Progressive Flow Whitepaper

Introduction

What is Progressive Flow?

Progressive Flow is a paradigm applied to Git, parallel to **Git Flow**, that was created to solve one of the latter's main problems: the bottleneck caused by release management.

Those who have used Git Flow on dynamic projects and in collaborative teams have probably found themselves having to manage multiple parallel features generated by customer requests. Initially, the process runs smoothly: tasks proceed, *features* are developed and closed, and the first completed change initiates the opening of a release. However, it is here that critical issues emerge.

Closed features are gradually added to the release awaiting customer approval. If a single feature takes longer than expected to be approved or, worse, requires complex rework, all subsequent changes get stuck in the release. This situation generates buildup, delays testing, and prevents the release of changes that would already be ready.

When approval finally comes, the problems do not end: after months, it becomes difficult to remember all the details of the changes, how they work, and which use cases to test. This scenario is surprisingly common and demonstrates how, in practice, the introduction of the end customer and its approval process can undermine the theoretical effectiveness of Git Flow.

Progressive Flow's solution
Progressive Flow is designed to overcome these limitations. By introducing the intermediate ***candidate*** branch, it eliminates the bottleneck and ensures a continuous flow, fully aligning with CI/CD principles. This approach allows the isolation of approved changes from those still in the works, ensuring rapid releases, accurate testing, and better management of ongoing activities.

Paradigm Overview

Progressive Flow is proposed as an improved alternative to Git Flow, designed to address its structural limitations. Its approach focuses on independently managing features, keeping them ready to be released at any time, without creating bottlenecks in the development and release process.

The paradigm introduces some key changes to the "lifecycle" of features, without disrupting the workflow familiar to developers already using Git Flow. This balance between innovation and continuity allows for easy and effective adoption.

The advantages of Progressive Flow

1. **Flexibility**
   Progressive Flow enables more frequent and independent releases, moving closer to the fundamental principles of Continuous Integration and Continuous Delivery (CI/CD). Features can be released without waiting for approval or completion of other activities.
2. **Familiarity**
   Built with strong inspiration from Git Flow, Progressive Flow maintains a similar workflow. For those already using Git Flow, the transition will be natural: the only significant difference is the more advanced and flexible feature management.
3. **Reduction of conflicts**

Thanks to the introduction of the intermediate candidate branch (discussed in more detail in later sections), Progressive Flow allows partial merges between candidate and release branches. This approach significantly reduces conflicts, simplifying integration work.

4. **Improved collaboration**
   Development teams can work in parallel without blockages or delays. Each feature progresses independently, preventing a pending change from blocking the entire process.
5. **Support for scalability**
   Progressive Flow is particularly well suited for complex or dynamic projects where customer demands and the need for frequent updates make a more rigid approach impractical.
6. **Traceability and control**
   The introduction of intermediate branches and structured workflows provides greater control and visibility over the entire development process, making it easier to track changes and identify issues.

Ultimately, Progressive Flow combines the familiarity of Git Flow with a more flexible and modern structure optimized for today's software development needs.

Differences with other paradigms:

Progressive Flow vs. Git Flow

- **Release Flexibility**: Git Flow requires all features to be completed and integrated before closing a release branch. Progressive Flow uses the intermediate candidate branch, allowing selective feature integration, reducing bottlenecks and speeding up the release of approved changes.
- **Complexity**: Both paradigms handle complex scenarios, but Progressive Flow simplifies the process while maintaining the integrity of the main (develop) branch and improving feature modularity.

Progressive Flow vs. GitHub Flow

- **Support for Multiple Environments**: GitHub Flow, being designed for simplicity, does not handle staging or pre-production environments. Progressive Flow integrates a full cycle that includes testing and approvals, improving code safety in critical environments.
- **Scalability**: While GitHub Flow is ideal for small teams and linear projects, Progressive Flow is better suited for larger teams or those with more complex workflows.

