abapGit: State of the Nation 2019 Q2

Table of Contents

Abstract

abapGit has been around for some years, this document describes where it has been and where it is possibly going. The development of abapGit is community driven, so no promises are made with this document, nor does it necessarily reflect the opinions of all authors and their employers.

Feedback and fixes for this document welcome at https://github.com/abapGit/sotn_2019_q2

Revision History

Date	Description
foo	todo
bar	todo

For full history see https://github.com/abapGit/sotn_2019_q2

1. abapGit

abapGit is a open source git client for ABAP written in ABAP. The project was started around 5 years ago, it has grown from being a small hobby project to something widely used in the ABAP community.

http://abapgit.org

1.1. Features

From the beginning there have been 4 main design goals:

- Easy installation
- Easy upgrade
- Small system footprint
- Code readable in git repository

Which has enabled all ABAP developers to easily start getting into the world of git. The git client can be installed on ABAP systems higher than v702 by anyone with a developer key.

The initial version supported only basic git commands with a very basic user interface, but over the years more and more features have been added, so that it now supports the most commonly used git workflows:

```
git config usernames
git checkout
git change remote
git create branch
git change branch
git delete branch
git status
git diff
git clone
git pull
git push
git log
git commit
git patch
git reset
git add
git rm
git merge
git create tag
git change tag
git delete tag
ignore
uninstall
```

Along with serializers for more than 80 different object types. Customizing is supported via Business Configuration Sets

1.2. Support

All development and support for abapGit happens via normal Open Source workflows, anyone can follow or help with the development, and anyone can suggest features or report bugs.

In order to optimize abapGit and add more features the code is updated very often, and as the code is delivered as custom code, anyone with a developer key can update it to the latest version in the development system.

In case professional support is required, then multiple customers and partners are using abapGit and has experience with the tool.

1.3. Git Hosting

abapGit implements the raw git protocol from scratch transported via HTTP connections. This makes sure it can be used with almost any git host. Connecting to modern git hosting services enables the ABAP developer to take advantage of the tools offered by the host.

Each user can decide where and how to put the ABAP code, inside the firewalls or in the cloud. There are many different services offered in the market:

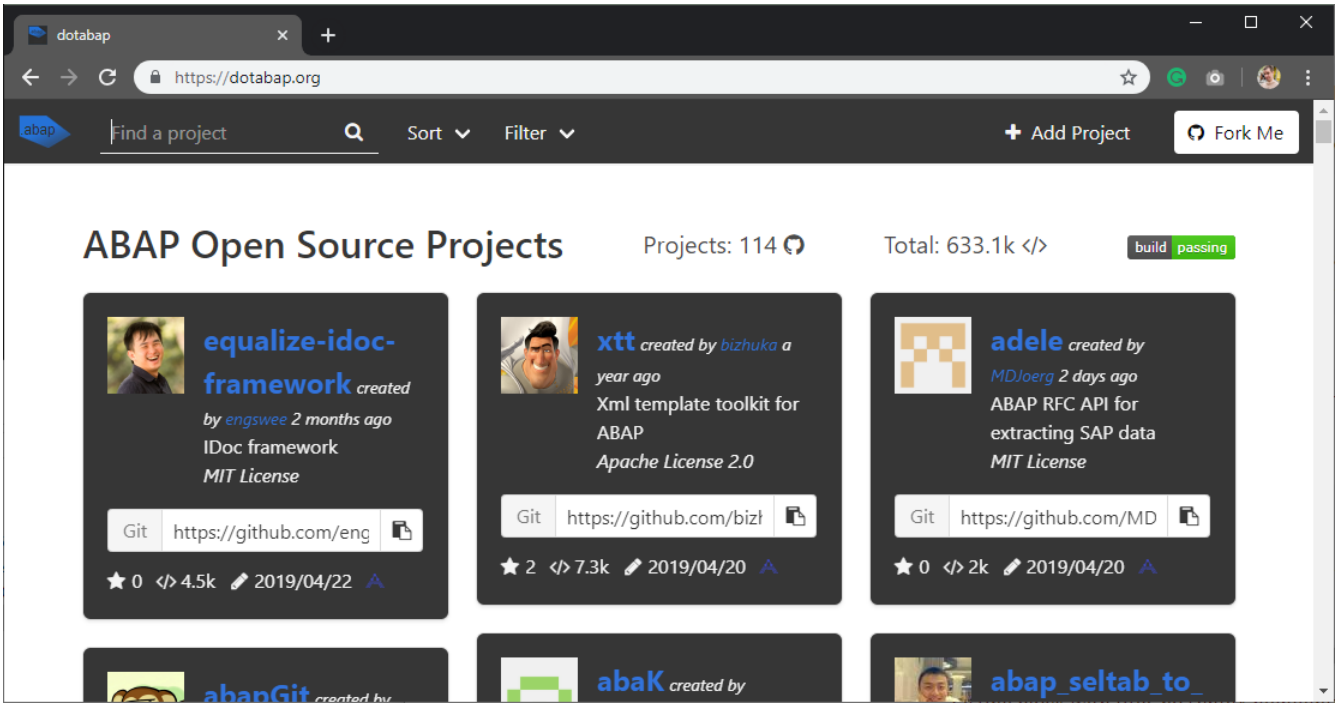
- GitHub
- GitLab
- Bitbucket
- Azure DevOps
- AWS CodeCommit
- Assembla
- abapGitServer

