

GDG Quito · Git Demo

Git, GitHub y GitFlow

First steps to version, collaborate and ship *without fear*.



Hi, I'm

Sebastian Alvarez

Fullstack Software Engineer
- **GDG Quito Organizer**

I build backends, cloud infrastructure and web products.
Git is the tool I use every day, on every project.

● GDG Quito

● Quito, Ecuador

@listerineh

Agenda

- **01** Introduction to Git
- **02** Introduction to GitHub
- **03** GitFlow: a branching model
- **04** Everyday Git commands
- **05** Demo: from `git init` to a pull request

01

Introduction to Git

What problem it solves and where it comes from.

How do five people work on the same code **without stepping on each other?**

The problem before Git.

Without version control vs. with Git

Without version control

- ZIP copies sent by email
- `project_final_v3_for_real.zip`
- Nobody knows which version is the right one
- One mistake and work is lost

With Git

- Full history of every change
- Branches to work in parallel
- Merge changes without copying them by hand
- Go back in seconds

A bit of history

- **2005**: the Linux kernel loses free access to *BitKeeper*, its version control system
- **Linus Torvalds** decides to write his own: Git
- Goals: **fast, distributed** and with guaranteed integrity
- Every commit is identified by a *hash* of its content: if anything changes, you notice
- Today it is the de facto standard in the industry

Git in numbers

2,005

year it was born

93%

of developers use Git

100M+

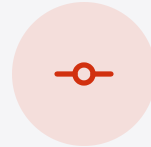
developers on GitHub

Key concepts



Repository

The project plus its whole history.



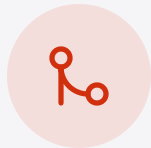
Commit

A snapshot of changes, with an author and a message.



Branch

An independent line of work.



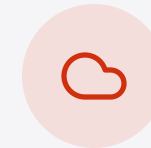
Merge

Joining the work of two branches.



Staging

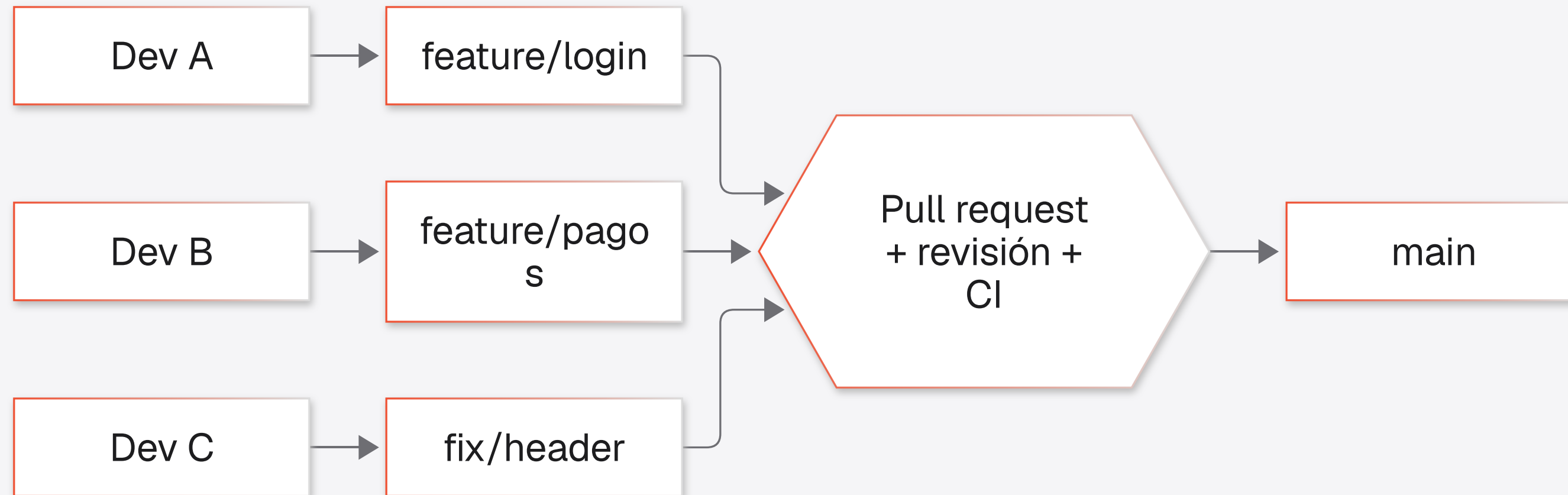
What goes into the next commit.



Remote

A shared copy, on GitHub for example.

How work flows in a team



Everyone works on their own branch; nothing reaches `main` without a review.

02

Introduction to GitHub

Git in the cloud, with tools to collaborate.

Git is not GitHub

Git

- A version control tool
- Runs on your machine, no internet needed
- Open source, created in 2005

GitHub

- A service that hosts Git repositories
- Adds collaboration: PRs, issues, Actions
- Alternatives: GitLab, Bitbucket

What GitHub adds



Accessibility

A visual platform with a friendly interface.



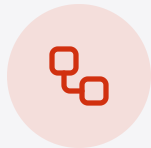
Tools

Projects, Issues and Wikis.



Collaboration

Pull requests, reviews and forks.



Automation

GitHub Actions: tests and deploys on every push.



Protection

Branch rules and required reviews.