Progressive Flow vs. Trunk-Based Development

- **Focus on Stability**: Trunk-Based Development aims for rapid iterations with frequent commits directly to the main branch. Progressive Flow offers a balance, using candidate branches to stabilize code before merging changes.
- **Risk of Premature Releases**: Progressive Flow reduces the risk of integrating untested changes by using revision cycles in the candidate branch, which are not provided in Trunk-Based Development.

Technical details

The *Candidates*

As mentioned earlier, candidates represent a key concept of Progressive Flow and mark a distinct evolution from Git Flow features. Let us delve deeper into their role and how they work.

Candidates are branches that arise directly from the develop branch and sit somewhere between develop and release. This approach allows development to be managed more flexibly and avoids the bottleneck typical of traditional features.

Why are they called *Candidates?*

The name “candidate” comes from the fact that these branches represent functionality that is a *candidate* for release. A candidate remains open until it gets the necessary approval to be promoted to production. This allows the state of technical completion (finished and tested) to be separated from the state of final approval, avoiding blockages and delays in the pipeline.

How do the *Candidates work*?

Here are the main steps:

1. **Creation**: Each new *candidate* is born from develop and includes the necessary developments for the current feature.
2. **Develop**: All development, **including any rework**, is done within the candidate. This makes each feature independent of the others.
3. **Testing on Staging**: *candidate* changes can be integrated into the release branch, which deploys to the staging environment. Here they are tested in a production-like context.
4. **Approval**: Once testing is completed and approval is received from the customer or team, the candidate is ready for release.
5. **Closure and Merge**: When the candidate is closed, changes are merged to both develop and master, updating both branches with the most recent and stable state.

Advantages of Candidates

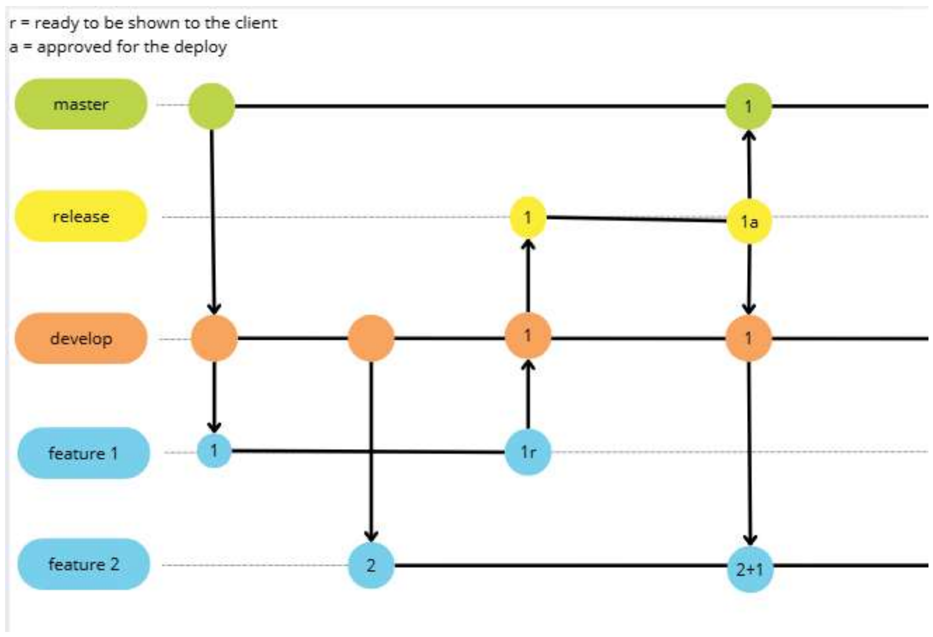
- **Feature Independence**: Each candidate is autonomous, reducing conflicts and dependencies between features in development.
- **Flexibility:** An approved feature can be released without waiting for the others.
- **Better Traceability**: The use of candidates allows the status of each feature to be clearly tracked, separating development, testing and approval.

