Paper TT02

All about versioning – an intro into Distributed Version Control

Mark Bynens, Janssen R&D, Beerse, BE
Sumesh Kalappurakal, Janssen R&D, NJ, US

ABSTRACT

Today as software developers, programmers or data scientists we need to work as a team and engage in collaborations to share our work. R, Python and other open-source software are globally used in pharma in various setups. As more open-source code has been developed, there is a huge need to collaborate and secure our code in a global environment. Repository management services have become a crucial part of collaborative software development. They can help software developers, programmers and data scientist organize their coding work, maintain quality, collaborate and share. This paper introduces collaborative software development, versioning, different types of version control systems, repository management services in general and will compare GitHub, Bitbucket, and GitLab.

INTRODUCTION

Software development has always been a collaborative undertaking. Although it is possible that one person designs, develops, implements, tests, documents and maintains the software, the developer still needs to satisfy end-user(s) requirements. In a team of multiple developers collaboration is even of a greater importance as developers must communicate not only with the end-user but also with each other in an iterative fashion.[1]

Teams can be geographically distributed, developers can work from home or the office, travel budgets are reduced, … there is a need for a robust set of tools to allow developers to work together from distributed and dynamic locations. These collaboration tools can structure collaboration, make it manageable and reusable. They are often indispensable in leading a software development project to success. [2][3]

Collaboration among individuals, from users to developers, is key to modern software engineering.

THE SOURCE

While there are a lot of tools on the market that help with communication and managing projects and tasks, we will begin by looking at the life-blood of any software development process: the (source) code. If you are developing software, it is a given that your project will include some (source) code. Every decision, big or small, will affect the (source) code in some way. [1]

How can developers in a distributed development team collaborate on a set of constantly changing (source) code files? How can they do it securely and efficiently? Some measure of control must be put into place or the effort might dissolve into chaos. [1] This brings us to the topic of this paper “versioning”. We will look at versioning, different types of version control systems, repository management services in general and will compare GitHub®, Bitbucket®, and GitLab®.

SOURCE CONTROL

Source control also known as revision control or version control is the practice of tracking and managing changes to documents, computer programs, large web sites, and other collections of information.[4] It’s a system that records changes to a file or set of files and allows you to roll back to a previous state, compare changes over time, retrieve the last version, etc... [5] Version control (revision control) applies to nearly any type of file on a computer, we will use software code as files being version controlled in this paper.

The various types of version control systems are:

- Local Version Control System
- Centralized Version Control System
- Distributed Version Control System

LOCAL VERSION CONTROL SYSTEM

Manually maintaining multiple versions of files is highly prone to error. If you copy and rename files, it's easy to use the wrong name for renaming so you cannot distinguish easily anymore between the different versions.

A **local version control system** was the first successful attempt to solve this issue.[5][6]

This system maintained track of files and stored all of the information within the local system. It is one of the simplest forms of version control and kept all the changes to the files under revision control in a database. At first, the new versions of the complete file were stored, afterwards only the differences between 2 versions where stored. So, the first version would be the file, each successive version would contain the difference between the current and the last version. This saved memory cost. These differences between files were called patch-sets and a local database was used to store these patch-sets.[7] Retrieval of a version of a file was done through a reconstruction of patches.

We have to go back to the 70's for the first generations of local version control systems. In 1972, the Source Code Control System (SCCS) was developed in SNOBOL at Bell Lab Labs by Marc Rochkind for an IBM System/370 computer. [25]

One of the most popular local version control systems was the Revision Control System (RCS)**,** which is still distributed with many computers. Developed as a successor and an alternative tool to SCCS, it was first released in 1982 by Walter F. Tichy at Purdue University. [8] It managed multiple revisions of files and automated the storing, retrieval, logging, identification, and merging of revisions. RCS was useful for programs or text that was revised frequently. It kept patch-sets of differences between files. By adding up all the patches it could then re-create what any file looked like at any point in time.

As files with their versions were stored locally and were not accessible to other users wanting to work on the same files, local version controls systems were very useful for single users but not for teamwork.

