abapGit Flow

Lars Hvam, Heliconia Labs, April 2024

Home	https://github.com/heliconialabs/abapgit-flow
License	© Heliconia Labs, 2023
Build	2024-04-21 10:10:03 UTC

1. Introduction

abapGit flow is a end-to-end development and deployment process for SAP ABAP systems. It allows for a modern development process, with a focus on quality and speed, where features can be deployed independently in any order.

1.1. Key Features

Quality built-in, with automated testing and code reviews

Leverage the power of automated testing, and code reviews, to ensure quality is built-in from the start. This allows for a fast development process, without sacrificing quality.

Independent changes, in any order to production

Using classic CTS for deployment, changes can be moved to production when they are ready, without having to wait for other changes to be ready.

Integrates with existing change management tools/process

The development process can be adjusted to fit the needs of the customer, and the existing tools and processes. None of the example tools listed below are mandatory, and can be replaced with other tools, or removed completely.

Examples: Tosca, Worksoft, Service Now, Jira, Basis Technologies, Rev-Trac, SAP ChaRM, Cloud ALM

Central or de-central development

Traditionally, ABAP development is done in a central development system, where all developers work on the same code base. This can be a bottleneck, and slow down the development process. With abapGit flow, the development can be done in a de-central way, where each developer works on their own system and pushes changes to git.

However, de-central development is not a requirement in abapGit flow

Flexible repository setup/split

The repository setup can be adjusted along the way, to fit the needs of the development team. The repository can be split into multiple repositories, or merged into one, without affecting the development process.

Classic on-prem development, 7.02 and up

Works on most SAP ABAP on-prem systems, upgrade tooling at any time, no need to wait for SAP to release new features. No need to upgrade to newest S/4 HANA to get the latest development tools.

Keeping the core moving

Keep the core moving towards a better core, improving step by step, without big bang implementations.

Safely refactor the core, without breaking existing functionality.

Editor independent

Use any editor the developer is comfortable with, including:

- SE24/SE80
- Eclipse ADT
- Visual Studio Code, standalone ABAP development
- GitHub web editor
- github.dev

AI ready

Get ready for AI powered development, by moving the ABAP code to git opens up for new possibilities, such as:

- GitHub Copilot
- GitHub Copilot X
- GitHub Copilot Workspace

1.2. Key Design Decisions

The default setup described in this document includes the following design decisions:

1. Changes can be moved independently to production, in any order using classic CTS
2. SAP ABAP system initiates all communication, no Cloud → On-prem connections
3. GitHub Cloud is the central repository
4. No fully automated deployment to production after the development is done, each feature is manually tested or requires manual approval before moving to production

However, the design can be adjusted to fit the needs of the customer, and the existing tools and processes.

For product/add-on development which is typically released as a whole, instead of per feature, see whitepaper ABAP Product Development.

1.3. Nuve Platform

Nuve Platform offers decentralized development systems and automated unit testing, streamlining continuous integration processes.

The platform features a self-service portal that enables developers to manage the lifecycle of SAP development instances easily, without requiring Basis expertise. Instances are provisioned in minutes, facilitating a cost-effective and practical decentralized development setup.

Nuve's GitHub integration automates the execution of unit tests within your workflow. This ensures valid results by running tests in real systems.

Nuve Platform

1.4. abaplint.app

abaplint.app provides fast configurable static analysis of ABAP code, with code insights and statistics.

