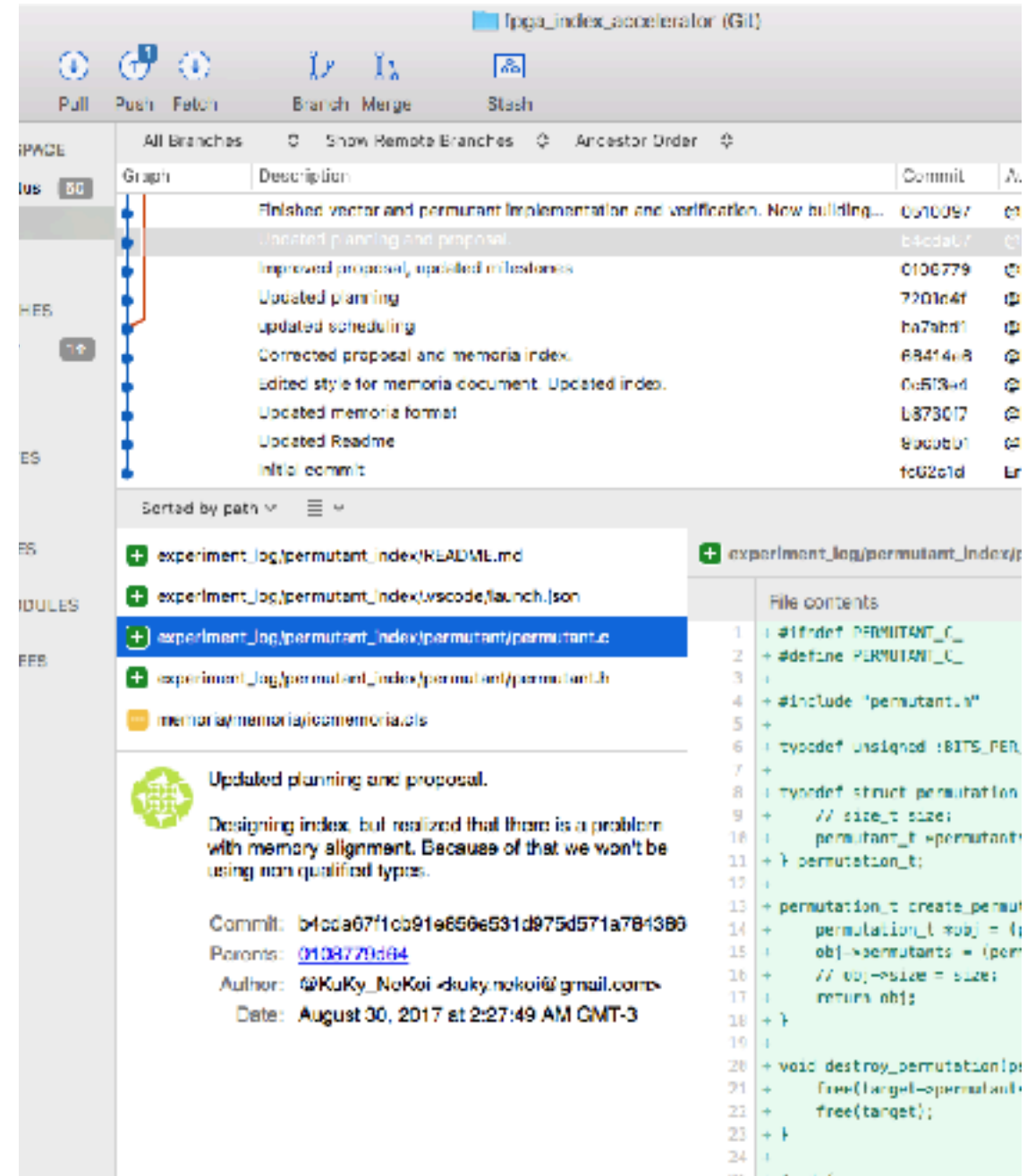
- Free

- Opensource

Useful tools

- **SourceTree:** Graphical frontend for GIT. Offers support for the entire workflow process, graphical inspectors and support for external diff tools.

- Free

- Propietary

Useful tools

- **BFG:** The BFG is a simpler, faster alternative to `git-filter-branch` for cleansing bad data out of your Git repository history

- In simple, it helps you to remove data from the entire git history, like sensitive data (passwords?) or big files (usually trash).

- Free and opensource.

BFG Repo-Cleaner

Removes large or troublesome blobs like git-filter-branch does, but faster. And written in Scala

`$ bfg --strip-blobs-bigger-than 100M --replace-text banned.txt repo.git`

an alternative to git-filter-branch

The BFG is a simpler, faster alternative to `git-filter-branch` for cleansing bad data out o repository history:

- Removing **Crazy Big Files**
- Removing **Passwords, Credentials & other Private data**

The `git-filter-branch` command is enormously powerful and can do things that the BFG the BFG is *much* better for the tasks above, because:

- Faster : 10 - 720x faster
- Simpler : The BFG isn't particularly clever, but *is* focused on making the above tasks e
- Beautiful : If you need to, you can use the beautiful Scala language to customise the B has got to be better than Bash scripting at least some of the time.

Exercise

Let's build a small console LED Display simulator. This display has to show a certain patter using a dot-like display and has to scroll smoothly on the screen.

At high level, the features to be developed can be summarized as follows:

- Scroll a sequence of characters.
- Generate the character sequence to scroll using a text.
- Retrieve terminal dimensions on the fly.
- Read and write files for sequences.
- Load and run sequences on the program.

Steps to follow:
- Create an account on https://github.com
- Form teams with 4-5 people each.
- Decide which feature your team will work on. Define which one will be the gitmaster of your team.
- Remember to *ask* your gitmaster to integrate your code using *pull requests*. Do not attempt to overwrite others work.

https://github.com/KukyNekoi/push-n-run

thx :3