Version Control in DNA

## Introduction

Version Control is an important part of any development process. It helps revert changes, as well as track forward changes effectively to ensure smooth and ongoing development.

Version control also helps us keep the product afloat in case of bugs, crashes, PROD issues, or any other unlikely event(s) which may hamper the product.

Another useful feature of version control is to let various teams work on their own thing, collaborate, and maintain multiple versions of the product.

## Objectives of Version Control

- To maintain a history of changes in the codebase.

- To enable collaborative development across teams.

- To ensure the stability and reliability of releases.

- To streamline the review and approval processes for code changes.

## Version Control Tool/Platform

At The Phoenix Group, we use GitHub for version control.

Link: <https://att.sharepoint.com/sites/DevCentral/SitePages/Logging-into-GitHub-with-User-ID(1).aspx>

NOTE: To open the GitHub repository, you will need to login to the AT&T HVD first.

## Key Features

- Branching, forking, and merging.

- Distributed or centralized repository support.

- Integration with CI/CD pipelines.

## Repository Structure

**MASTER:** This is the most important branch of the entire project. The master branch can only be used to clone or pull data from. The master branch will serve as the backup branch where the last known stable PROD ready code will be stored.

As a developer, do not, under any circumstances, push your code to master branch.

**PROD:** Thisis the branch used for the deployment of production ready code with all the sign-off granted features. When post-production testing and sanity testing complete, the stable code on the production branch will become the last known stable production-ready code and will be merged with the MASTER branch.

**DEVELOP:** This branch will act as the codebase for all features currently under development. All new features/changes/fixes, etc. must be developed within this branch.

Note: It is recommended you create your own sub-branches for your specific features and regularly commit and push your code to your individual sub-branch.

**RELEASE:** This is where pre-production testing will happen with all the signed-off features ready to be deployed. This is a temporary branch which will be deleted when merged with PROD.

**FEATURE\_feature\_name:** These branches are specific to the individual features each team will be working on. Each feature branch will be cloned form the master branch at the start of a new development cycle.

**UAT:** This is where all the developed and unit tested code will be pushed to – the code on this branch will be used to deploy on the UAT environment.

**HOT\_FIX:** This branch is cloned from master in case of any large scale or critical post-production issues and deployed while those issues are being fixed.

## Branching and Merging Strategy

In DNA PROD and development, we follow the following branching and merging strategy to ensure optimal and smooth work that offers the best results every time:

Flow is as follows:

1. Development cycle starts.
2. Development team members clone the {MASTER} branch into their respective {FEATURE} branches by taking a pull from {MASTER}.

This will happen irrespective of whether the features in progress have been fully implemented or not. The idea is to have the latest production ready and stable code into the local repositories of each developer as well - this way, they will keep making their latest developments with updated code.

1. When development of a feature is done, that feature branch will be merged into the {DEVELOP} branch. The code on the {DEVELOP} branch will be used to deploy at the dev environment.  
   The feature branch WILL NOT BE DELETED just yet.
2. All unit testing is done on the code on the {DEVELOP} branch.  
   Any changes to be made will be made by switching back to your respective {FEATURE} branch, and then merge with {DEVELOP} branch again.
3. When the development team is satisfied with the feature development, the feature branch will be merged into the {UAT/QA} branch.   
   The code on the {UAT/QA} branch will be utilized for deployment to the UAT environment.
4. The QA team completes their round of testing, and a go-ahead is received for the specific features that have been granted a sign-off for deployment.
5. All respective feature branches will merge to create {RELEASE}, a temporary branch.
6. Pre-PROD testing is done on the UAT environment on only the signed-off features and given a go-ahead for deployment.
7. {RELEASE} is merged into the {PROD} branch; {RELEASE} gets deleted.
8. Deployment team then uses the latest code on {PROD} to perform deployment on the PROD server.
9. Post-PROD testing and sanity tests happen.
10. When deployed PROD code is deemed to be stable, {PROD} is merged with {MASTER}.
11. {MASTER} branch now has the last known stable code.