CENTRALIZED VERSION CONTROL SYSTEMS

For developers to interact on different systems or projects, Centralized Version Control Systems (CVCSs) were developed. These systems are based on a client-server architecture. There is a central server with a single, centralized master copy of the code, master repository, with the entire history of all the changes from which developers request the latest version of work and push the latest changes too.[9]
In centralized version control there are 2 main elements:

- Master copy: the centralized master copy of the code which holds all the source files, as well as all the versions of the files.
- Local copy: a local copy of the files, which is on your computer

The local copy of files is also called working files as these will be edited by the developers, each user has their own working copy. A developer can pull what other developers have made from the central server, make their changes, make sure that they work properly and subsequently push their changes back to the central server so that other developers can see them.[10][11]

To allow only one developer to be working on a piece of code at any one time, files may be flagged or even locked also called 'checked out'. The file is marked as being "in use" by a particular developer and alerting other developers the file is being modified. When the developer checks their code back in, the lock is released and it's available for other developers to check out.

As a centralized version control system is based on a client-server model, administrators have control over users and access.[12]

A drawback of a centralized version control system is that when the central server goes down the developers can not pull down any changes from the central server or push any updates to the central server. Centralized version control heavily depends on access to the central server. This also means that if the central server crashes or is corrupted this might result in losing the entire data of a project or multiple projects. Another drawback is that the central repository is not locally available and to perform any action on the central repository you need to be connected to a network.[13]

DISTRIBUTED VERSION CONTROL SYSTEMS

Where centralized version control systems uses a client-server approach to version control, distributed version control systems use a peer to peer approach. There is no single central copy of the code or repository, instead, the repository including its full change history is replicated or "cloned" onto each developer's local machine. As a developer doesn't need to be connected to the central repository to perform version control tasks locally, distributed version control systems can be seen as self-contained. [13] [14]

Does that mean that there cannot be a central project repository? Technically all repositories are equivalent in the distributed peer to peer architecture. In most cases and practice, the repositories will be organized in a social hierarchy and at last, one will be marked as the central repository which is an authoritative one, where the latest team-approved changes are expected to be found. In this primary project repository "official" code releases are created. [15] [16]

Having several remote repositories also means that you can collaborate with different groups of developers in different ways simultaneously within the same project. Several different types of workflows that aren't possible in centralized version control systems can be set up. [17]

To get a more in-depth idea of distributed version control systems and its features, let's look at one of the most popular distributed version control systems: GIT, a free and open-source version control system

GIT

Git is a version control system that is rapidly becoming the standard for open-source projects.

HISTORY

In developing the Linux kernel a very large distributed team of developers were involved and as they were struggling with revision management, the Linux kernel team adopted a scalable distributed version control product called BitKeeper in 2002. Although it was a closed source, proprietary distributed source control product a free community version was used for the development of the Linux kernel. Except for being controversial to use a proprietary product for an open source project it gave not many problems.

Until April 2005 when BitMover, the company behind BitKeeper, announced it would discontinue the free community version of BitKeeper. Linus Torvalds looked around for other off-the-shelf systems but none of them suited the kernel development team's performance needs or wanted features. So, he wrote a new version control system from scratch, based on lessons he learned from using BitKeeper, and Git was born. Initial development was said to have taken a few days.

Since its birth in 2005, Git has evolved and matured. It's easier to use while keeping its initial qualities: incredibly fast, very efficient from small to very large projects and supports non-linear development with an incredible branching system. [18]

GIT FUNDAMENTALS

First, we will go over the basics as it is important to well understand the fundamentals of how Git works and how it stores and thinks about information.

SNAPSHOT BASED VERSION CONTROL

Most other version control systems are delta-based version control systems. They store information as a list of file-based changes, as a set of files and the changes made to each file over time.

Git doesn't track the changes of each file individually. Git stores data as snapshots of the project over time.
A snapshot essentially reads all the files, stores them as a whole, as a new reference and then compares it with the previous version. Git doesn't store the file again if the file hasn't changed, just a link to the previous identical file it has already stored. Every time you save the state of your project in git, it becomes a snapshot because it has references to all the files and not just for the ones which were changed. All the files can thus be generated from any snapshot easily.

WORKING LOCALLY
Most operations in Git are done locally and only need local files and resources to operate. Since the entire history of the project is available on your local disk most operations can be performed quickly. If you want to do a little work on an airplane or a train, you can work locally until you get to a network connection to upload. Note that Git can also be used as a local version control system. It does not require a remote repository.