abapGitServer is a git server implemented in ABAP, so the code does not have to leave the application server.

1.4. Community

The community around abapGit has steadily grown over the years, around 70 developers from the community have contributed code to abapGit, plus more have helped suggesting features or finding bugs in the source code.

Ever since the Code Exchanged was closed in 2013 it has been difficult to discover ABAP open source projects. The "Explore" feature in abapGit helps users to find and reuse existing ABAP code from across the community, currently dotabap.org lists more than 100 projects totalling 600000+ lines of open source ABAP code.



The first abapGit Meetup was held in 2018, along with multiple talks at various past and upcoming SAP inside tracks. 2019 also features a new community event format, abapGit Bunkai which will take place during 2019.

2. Community Projects

Multiple community projects provides extra possibilities in the ABAP development workflow. This chapter lists a subset of the projects to watch in the future.

2.1. ci-lib

https://github.com/flaiker/ci-lib

2.1.1. Description

todo

2.2. abap-adt-api

https://github.com/marcellourbani/abap-adt-api

2.2.1. Description

todo

2.3. abap-doc-export

https://github.com/Sirius-A/abap-doc-export

2.3.1. Description

todo

2.4. vscode_abap_remote_fs

https://github.com/marcellourbani/vscode_abap_remote_fs

2.4.1. Description

todo

2.5. abap_git_hosts_apis

https://github.com/abapGit/abap_git_hosts_apis

2.5.1. Description

todo

2.6. abapCI

https://github.com/andau/abapCI

2.6.1. Description

todo

2.7. sapcli

https://github.com/jfilak/sapcli

2.7.1. Description

Leverages ADT HTTP API to allow executing ABAP development operations without human interaction in order to enable high level of scripting and integration with tools like Jenkins, Jira, or GitHub.

Use case examples

- Triggering quality checks (ABAP Unit, ATC) from CI tools
- Automating CTS operations (e.g. releasing transports after GitHub review)
- Deploying ABAP Objects from git repositories created by abapGit

Inspired by OpenStackClient, AWS CLI, Azure CLI, Kubectl, Cloud Foundry CLI and many other command line tools used by IT professionals to automate their tasks and work efficiently from command line.

2.8. abap-transmogrify

https://github.com/larshp/abap-transmogrify

2.8.1. Description

todo

2.9. abaplint

https://github.com/larshp/abaplint

2.9.1. Description

todo

2.10. abapGitServer

https://github.com/larshp/abapGitServer

2.10.1. Description

todo

3. Notes

todo, move chapter? rename chapter?

3.1. Branching

Branching in git is a central mechanism to separate development tasks. abapGit provides the basic git features for handling branches, the feature is delivered to the users, and the users decide how to use the features. As abapGit runs on the ABAP Application Server and developers use the Application server for development, all development is subject to the how the kernel works. The ABAP Kernel only allows one version of an object to be active at a time, ie. developers cannot have multiple versions of the same program running at the same time, this also means typically git branching is not possible within the same ABAP system.

If more active branches are required for development or testing, more application servers are required in the landscape.

3.2. File Format

abapGit is developer focused, so the files generated by abapGit are developer focused, this means each file is stored in git in a format that is similar to what is shown in the developer tools.

