



A Version control system that manages dependencies in large files

Rémi Arnaud  
David Gaya  
Maxime Helen

---

# Outline

Today's solutions?

Asset manager

Workflow

Benefits

Integration roadmap



# Today's solutions

## O1

Traditional Version control system

- Technology which is resource demanding
- Requires skilled staff to support
- Expensive
- Required for collaboration
- Critical for data protection

# Version control systems timeline



**CVS: Concurrent Versions System**



**Perforce®**



**SVN**



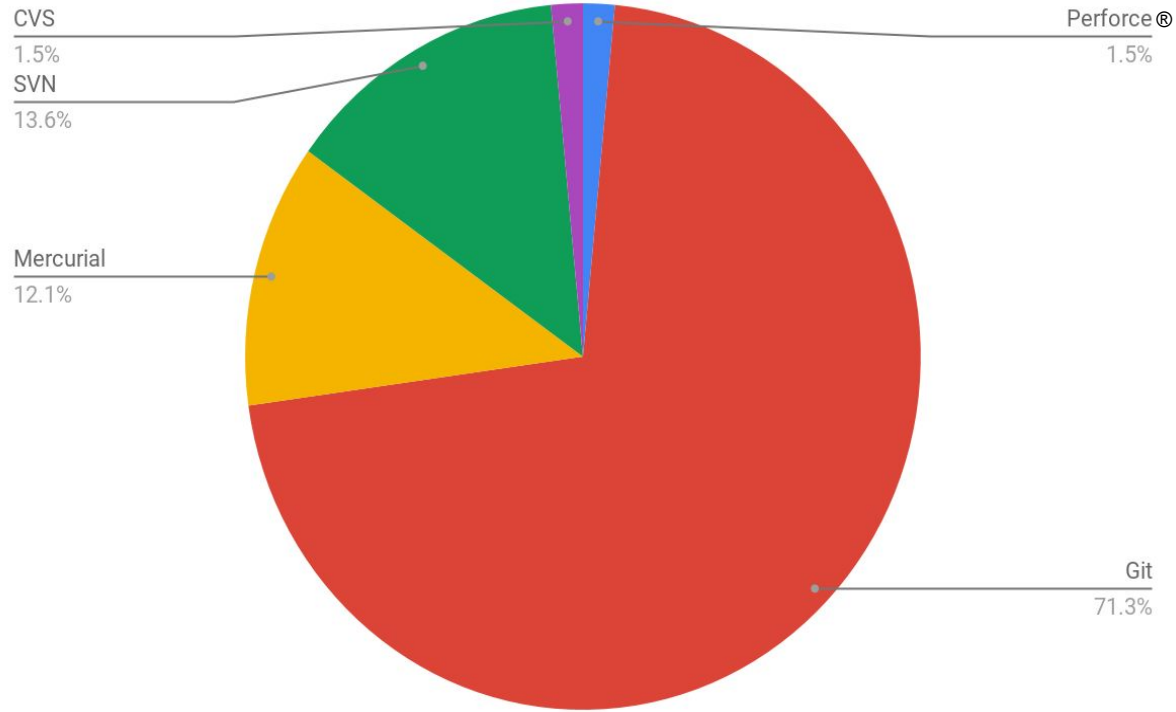
**Git**



**Git LFS: Large File Storage**




# Popularity (2016)



# Specificities



	Branch	Sparse checkout	Large binaries	Price	Handle file dependencies
Perforce®	No*	No	Yes	Not cheap	No
SVN	No*	Yes	Yes	Open source (pay for infrastructure)	No
Git/Mercurial	Yes	No	No	Open source (pay for infrastructure)	No
Git LFS	Yes	No	Yes	Not cheap	No



# Bilrost asset manager

## 02

What if dependencies are too  
complicated to control?

- A file can depend to another one.
- A file doesn't contain meta informations for indexing, for example:
  - User owning the file
  - Tags for being found quickly

# Bilrost asset



Entry	Meaning
Main	Main dependency if any
Dependencies	List of file dependencies
Description	Owner description
Tags	Asset tag names
Meta	Meta data related to the asset information: creation, modification, author...

- Is a container listing file pointers!
- Is meta-data stored internally by Bilrost
- Referenced by a uri:  
    /assets/[...namespaces]/[asset\_name]  
    eg. /assets/a/b/c
- Mutable only from Bilrost API
- Implements dependencies between files
- Allows indexing the meta-data



# Bilrost resource



- Represents files to version
- Is an asset dependency
- Referenced by a uri:  
    /resources/[...folders]/[file\_name]  
    eg. /resources/a/b/c.png
- Mutable from user file system within **workspaces**

# Bilrost project



- Contains versioned assets and resources organized within branches.
- Named under GitHub® policy format:  
[organization\_name]/[repository\_name]  
eg. org\_name/Game-assets
- Centralized in GitHub® server for assets and Amazon S3® for resources.
- Defines the authorization scope.

# Bilrost branch



- Almost the same as a git branch
- Is the duplication of files so that modifications can happen in parallel along both branches. Modifications are then isolated.
- Is identified by a name

# Bilrost workspace



- Project representation in user file system.
- Contains assets and resources within a target branch, only the ones user subscribed to.
- Branches are switchable.
- Bilrost manages a favorite list of workspace pointers for being quickly findable within user's disk from its API.