DATA INTEGRITY
Before everything is stored in Git it is check-summed and is then referred to by that checksum. In this way Git is secure, and it maintains the integrity of the content being version controlled. It is impossible to change the content of any file or directory, loose information or tamper with files without Git knowing about it or detecting it. Git objects in the Git repository like the content of the files, the relationships between files and directories, versions, tags and commits are secured with a cryptographically secure hashing algorithm called SHA1, the mechanism that Git uses for check summing.
This algorithm generates based on the contents of a file or directory structure in Git a unique 40-character string composed of hexadecimal characters (0–9 and a–f). A SHA-1 hash looks something like this: a6eb74ea214a6cfa6771f4be041d5cce7bee3e34. Because Git uses them so much, you will see these hash values everywhere in Git.

GIT BASICS
Let's have a look at some git basics: getting a Git repository, git file status lifecycle and recording changes to the repository.

GETTING A GIT REPOSITORY
There are mainly 2 approaches to getting a Git repository. The first takes an existing project or directory and imports it into Git. The second clones an existing Git repository from another server.

GIT FILE STATUS LIFECYCLE
In a Git repository, files pass through different states. There are 4 states a file can go through: untracked, tracked, modified and staged. During development, the files can go through each of these states many times. Let's have a look at each state.

- Untracked: if the file didn't exist before, and you run git status, you see the file under the untracked files. Untracked means that Git sees a file you didn't have in the previous snapshot.
- Tracked: to begin tracking the new file, you need to add the file. From this point on until you change the status to untracked, git will be watching this file actively, and any changes you make to it will be noted.
- Modified: means that you have changed the file but have not marked it to go into the next commit.
- Staged: means that you have marked a modified file in its current version to go into your next commit snapshot.

RECORDING CHANGES TO THE REPOSITORY
Now that we have a Git project, how do we record the changes? Let's demonstrate this by adding and changing a new file.

If the file didn't exist before, and you run git status, you see the file under the untracked files. To begin tracking the new file, you need to add the file. When you add the file with the git add command, the file is tracked. Once the file is being tracked any changes to it will be detected by git.

Let's say that now we start working on that file and we do some changes to it. Once we save the file it will mark with a modified status. The tracked file is now modified but not yet staged. To stage it, you need to add it again. If the files are staged, they will go into the next commit.

Now that your staging area is set up the way you want it, you can commit your changes. Remember that anything that is still unstaged won't go into this commit. They will stay as modified files on your disk. The simplest way to commit is to type the commit command. The changes are now safely stored in your local database. Issuing the commit command will also take all the files back to the tracked status, ready to begin the whole process again.

To send our changes to a remote repository we need to use the git push command.

ADVANCED GIT

GIT BRANCHING

A branch represents an independent line of development. All new commits are recorded in the history of the branch resulting in a fork in the history of the project. When you want to add a new feature or fix a bug, a new branch can be spawned to encapsulate those changes. This way unstable code cannot so easily get merged into the main codebase, and the new feature's history can be cleaned up before merging it into the main branch.

The git branch command only creates, list, rename, and deletes branches. To switch between branches, you need to use a git checkout command and to put a forked history back together again you need a git merge command. [17]

CHECKING OUT BRANCHES

The git checkout command lets you navigate between the branches created by git branch by updating the files in the working directory to match the version stored in the branch you want to check out. It tells Git to record all new commits on that branch. Think of it as a way to switch between workspaces and to select which line of development you want to work on.
Having a dedicated branch for each new feature or bug fix makes it easy to try something out without the fear of destroying existing functionality and makes it possible to work on many features or bug fixes at the same time. [17]

GIT MERGE

The git merge command lets you integrate changes from multiple independent lines of development into a single branch. It's Git's way of putting a forked history back together again and will combine multiple sequences of commits into one unified history. [17]

3-WAY MERGE

A 3-way merge uses three commits to generate the merge commit: the two branch tips and their common ancestor.

When a git (3-way) merge is used for to combine two branches, the merge takes the 2 branch tips and will find a common base between them. Once Git finds a common base commit it will create a new "merge commit" that results from combining the changes from the two branches from the point where those two branches diverged. Git will try to do the combining automatically. Git will be unable to do this automatically when it encounters a piece of data that is changed in both branches' histories. This is known as a version control conflict. A user intervention is needed to resolve this. As they have two parent commits "merge commits" are unique against other commits. [17]

FAST FORWARD MERGE

A fast-forward merge can be done when there is a linear path from one branch tip to another branch tip. All Git has to do to integrate both histories is to move or "fast forward" one branch tip up to the other branch tip. Since all the changes or commits from both branches are now available on one branch both histories have been effectively combined.