Difference between Candidate and Feature

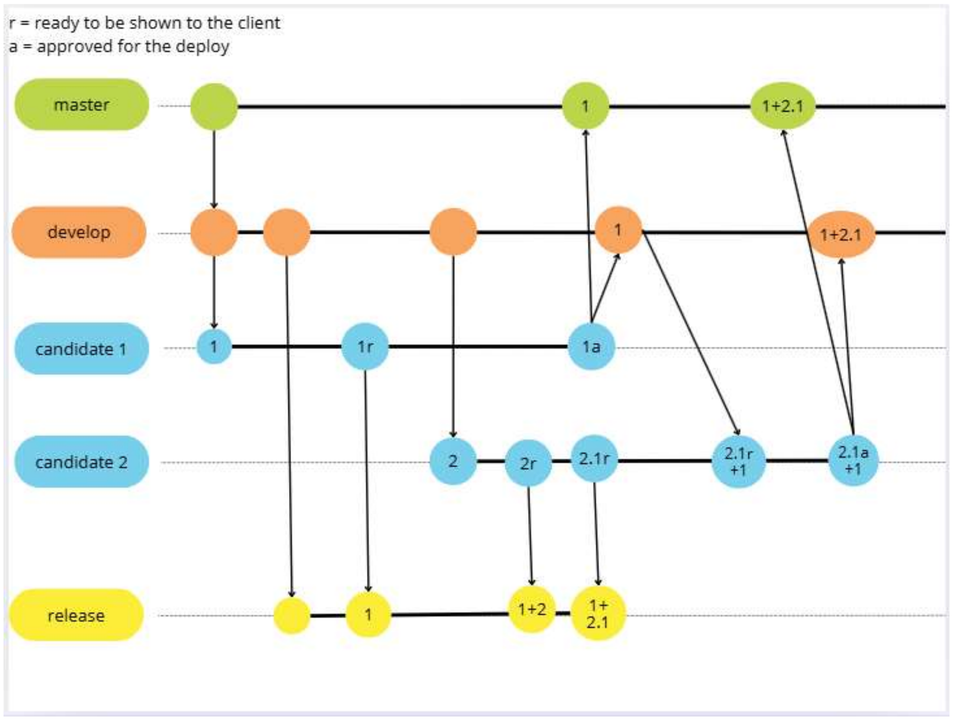
The main difference between *candidates* and *features* lies in their purpose and lifecycle management. Git Flow *features* are designed exclusively for feature development and are integrated directly into the develop branch upon completion. *Candidates*, on the other hand, add an intermediate layer: they not only represent development work, but remain open until final approval, separating the “technically ready” state from the “approved for release” state. This approach keeps the develop branch cleaner, avoiding the mixing of changes still subject to rework or unapproved, and allows for more granular and independent management of pipelined features.

Difference between Candidate and Release
Candidates are branches focused on the development and approval of individual features. The release branch, on the other hand, represents the aggregate state of production-ready candidates. Candidate

changes are temporarily integrated into the release branch for testing, but their final fate depends on individual approval.


r = ready to be shown to the client
a = approved for the deploy
master
release
develop
feature 1
feature 2
1
1
1a
1
1
1
1r
2
2+1

Git Flow Workflow



Progressive Flow Workflow

Explanation of Progressive Flow Diagram

The diagram illustrates the workflow of the **Progressive Flow** paradigm, highlighting how the major branches interact to optimize the development, review and release cycle.

Structure of the Branches:

1. **Master** (green): Represents the stable branch that contains the approved and released code in production.

2. **Develop** (orange): This is the main branch of development, where candidates are integrated once approved.
3. **Candidate** (light blue): These branches contain independent developments. Each candidate is autonomous and can be reviewed, tested and approved individually.
4. **Release** (yellow): Functions as staging to collect candidate branches ready for release.

Status indicators:

- r (ready): Indicates that a **candidate** is ready to be shown to the customer.
- a (approved): Reports that a **candidate** has passed testing and can be deployed.

Workflow:

1. **Creation of Candidates**:
   - Candidate branches arise from develop. For example, candidate 1 and candidate 2 represent two parallel developments born from the same develop.
2. **Transition to Release**:
   - Once a candidate is ready (1r, 2r), it is integrated into the release branch for staging tests.
3. **Approval and Deploy**:
   - Approved candidates (1a, 2a) are integrated into master for production release and propagated in develop to keep branches synchronized.
4. **Partial merge**:
   - The diagram shows how partial merges between candidates and release avoid conflicts and promote faster release cycles.