# Bilrost subscription



- Is bound to a workspace
- Tells which content user want to work with
- Sparse checkouts resources defined by an asset, a namespace or a search query. Ex:
  - /assets/feline/cat
  - /assets/feline
  - /assets/feline?q=black
- Operated from latest version of target branch

# Bilrost stage



- Includes a list of assets to scope within next commit.
- Only created, modified, removed, renamed assets/dependencies are stagable.

# Bilrost commit



- Pushes the changes in staged assets to the project.
- Requires a comment to describe the recorded changes.

# Bilrost deploy



# .bilrost.json

```
{
  "origins": [
    {
      "organization": "org_name",
      "project_name": "game-assets",
      "branch": "release_branch",
      "deploys": [
        {
          "ref": "/assets/NDA",
          "dest": "GameExample/Assets"
        },
        {
          "ref": "/assets/Public"
        }
      ]
    }
  ]
}
```

- Downloads asset content to user's disk
- Is directed by JSON definition file
- Assets are found with:
  - ◆ Reference uri
  - ◆ Project name
  - ◆ Branch name
- Two modes: "move" and "link"



# Additional features



- Resources are stored only once in the cloud storage per organization.
- Local cache
- Search API for discovering assets and its files



# Bilrost workflow

## 03

How does Bob use Bilrost for versioning  
a new Game asset?

Pre-requirements:

- Bob and Alice are authenticated to the same project,  
“org\_name/Game-assets”

# Demo



1/ Bob commits its first asset in production branch

2/ Alice modifies the asset in maintenance branch



# Benefits

## 04

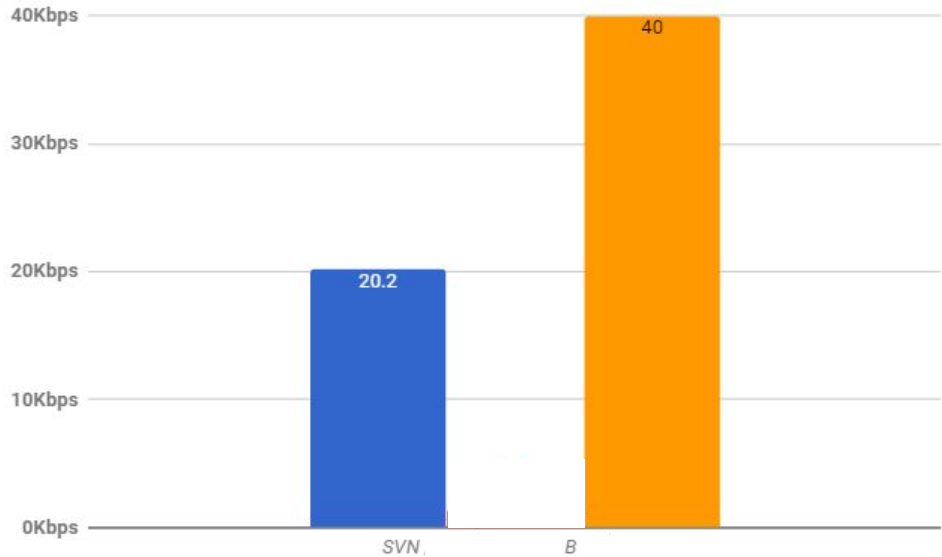
Why is bilrost much better?

- A Version control system coupled with an asset manager for handling dependencies
- Easy deployment
- Support Large Files

# Better performances



Download faster! Tests done with 180 MB assets



Version control system	SVN	Bilrost with CF
Down speed (Mbps)	69	84
Duration (s)	89	45
Duration (min:s)	1:56	0:45
Speed (Mbps)	2.02	4
Speed (Kbps)	20.2	40

# Scalable



- Supports Amazon S3® for now, but could be hooked to any cloud storages. eg. Google Cloud™, Windows Azure™, IPFS®...
- Optional optimized CDN associated with cloud storage solution
- Supports undefined large repositories
- Better cost management
- GUI easily hookable with API

# Conclusion



- VCS used in game industry not designed originally for game development... Bilrost does this!
- Validating/converting assets
- Platform for sharing assets using git
- Bilrost is a tool to create a build system on top of it. When a resource changes we can easily know which assets are affected and rebuild them.

# References



<https://gamedev.stackexchange.com/questions/480/version-control-for-game-development-issues-and-solutions?rq=1>

<https://softwareengineering.stackexchange.com/questions/136079/are-there-any-statistics-that-show-the-popularity-of-git-versus-svn>

<https://www.atlassian.com/git/tutorials/comparing-workflows#!workflow-gitflow>





# Git LFS

- Branch support- Every addition can be controlled before being merged within production game iteration!
- No sparse checkout: Impossible to download individually game asset without fetching the whole project.
- Support for large binary files (>200MB) is recent and expensive. \$5 every 50GB allotted bandwidth. Every byte checked in/out are taken out from available bandwidth.
- No locking system. There is no way to prevent users to modify a file that is already being touched by someone else.



# SVN

- Gets slower bigger the project becomes!
- Hard to maintain release branch. If one asset changes, how to revise this only asset?
- Branching means to duplicate all resources in local/remote storage. Besides rising cloud storage costs, this slows even more operations.
- Locking system is supported but not automatically operated when collaborating on the same asset.
- Sparse checkout is supported.
- Known problems:
  - Artists check in corrupted files.
  - Release sensitive data.
  - Mixed-in props and source code.



---

# Perforce®

- Expensive
- Branch support
- No sparse checkout
- Git is just better...