When there is no linear path from one branch to another a fast-forward merge is not possible. This is the case if the branches have diverged. Git has no choice but to combine them via a 3-way merge.[17]

RESOLVING CONFLICT

When the two branches both changed the same part of the same file, Git won't be able to figure out what to use for the merge and Git will not be able to automatically merge the two branches. When such a situation occurs, Git stops right before the merge commit so that the conflicts can be resolved manually through a user intervention. Merge conflicts will only occur in the case of a 3-way merge. In a fast-forward merge it is not possible to have conflicting changes.[17]

REBASING

In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase. [17]

THE BASIC REBASE

The easiest way to integrate 2 branches is the merge command. There is however another way, called rebasing in Git, where you can take all the changes that were committed on one branch and replay them on another one. As the completed work from one branch is transferred to another branch, the history is flattened. After integration there is no difference in the end product between a merge or a rebase, but rebasing makes for a cleaner history. The history looks like a linear history: even when originally the work happened in parallel, it appears that all the work happened in series. This is because rebasing replays changes from one branch onto another in the order they were introduced, whereas merging takes the endpoints and merges them. [17]

COLLABORATING WITH GIT

WORKING WITH REMOTES

To be able to collaborate with Git on a Git project, you need to have a remote repository for that project. A remote repository is a version of your project that is hosted on the Internet or on a network somewhere. To share your work with others you'll need to able to sync your work from your local repository with the remote repository frequently. Collaborating with others involves pushing and pulling data to and from and managing a remote repository. You can be part of several remotes and have either read-only or read/write access to them. [17]

FETCHING AND PULLING FROM YOUR REMOTES

To get data from a remote repository, you can run a git fetch command. The command goes out to that remote project and pulls down all the data from that remote project to your local repository that you don't have yet. Afterwards you should have references to all the branches from that remote, which you can merge in, inspect or modify at any time. The command doesn't automatically merge the fetched data with any of your work or modify what you're currently working on. You have to merge it manually into your work.

If you want to automatically fetch and then merge data from the remote repository into the code you're currently working on, you can use the git pull command. Running git pull generally fetches data from the server you originally cloned from and automatically tries to merge it into your current code. [17]

PUSHING TO YOUR REMOTES

When you want to publish local repository content on the remote repository, you can use the git push command. In that way other team members can see the changes you made to your local repository and update their local repository with it. Pushing is how you transfer commits from your local repository to a remote repository. [17]

GIT COMPARED

SHORTCOMINGS OF GIT

Some of the shortcomings of Git should also be mentioned. Git is most efficient at version controlling text-based files i.e. scripts, config files, notes, documentation. It can be also used to version control binaries, databases, pictures or word and excel documents. However, this often requires some additional configuration or some custom coding and is not always satisfactory.

OTHER VERSION CONTROL SYSTEMS

Even though Git is the most used version control software overall, the largest competitors of Git are SVN® and Mercurial® and some niche areas have specific solutions (often based on Git of SVN in the background):

- DVC®: an open-source version control system for machine learning projects for data scientists. Manages the whole flow i.e. version control for data, scripts, and output[19]
- Word and Excel: SharePoint® keeps versions after someone edits an online file
- Versions®: git for designers[20]
- Database versioning[21]

The one version control system that every developer is currently waiting on for it to be open-sourced by Google is Piper®.[22] It is considered to be the holy grail of version control, so Google will probably not give this trade secret up in the next few years …

REPOSITORY MANAGEMENT SERVICES

Repository management services are third-party web applications that wrap and enhance a version control system. The usage of a repository management service cannot be seen separate from the usage of the underlying version control system. Each repository management service has various support for underlying version control systems. There is then also a wide variety of modern software repository management services available to choose from. Each come with their own strengths and weaknesses.

Repository management services are key components of collaborative software development. They enable software developers to manage changes to the source code and related files, create and maintain multiple versions in one central place. The enable teams to move faster and preserve efficiency as they grow bigger. Even if you work in a small team or solo, there are numerous benefits of using them.

It is important to acknowledge that repository management services and version control systems are two separate entities. Version control systems are the low-level command-line utilities that are used to manage the software development life cycle changes to a collection of source code files.

A good repository management service will provide tools for discussing, managing, measuring and monitoring software development in an efficient and organized way.

We will briefly introduce and compare 3 popular repository management services GitHub, GitLab, and Bitbucket by touching on multiple aspects including basic features, relationship to open source, importing repositories, free plans, cloud-hosted plans, and self-hosted plans.