And as the code is stored in git, no usernames, timestamps, states (e.g. active/inactive) or other system specific information should be part of the serialized object files.

3.3. Deployment

abapGit does not focus on deployment of code, however it can be used to deploy code to various systems.

Within a landscape it is recommended to use CTS for deploying to QA and Production environments.

Compared to abapGit, CTS is

- Reliable - CTS is well tested and has proven very stable over the years
- Auditable - File checksums ensures that the deployment process is auditable, whereas git is designed to be mutable, history can be changed back in time if needed
- Optimized - Good import performance and integrity can be ensured by CTS, where a abapGit import might cause dumps and long runtime

3.4. Testing

For modern development processes unit testing is important, abapGit does not help with this. However abapGit can help developers to be introduced to other development workflows, which in turn can raise the awareness and show the importance of automatic testing.

3.5. Containers

Currently no easy way exists for spawning an ABAP system running in a container. Hopefully SAP will provide additional tooling and best processes for building ABAP containers.

Once a ABAP container is spawned, sapcli or REST based tooling can be used to deploy the code from git.

4. Continuous Integration

Continuous integration is the process for providing feedback to the developer regarding recent changes.

In a git context the developer will do a commit or open a pull request and automatically be informed about the code. Typically this consists of compiling the code, static analysis, and running unit tests.

Note that the developer is only interested in feedback about the current changes, so information should be restricted to the current changes, not other changes ongoing in the same system(if any).

In git the developer will have a branch containing the changes, this branch is deployed to a isolated system, which runs the toolchain. This can be a ABAP application server running as a container, for each unit of work a system is spawned, code deployed, syntax check, code inspector, and unit tests are run.

However it might be difficult to spawn a ABAP system for each commit, so developers might be forced to use the same ABAP system for continuous integration, but note that this might give wrong information to the developer. Also note that development happens in parallel, so continuous integration should also run in parallel to ensure fast feedback.

abapGit is a community project and thus does not own any licences for running ABAP servers connected to processes on the internet, so it is not possible to run continuous integration using ABAP application servers for abapGit. Instead abapGit uses abaplint to perform limited static analysis and provide feedback for eg. naming and downport problems.

5. Continuous Delivery

Continuous delivery extends on continuous integration by automating the process of deployment of the code to the landscape.

hmm?

6. End to End Workflow

This chapter describes different options for a modern and feasible end-to-end ABAP development setup.

todo, There are different requirements and responsibilities in organizations and workflows.

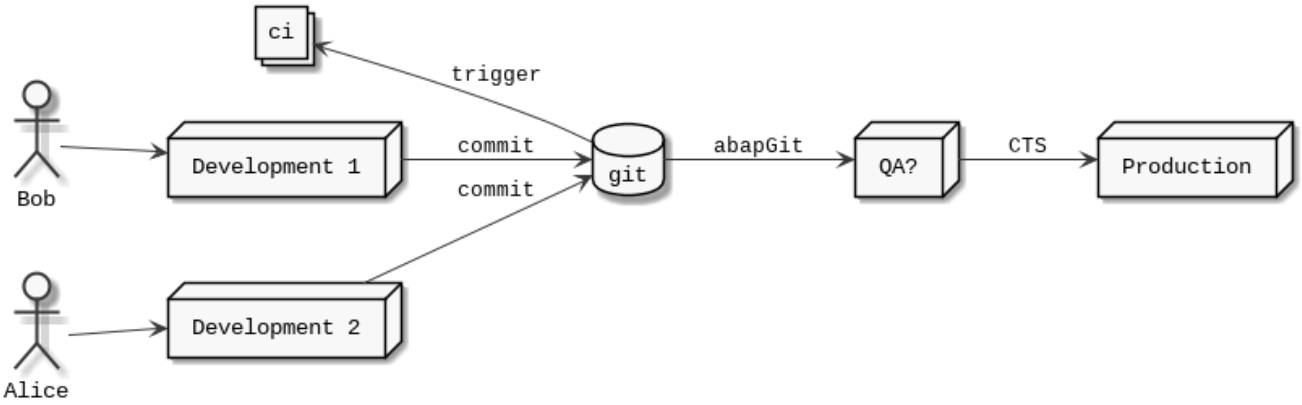
Inspiration, combinations of scenarios possible.