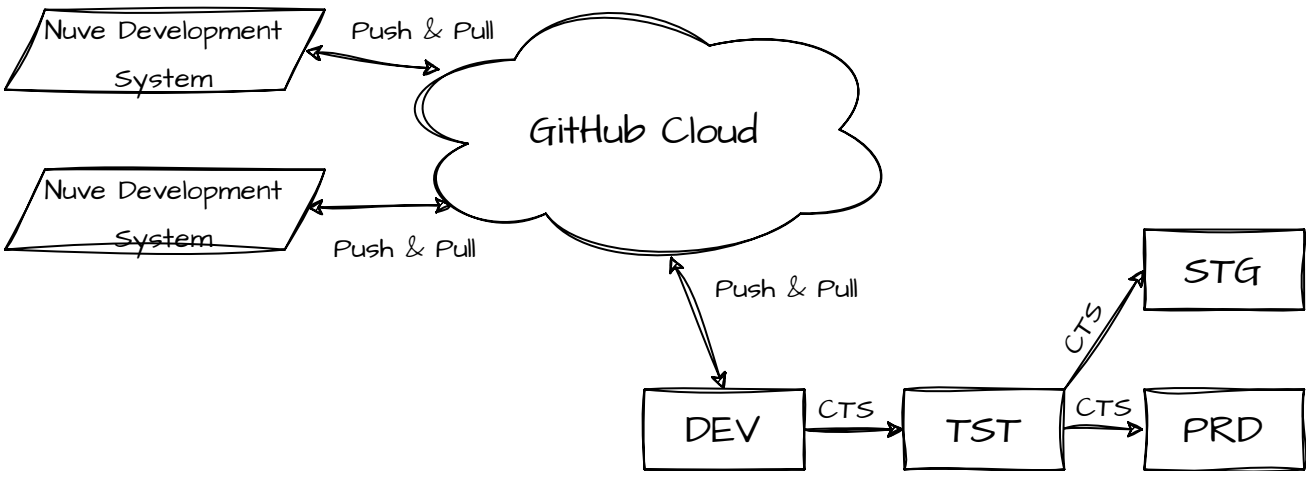
Configuration is versioned and changed like code, any developer can suggest changes and changes are reviewed before being merged.

All developers have access to insights and statistics, without additional infrastructure setup or maintenance.

abaplint.app

1.5. Example landscape

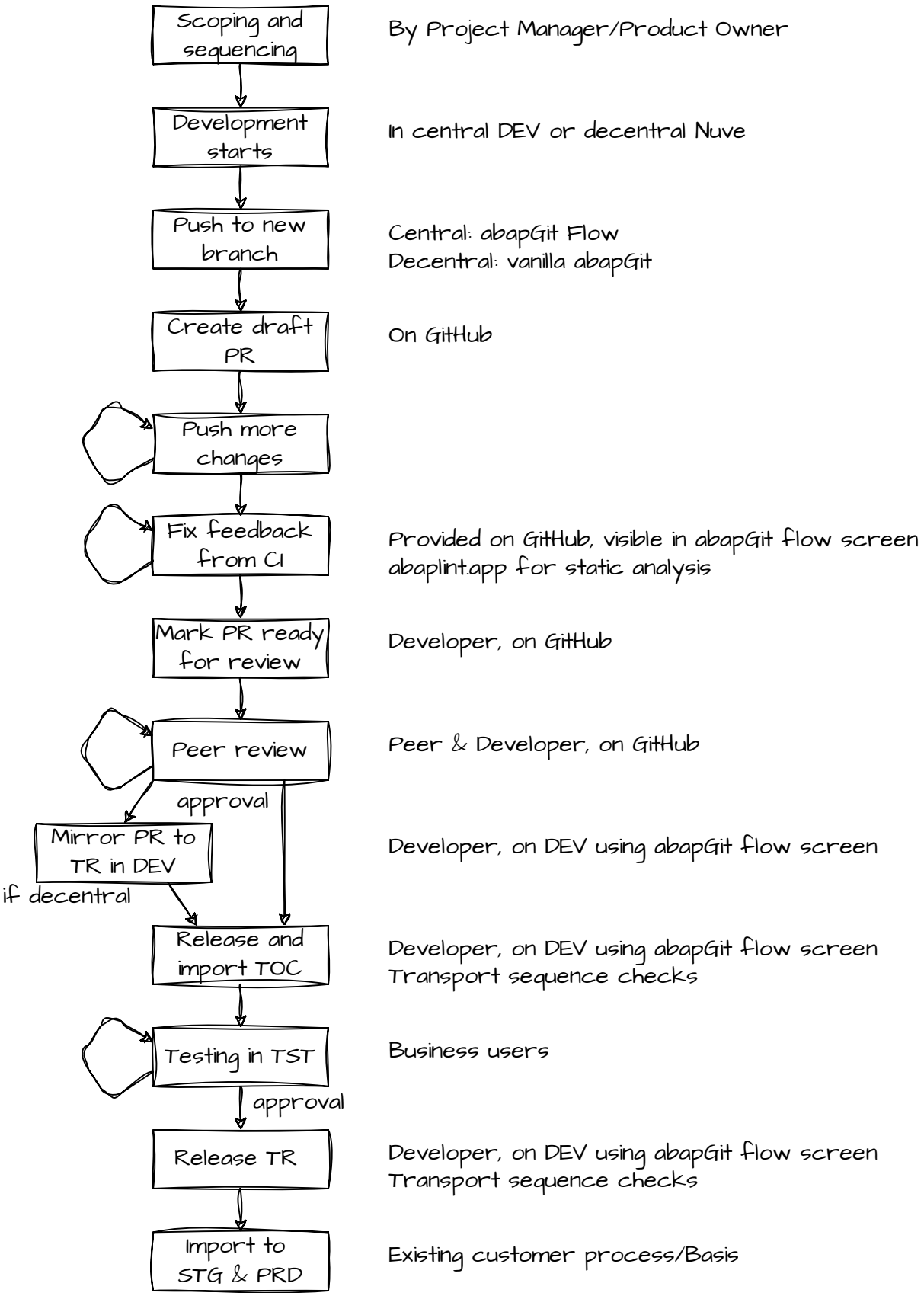
For the setup and examples in this document, the following landscape is used:



- DEV is the central development system, managed by the customer infrastructure team
- TST is used for testing and sign-off by business
- STG staging system to do final validation before moving to production
- PRD production environment

2. End to End Process

There are many steps in the end-to-end process, however, users are guided through the steps via tooling and automation.



3. Development

Development is done centrally in DEV and/or de-centrally in the Nuve systems.

The package structure is split into sizeable packages, corresponding to the areas of responsibility inside the system. Each area has one or more repositories on GitHub.

Before starting the development, the tasks are scoped to a reasonable size. Then tasks are assigned to developers, and the developers create a branch for the task and start working on it. The branch is pushed to GitHub and a pull request is created.

3.1. Technical Quality Assurance

Feedback regarding static analysis and unit tests is given by the continuous integration system. The developer can see the feedback in the pull request, and fix any issues.

Fast running feedback is prioritized first, if they are successful, longer more precise tests are run.

When the functionality is ready, the developer sets the pull request to "Ready for review", and collaborates with the peer reviewer until the pull request is approved.

For de-central development, approved pull requests are mirrored into central DEV.

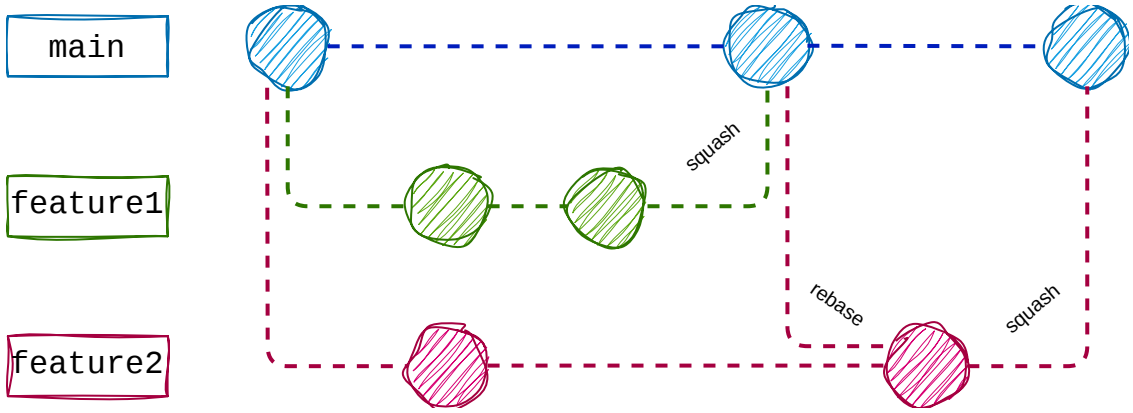
After moving a transport of copies to the TST system, the development is ready for functional testing, and sign-off by business.