GITHUB

GitHub is web-based version control and collaboration platform for software developers founded on Git. GitHub was launched in 2008 by Tom Preston-Werner, Chris Wanstrath, and PJ Hyatt. It was one of the first Git hosting platforms. The open-source community started to use it for code sharing. It made GitHub an instant success, so the platform started to gain lots of users. Today GitHub has about more than 40 million users and hosts more than 100 million repositories. In 2018 Microsoft is acquired GitHub for a jaw-dropping price tag of 7.5 billion dollars.
GitHub charges for the use of private repositories but allows developers to change, adapt and improve software from its public repositories for free. GitHub provides a web interface to the Git code repository and management tools for collaboration. GitHub can be seen as a social networking site for software developers. Members can receive updates for specific projects, communicate publicly or privately, follow each other and rate each other's work. As GitHub is so intuitive to us, so useful for collaboration and its workflow is surprisingly simple and sane, even non-software developers have begun to use GitHub to work on text documents or other file types.

GITLAB

Gitlab was launched in 2011 by Dmitriy Zaporozhets and Valery Sizov providing an alternative to the already available repository management solutions. GitLab is an open-source alternative to GitHub. The acquisition of GitHub by Microsoft stirred up controversy in the open-source community and a lot of people were worried that GitHub would lose its open source roots. So, people were looking at alternatives like GitLab. Similar to GitHub, GitLab is a Git based repository hosting platform. From the start, GitLab wanted to distinguish itself from GitHub so it created a single product for the entire DevOps lifecycle. In GitLab tools like issue trackers, continuous integration and continuous delivery are part of the product. GitLab provides a single interface to the whole DevOps life cycle. Today GitLab is used by more than a hundred thousand organizations. Organizations like IBM, Sony, NASA, and Alibaba are using GitLab. GitLab is built on an open core model. That means there are two versions of GitLab: Community Edition and Enterprise Edition. GitLab Community Edition is open source. GitLab Enterprise Edition uses the same core but adds additional features and functionality on top of the GitLab Community Edition. The GitLab Enterprise Edition is under a proprietary license.

BITBUCKET

Bitbucket was created by Jesper Nøhr in 2008 as a small start-up, supporting Mercurial projects. In 2010 Bitbucket was acquired by Atlassian and from 2011 it also started to support Git hosting, which is now its main focus. It integrates smoothly with other services from Atlassian like Jira, HipChat, Confluence and Bamboo.

Bitbucket is a repository management solution designed for professional teams for source code and development projects. It gives you a central place to collaborate on your source code, guide you through the development flow and manage your repositories. Bitbucket offers both commercial plans and free accounts. In April 2019 Atlassian announced that Bitbucket reached 10 million registered users and over 28 million repositories.

Bitbucket has three deployment models: Cloud, Server, and Data Center.[23][24]

COMPARE

BASIC FEATURES

Each of the 3 platforms has its features and capabilities. As they serve the same goals and implement the same methods and if we are only looking at basic features, they show a lot of similarities:

- Pull request;
- Code review;
- Integrated editing;
- Bug tracking; Markdown support;
- Two-factor authentication;
- API with extended features;
- Forks/clones of repositories;
- Snippets;
- 3rd party integrations
- Advanced permission management

For a more detailed feature comparison please visit the feature pages of Bitbucket, GitHub, GitLab.[28]

OPEN SOURCE

From GitHub, GitLab and Bitbucket only GitLab has an open-source version. GitLab is built on an open core model. GitLab Community Edition is open source. The GitLab Enterprise Edition is under a proprietary license.

Although GitHub hosts the most open source projects its source code is not open source. Bitbucket is not open source. All three support open source projects. GitHub offers free public repositories, Bitbucket also offers free private repositories and GitLab offers an entirely free Community Edition. [28]

CONNECTING WITH OTHER DEVELOPERS

GitHub and Bitbucket offer the ability to easily follow other users. All have public repository discovery functions. Even though GitHub is not open source, it is still the hotbed of open source collaboration. With the free hosting of public projects and the early adoption of social features, it is a serious social hub for software developers. [28]

SUPPORTED REPOSITORIES

The ability to import and use previous projects is critical. Bitbucket, for the time being, stands out from the other 2 because this is the only one that supports Mercurial repositories. On June 1, 2020 however all Mercurial features and repositories will be officially removed from Bitbucket and its API.