6.1. Decentral Development

- Every developer will have their own local development system
- Few requirements to other repositories/custom objects(optional)
- Organized package structure consisting of multiple repositories(optional)

https://searchsap.techtarget.com/tip/Implementing-modern-practices-in-an-ABAP-development-shop

todo



Both developers use git on their systems and do commits/branching, code is committed to a git repository which triggers CI.

? todo, then central QA system ?

6.2. Product Development

- Release cycles
- Central authority for selecting features
- Production system not critical for business operations

todo

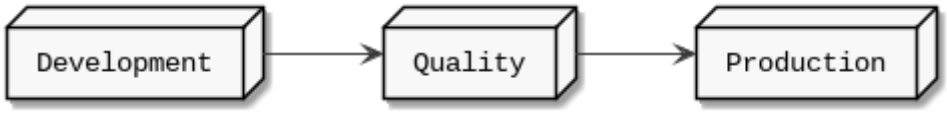
tag releases

6.3. Three Tier Central

- All developers on one central ABAP system
- Dev, QA, Production landscape
- Continuous delivery to production
- Cherry picking features
- Each transport is a feature is a branch
- Leverage modern development tooling/processes
- Let ABAP developers work with normal tools

6.3.1. Setup

In a simple 3 tier system setup, testing is done by business in the quality assurance system after the developer has released the transport. The transport is moved to production after business users have accepted the test.



Assumption: The ideal world is a continuous delivery setup, where features are moved to production as early as possible. I.e. no predefined release and testing cycles.

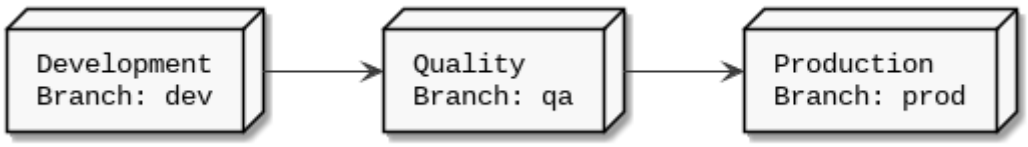
Assumption: The number of quality assurance systems in the landscape is much lower than the number of features being implemented. It would be nice to spawn one quality system per feature, but this is not feasible due to cost and integration to other systems in the landscape.

Risk is increased as multiple features are tested in the same system. Some of this risk can be mitigated using static analysis and continuous integration.

6.3.2. Git Branching Strategy

Every transport corresponds to a feature which corresponds to a branch.

Every tier in the landscape has a corresponding branch, which at any given time contains a snapshot of the code in that system.



The developer opens a transport and implements the required changes. abapGit runs in background and automatically creates a branch for each transport and commits the latest code at regular intervals.

OPTION: Let the developers open branches and do commits manually.

OPTION: Decentral development

Once the PR is approved, a transport of copies is released from development and imported to quality system.

When business approves the feature, transport is released from development and imported to quality + production, plus feature merged to the `prod` branch.

6.4. Mitigating Risk

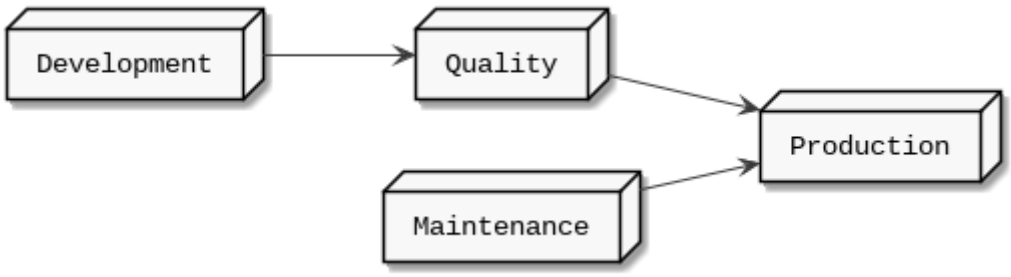
todo

CI

Static analysis

6.5. With Maintenance Branch

- An extra system is added to the landscape for hotfixes to production
- Few hotfixes are required and development effort for these is low
- Maintenance reflects the production system except for ongoing hotfixes



7. The Future

7.1. abapGit

- ADT
- Local editing and runtime
- Requirements
- Translations
- Stash?
- REST API
- Increase CI coverage
- Dependencies= APACK + Git Dependencies

More examples around using git in ABAP environments

7.2. ABAP Development

more editor options(vscode, language server)

local development

cross compiling