3.2. Branching examples

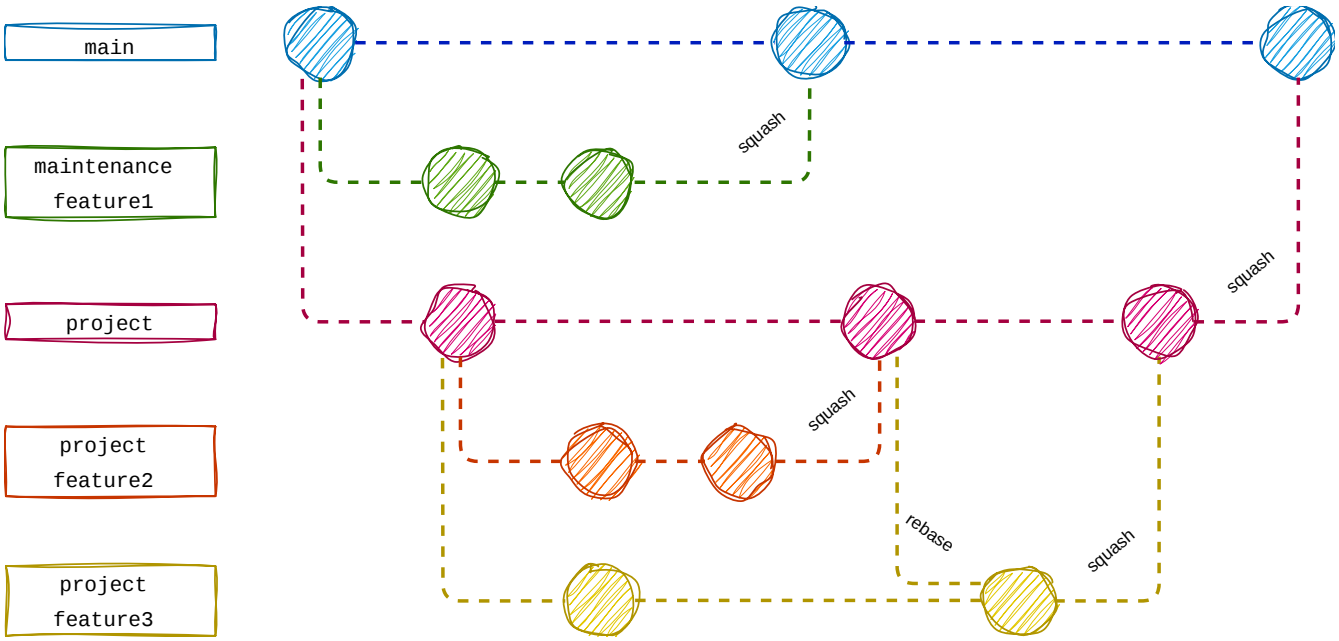
Developers create feature branches based on `main`, and create a draft pull request when the first commits are pushed. Push often, both to get a backup, increase visibility, and get the feedback from continuous integration.

If the branch is not up to date, the developer can press the "Update branch" button to get the latest changes from `main`.

When the transport is moved to production, the pull request is squashed into `main` and the branch is deleted.



It is also possible to have a more long lived branch, here a `project` branch is created from `main`, and the features are developed from that. But do try to avoid long lived branches, keep moving changes to production if possible, or make it possible via eg. feature flags.



3.3. Development guidelines recommendations

Typically customer developments consists of multiple independent objects. These objects can be organized into small packages. Example: one package with a transaction, report, message class, and a global class for business logic.