GitHub and Bitbucket support importing repositories based on multiple different version control system. GitLab on the other hand only supports Git. Git is the most popular version control system but moving to GitLab could be complicated if you are using Mercurial or Subversion repositories at the moment.

GitHub supports: the import of Git, Subversion, Mercurial, Team Foundation Server.
GitLab supports: the import of Git and easy import from other services GitHub, Bitbucket, Google Code, Fogbugz.
Bitbucket supports: the import of CodePlex, GitHub, Google Code, SourceForge, Subversion, or another Git/Mercurial-based hosting site.

PAYMENT PLANS

FREE CLOUD-HOSTED PLANS

All the 3 providers offer a free cloud-hosted plan, but they have some significant differences when we look at the details.

GitHub free cloud-hosted plan allows you to host an unlimited number of public and private repositories. On private repositories there is however a limit of 3 collaborators. Included are also issues and bug tracking and project management features.

Bitbuckets free cloud-hosted plan let's individuals and small teams up to 5 users collaborate free on an unlimited number of private repositories. It includes Jira Software integration, Trello integration, CI/CD Integration and merge checks. Repositories here have a 1 GB soft size limit and a 2 GB hard size limit.

GitLab free cloud-hosted plan lets an unlimited number of users collaborate on an unlimited number of public and private projects. Repositories have a 10GB space limit which is a very generous compared to what the other 2 providers offer.

If you are looking for a free cloud-based solution for private projects GitLab's offer is probably the most appealing.

PAID CLOUD-HOSTED PLANS

With GitHub's individual pro account ($7 per user per month), you can easily get hold of the essential functionalities that you can always get in the free account, along with the ability to host any number of public and private repositories. There is no limit on how many users with a personal account can collaborate, but they can't use team features such as teams access control, user management and billing is done individualy. To make use of the team features, you can opt for a GitHub team plan, that starts at $25 per month and includes your first 5 users. You can add a user at the rate of $9 per month.

For Bitbucket, the cloud-hosted standard plan starts at 5 users for $15 a month. For every additional user you pay $3 per month extra. This plan covers an unlimited number of private repositories and an unlimited number of users with a Git large file storage limit of 5GB. It includes Jira Software integration, Trello integration, CI/CD Integration and merge checks. Not included are deployment permissions, IP Whitelisting, required two-step verification and smart mirroring. If you would like to have these included you can get a premium plan which starts at 5 users for $30 a month. For every additional user you pay $6 per month extra.

Gitlab has 3 cloud hosted plans: Bronze ($4 per user per month), Silver ($19 per user per month) and Gold ($99 per user per month). Silver has additional features compared to Bronze like deploy boards, priority support, … And Gold has additional features compared to Silver like roadmaps, free guest users, Kubernetes cluster monitoring, …

PAID SELF-HOSTED PLANS

GitHub, GitLab, and Bitbucket self-hosted versions come with several enhanced features compared to their cloud-hosted counterparts.

For GitHub you need to contact their sales team with regards to the pricing of their Enterprise plan. GitHub Enterprise can be deployed to your servers, AWS, Azure and Google Cloud Platform.

Bitbucket Server version starts at a one-time payment of $10 for 10 users and has a limit of 2000 users. If you need more than that we suggest that you should check out Bitbucket Data Center which starts at $1980 a year for 25 users. The Bitbucket Data Center version is cheaper than the Server version.

Like the GitLab cloud hosted plans, Gitlab also offers self-managed plans: Core (free), Starter ($4 per user per month), Premium ($19 per user per month) and Ultimate ($99 per user per month). The starter has additional features compared to Core like next business day support, multiple LDAP / AD server support, … Premium has additional features compared to Starter like KMDJMDTNPKKJMQ disaster recovery … And Ultimate has additional features compared to Premium like roadmaps, free guest users, Kubernetes cluster monitoring, …

CONCLUSION

Modern version control systems are designed in such a way that they help addressing problems that teams face when collaborating. They are used to manage the software development life cycle changes to a collection of source code files. To solve the problem of working together however it takes more than just a great version control system like Git. On top of Git, we also need a collaboration platform built where you can share your work, see your team's work, complete code reviews, and connect integrations that help you build, test, and deploy your code. Repository management services are third-party web applications that wrap and enhance a version control system. A good repository management service will provide tools for discussing, managing, measuring and monitoring software development in an efficient and organized way.