### HOT\_FIX Scenario

In case issues/bugs are reported during post-prod testing,

1. Data will be pulled from the {MASTER} branch and deployed temporarily while post-prod issues are being addressed.
2. A temporary branch, {HOT\_FIX} will be cloned from the production server. All hot fixes will be applied here, and the code will be pushed into {RELEASE}.
3. Development and QA teams will collaborate to resolve these post-production issues and push the codebase post-resolution to the {PROD} branch.
4. The fixed code in {PROD} is then deployed to the production environment.
5. Deployed code is tested for sanity and successful working.
6. When deemed at stable, this code will now be merged into {MASTER}.
7. {MASTER} branch now has the last known stable code.

## Conventions

It is expected of all teams working on DNA, its projects and initiatives to follow the same naming conventions along with the aforementioned branching and merging strategy.

This will create a standardized and uniform version control system that everyone in the entire project can independently understand and get on board with.

Use small characters, even though this guide has used capitals to name all your branches.

Use hyphens to separate words in the same norm and underscores to separate the norms. For example, a feature branch that seeks to implement a tail spend dashboard in development phase can be named tail-spend-dashboard\_php\_development. Here, **tail spend dashboard** is one norm, **php** is another and **development** is another. Those three have been separated by an underscore.

## Commit Guidelines

**Frequency:** Usual development guidelines say, “every logical change should be committed.”

For DNA, we follow the same guiding principle of committing every logical change.

**Message Format:** Start with a capitalized verb (e.g., "ADD", "FIX", "UPDATE") and then describe your change. For example, “FIX: Resolve API timeout issue in authentication module”.

There is no further restriction on how a commit message should be.

## Merge Protocol

* Use pull requests (PRs) or merge requests (MRs) for all merges.
* Perform peer reviews and testing before merging.
* Resolve conflicts before integration.

## Access and Permissions

* **Developers:** Read/write access to `feature` and `develop` branches.
* **Reviewers:** Approval rights for Pull Requests / Merge Requests.
* **Admins:** Manage repository settings and permissions.

## Backup and Recovery

- Enable periodic backups of the repository.

- Define recovery procedures in case of data loss.

## Customization

This document is designed to only serve as a baseline. For any additional version control operation that lies outside of the purview of this document, please engage with stakeholders to adapt the practices based on project-specific needs.

These needs can be:

* Additional branches or workflows.
* Integration with other tools (e.g., JIRA, Jenkins).
* Enhanced permissions or access controls.

## References

- Official documentation for GitHub can be found here: <https://docs.github.com/en> .

## Annexure

### Naming Convention Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Lifecycle** | **Type** | **Protected** | **Required/optional** | **Data** | **prefixes** |
| MASTER | Long-lived | Shared | Yes | Required as Backup of LKSV | Ingested | master/ |
| PROD | Long-lived | Shared | Yes | Required | Ingested | prod/ |
| RELEASE | Short-Lived | Shared | No | Required for merge of signed-off features and regression testing | Ingested | release/ |
| UAT | Long-lived | Shared | Yes | Required for client-side testing and deployment to UAT server | Ingested | uat/ |
| DEV | Long-lived | Shared | Yes | Required for feature collab and testing | Ingested | dev/ |
| FEATURE | Short-lived | Individual | No | Should be used for all individual changes | Cloned | feature\_ |
| HOT\_FIX | Short-lived | Individual | No | Optional, required only during P1 issues. | Cloned | hotfix/ |

### Flow Illustration

A diagram of a company

Description automatically generated with medium confidence

# Folder Structure

The following folder structure is to be used and followed for maintaining database files and objects such as SPs, VIEWS, FACT TABLES and more:

|-- schema\_name // [folder]

| |-- tables // [folder, as applicable]

| | |-- table\_1 // (definition of the table – SQL File)

| | |-- table\_2 // (definition of the table – SQL File)

| |-- functions // [folder, as applicable]

| | |-- function\_1 // (definition of the function – SQL File)

| | |-- function\_2 // (definition of the function – SQL File)

| |-- procedures // [folder, as applicable]

| | |--procedure\_1 // (definition of the procedure – SQL File)

| | |--procedure\_2 // (definition of the procedure – SQL File)

| |-- views // [folder, as applicable]

| | |--view\_1 // (definition of the view – SQL File)

| | |--view\_2 // (definition of the view – SQL File)