Publishing

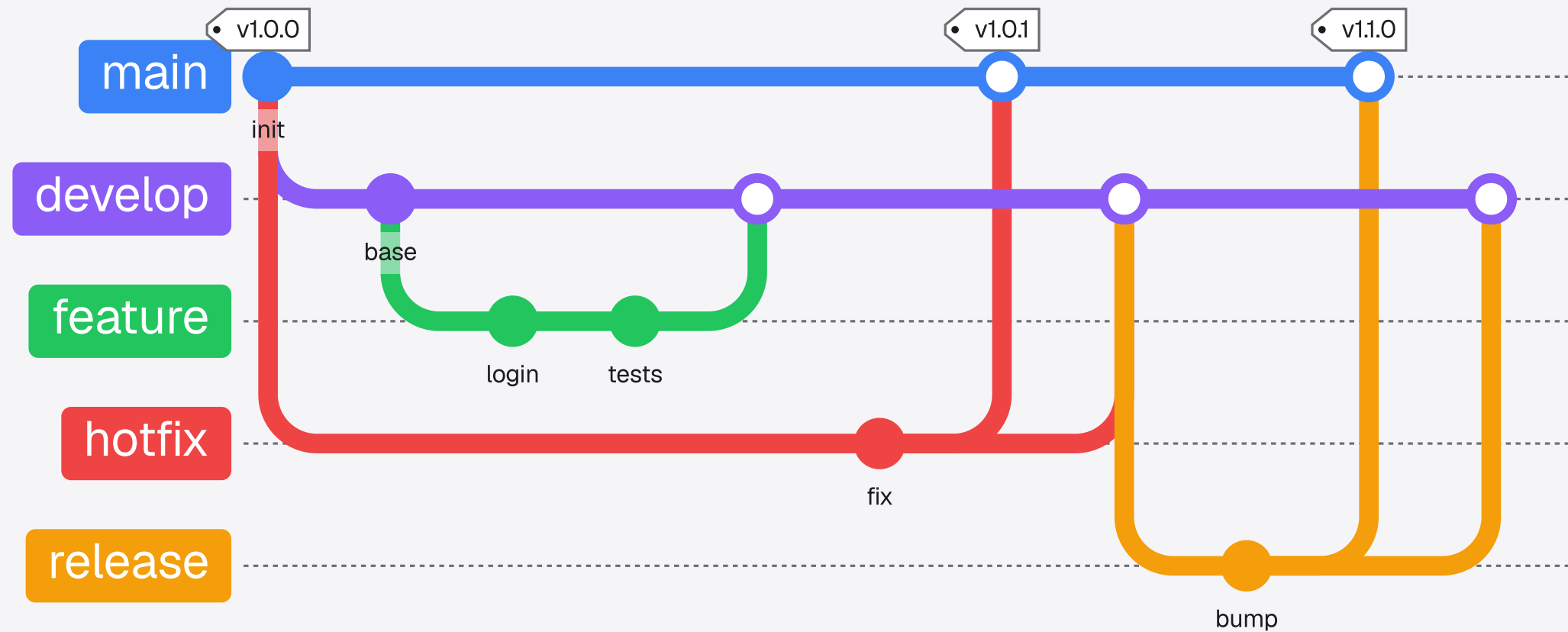
GitHub Pages for static sites.

03

GitFlow

A branching model for teams that ship versions.

GitFlow at a glance



`main` always mirrors production; everything else is integrated into `develop`.

The GitFlow branches

- `main` : what runs in production, tagged by version
- `develop` : where the next release comes together
- `feature/*` : one feature; branches off develop and goes back to develop
- `release/*` : stabilizes a version before it ships
- `hotfix/*` : fixes production from main and goes back to main and develop
- With continuous deployment, **GitHub Flow** or *trunk-based* development is usually simpler

04

Git commands

What you will use every day.

Setup

BASH

```
# Once per machine
git config --global user.name "Your Name"
git config --global user.email "you@email.com"

# Only for this repository
git config --local user.name "Your Name"
git config --local user.email "you@work.com"
```

`--local` wins over `--global`: handy to keep work and personal projects apart.

Starting a repository

BASH

```
# A new project
```

```
git init
```

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

```
# A project that already exists
```

```
git clone <url>
```

`git clone` brings the full history and sets up the `origin` remote.

The three areas of Git



Understanding this path explains almost every command.

Branches

BASH

```
git branch          # list branches
git branch <branch>  # create a branch
git switch <branch>   # switch branches
git switch -c <branch> # create and switch
```

The classic way, still works

```
git checkout <branch>
git checkout -b <branch>
```

`git switch` (Git 2.23+) splits "changing branches" from what `checkout` did with files.

Staging and commits

BASH

```
git status          # what changed
git diff            # unstaged changes
git add <file>      # move to staging
git diff --staged   # what goes into the commit
git restore --staged <file> # take it out of staging
git commit -m "feat: descriptive message"
```

`git restore --staged` is the modern form of `git reset <file>`.

Syncing

BASH

```
git fetch          # get changes without applying them
git merge <branch> # merge another branch into this one
git rebase <branch> # replay your commits on top
git pull           # fetch + merge
git push -u origin <branch> # publish your branch
```

`git pull --rebase` keeps history linear when you work on the same branch.

Good practices

- **Small** commits with clear messages (`feat:` , `fix:` , `docs:`)
- One branch per change, with a name that explains it
- `pull` before you start and before you `push`
- Never rewrite shared history; if you must, use `--force-with-lease`
- A `.gitignore` from day one: no secrets, no builds
- Every change goes in through a **pull request**, with a review

05

Demo

From `git init` to a pull request.

More questions? Let's connect

The slides stay at listerineh.dev/slides.

linkedin.com/in/listerineh

GDG Quito