In this paper, we have compared 3 repository management services: Gitlab, Bitbucket, and Gitlab.

We cannot announce one service to be ultimately superior to the others. All of them are powerful and feature rich services. In certain scenarios we can nevertheless recommend a certain service:

- when working on open source projects, GitHub makes sense
- when you also want to use other products from Atlassian like Confluence or Jira, Bitbucket seems like a good choice
- when you want an open source solution, you should go with GitLab. [28]

One of the 3 repository hosting services can likely give you what you need. If it is not the case, then check out Assembla or CloudForge. Note that Microsoft, Amazon, and Google all have solutions for devops to store remote git repositories and automatically deploy to Azure, AWS, and Google cloud infrastructure respectively. [26] [27] Practically doing the same as GitHub, Bitbucket and GitLab.

REFERENCES

[1] https://www.codemag.com/Article/0203021/Collaborative-Development-Part-1Source-Control
[2] https://steelkiwi.com/blog/collaboration-is-a-key-to-project-success/
[3] https://sdtimes.com/collaboration-software-development-people-tools-run-show/
[4] https://en.wikipedia.org/wiki/Version_control
[5] https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
[6] https://subscription.packtpub.com/book/application_development/9781849517522/1/ch01lvl1sec12/types-of-version-control-systems
[7] https://www.toolsqa.com/git/distributed-version-control-systems/
[8] https://en.wikipedia.org/wiki/Revision_Control_System
[9] https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs
[10]https://www.perforce.com/blog/vcs/what-svn
[11] https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs
[12] https://medium.com/@kamaumike2013/what-is-version-control-54e4e6e187af
[13] https://pediaa.com/what-is-the-difference-between-centralized-and-distributed-version-control/
[14] https://en.wikipedia.org/wiki/Distributed_version_control
[15] https://medium.com/faun/centralized-vs-distributed-version-control-systems-a135091299f0
[16] https://sail.cs.queensu.ca/Downloads/2016_ContinuouslyMiningDistributedVersionControlSystems_AnEmpiricalStudyOfHowLinuxUsesGit.pdf
[17] https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control
[18] https://medium.com/@willhayjr/the-architecture-and-history-of-git-a-distributed-version-control-system-62b17dd37742
[19] https://dvc.org/
[20] https://versions.sympli.io/
[21] https://www.red-gate.com/products/sql-development/sql-source-control/
[22] https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext
[23] "Celebrating 10 million Bitbucket Cloud registered users". Bitbucket. 17 April 2019.
[24] "Bitbucket now rocks Git". *Bitbucket official blog. 3 October 2011*. Retrieved 18 October2011.
[25] https://www.debuggershub.com/version-control/
[26] https://docs.microsoft.com/en-us/azure/devops/get-started/?view=azure-devops
[27] https://aws.amazon.com/codecommit/.
[28] https://stackshare.io/stackups/bitbucket-vs-github-vs-gitlab

ACKNOWLEDGMENTS

We would like to thank Gayathri Kolandaivelu (Statistical Programming Leader) and especially Kim Vandendijk (Senior Analyst Application Services at Janssen R&D) for giving us advice and comments and reviewing this paper.

RECOMMENDED READING

- Git - Book: https://git-scm.com/book/en/v2
- Pro Git | Scott Chacon | Apress: https://link.springer.com/book/10.1007%2F978-1-4842-0076-6
- Learn Version Control with Git for Free: https://www.git-tower.com/learn/
- Version Control with Git: Powerful tools and techniques for collaborative software development eBook: Jon Loeliger, Matthew McCullough

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Mark Bynens
Janssen R&D
Turnhoutseweg 30, B-2340 Beerse
Belgium
Phone: +32 494 05 56 88
Email: mbynens@its.jnj.com

Sumesh Kalappurakal
Janssen R&D
920 Route 202 S, Raritan
New Jersey - 08869 USA
Email: SKalappu@its.jnj.com

DISCLAIMER:

The opinions expressed in this document are those of the authors and do not necessarily represent the opinions of PhUSE, members' respective companies or organizations, or the products' respective companies mentioned in this paper. Although GIT and GIT based remote repositories have been used to explain more about distributed version control systems and remote repositories, the authors believe that other distributed version control systems or other remote repositories can equally be used. It is the reader's responsibility to determine which distributed version control system or remote repository would best suit their need. Additionally, the authors do not endorse any specific commercial products or services.