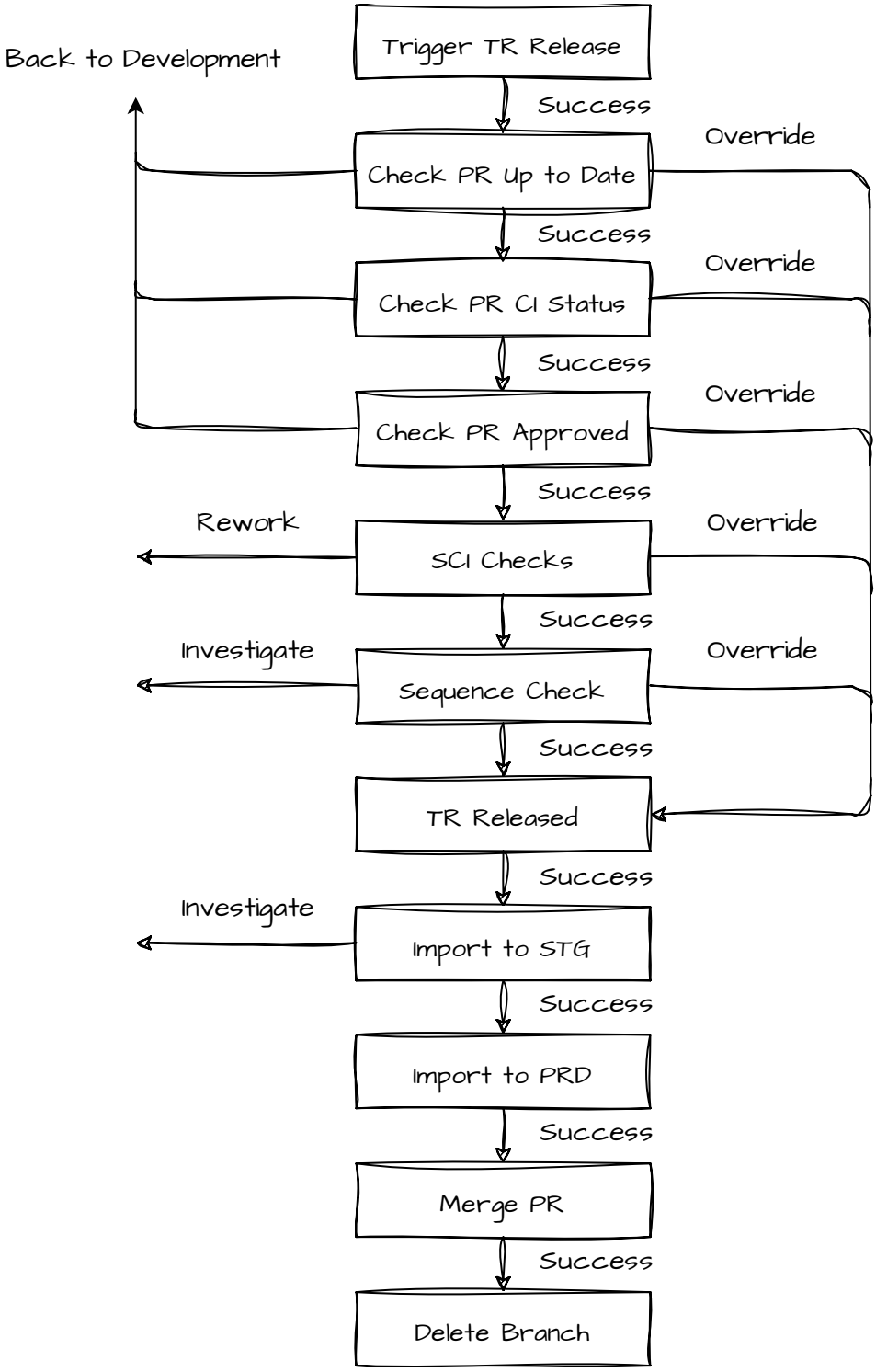
Reduce reuse of message classes, create many, eg. one for each global class. This will reduce the risk of conflicts in the message class XML file. abaplint rule `easy_to_find_messages` can be used to enforce this, while making it easier to find the exact place a message is issued.

4. Deployment

When development is done, unit tested and reviewed, the next step is to deploy the changes to production. This is done via the classic Change and Transport System (CTS).

4.1. Deployment Process Overview

Most steps of the deployment process are automated. A failure at a late stage indicates a need for earlier process improvements to detect such issues.



4.2. System SCI checks

Static analysis feedback is provided to the developer immediately on the first push.

In GitHub, the code is isolated from the system's structure. SAP Code Inspector (SCI) can be used to perform additional checks, especially for assessing aspects like package structure, which GitHub alone does not handle.

4.3. Sequencing

Development projects often have dependencies that may not yet be available in the production environment. This can cause issues when moving the transport to production.

This can be discovered already in the pull request during development, eg. via the abaplint.app cross check or parallel changes feature.

To help checking the SAP standard transaction `/SDF/TRCHECK` can be used.

And for final verification, the transport is imported into the `STG` system, just previously to importing into the `PRD` system.

4.4. Rollback

Transports can be rolled back by creating an additional transport with the original contents.