Highlighted Benefits:

- **Candidate Independence**: Each candidate is autonomous, reducing bottlenecks typical of other paradigms.
- **Flexibility in Releases**: The model allows for a smooth CI/CD cycle by separating development, staging, and production.
- **Optimized Collaboration**: Teams can work in parallel without interruption.

Practical implementation of Progressive Flow

Installation via GitHub API

To install Progressive Flow from the official repository, visit https://github.com/vin100proflow/progressive-flow/ .
Use the following command to automatically download the latest .deb package, renaming it as proflow_latest.deb:

```
curl -s https://api.github.com/repos/vin100proflow/progressive-flow/releases/latest \
| grep "browser_download_url" \
| grep ".deb" \
| cut -d '"' -f 4 \
| xargs wget -O proflow_latest.deb
```

Once the download is complete, run the following command to install the package:

sudo dpkg -i proflow_latest.deb

Note: This method requires manual updates. In case of new versions, it is necessary to remove the previous version (see Uninstallation section) and reinstall the updated version.

Manual installation from GitHub repository

If you have problems with the installation via API, you can download the package manually:

Visit the official repository: https://github.com/vin100proflow/progressive-flow/.

Go to the Releases section (in the navbar on the right).
Download the .deb file or one of the compressed packages.
Place the downloaded file in the desired directory (e.g., /home/username/) and run the following command:
sudo dpkg -i package_name_progressive_flow.deb

Note: This method also requires manual updates in case of new versions.

Installation via Launchpad.net PPA

To take advantage of automatic updates, it will be possible to add the official PPA of Progressive Flow (https://launchpad.net/~vin100proflow/+archive/ubuntu/progressive-flow) via Launchpad:
sudo add-apt-repository ppa:vin100proflow/progressive-flow
sudo apt update
sudo apt install progressive-flow

With this method, Progressive Flow will be updated automatically with each subsequent run of *sudo apt update && sudo apt upgrade.*

Disinstallazione di Progressive Flow

Uninstallation: installed from GitHub API

To remove Progressive Flow installed via .deb package, run the following commands:

sudo dpkg -r proflow
sudo dpkg --purge proflow

Uninstallation: installed manually from GitHub Repository

To remove Progressive Flow installed via .deb package downloaded from the GitHub Repository, run the following commands:

sudo dpkg -r proflow
sudo dpkg --purge proflow

Uninstallation: installed from PPA

To uninstall Progressive Flow and remove the PPA:

```
sudo apt remove progressive-flow
sudo add-apt-repository --remove ppa:vin100proflow/progressive-flow
```

Conclusion

Progressive Flow represents a step forward in software development flow management, addressing the limitations of established paradigms such as Git Flow. With the introduction of candidate branches, this model allows greater flexibility, supports faster release, and reduces bottlenecks caused by complex approval processes. The benefits for development teams are tangible: fewer conflicts, more independence between features, and smoother collaboration.

Looking ahead, Progressive Flow aims to become even more adaptable to the needs of diverse development teams. One potential enhancement is the ability to fully customize branch names based on project-specific workflows, allowing teams to align the branching model with their unique processes and nomenclature. Additionally, integrating with CI/CD pipelines remains a priority, enabling smoother transitions between development stages and more efficient automation of quality checks and deployments.

The change to Progressive Flow is simple, especially for those already using Git Flow, and the benefits it offers are immediate. We invite development teams and technical managers to experiment with Progressive Flow in their projects. Join us in building a more agile and dynamic future for software development. Visit the official repository, adopt the paradigm, and share your feedback to continue to improve together!

If you have any questions or feedback, please feel free to contact me at vincent.legnani.biz@gmail.com.

I sincerely thank you for taking the time to read this whitepaper. I hope that Progressive Flow can truly simplify and improve your work experience.

Kind regards,
Vincent Legnani