UNIVERSITY OF MISKOLC,
FACULTY OF MECHANICAL ENGINEERING
AND INFORMATION TECHNOLOGY

Software Technology Lab

GEIAL316-B2a

Version control

Tamás Tompa, PhD
assistant professor
Department of Information Technology

Miskolc, 2026

Version control

- **Version control** is a set of procedures that allow **to manage variants (versions) of a dataset**
  - **Dataset:**
    - **source code, documents, etc.**
    - in the case of software, it primarily means the storage **of modifications made to the source code during the life cycle of the software**
    - **but it's not just the source code** that can be versioned!

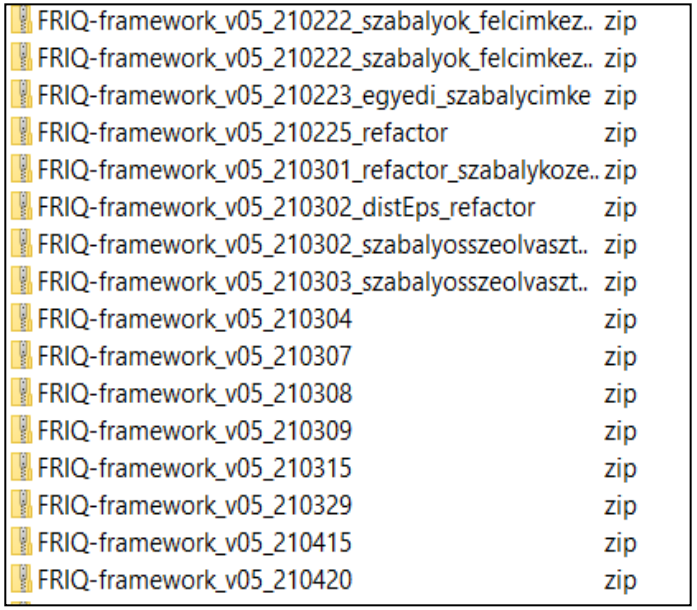
Version control

- During development, **the source code goes through many iterations**
  - in case of a problem, **should revert to an earlier version of the source code**
- It is a great help **in teamwork** and in the case of individual software development
  - sharing the codebase, the current status of everyone
- Names: Revision Control, Version Control System (VCS), Source Control

Version control

- **Simple "version control"**
  - All changes were saved in a separate folder
    - feasible but difficult
    - time-consuming, inefficient
    - not transparent
    - and if these are also lost?
    - → strongly not recommended..

How not... →



- **Solution: using version control systems**
  - e.g. Git, Mercurial, SVN, etc.

Basic concepts

- **Repository: storage**, repo for short
  - can use local and remote repositories, such remote repositories as Github, Bitbucket, etc.
  - storages contain changes and the current state of the project as snapshots

- **Working copy: a copy of a piece of code** that the developer is currently working on on his own machine

- **Push:** upload data/commits to the remote repo
- **Pull: changes download** from **the remote repo**
- **Diff/Change:** display **changes** between two files

Basic concepts

- **Commit:** it is a **concise comment** that summarizes **what modification has been made**
  - changes to the code can be validated in the form of commits
  - it is advisable to commit it after any minor modifications
    - not per 1000 lines, but not per character either...

How not...:

pici css móka committed 3 months ago	elfelejtettem ezt a portot committed 4 months ago
css committed 3 months ago	amúgy sem működik... committed 4 months ago
pár apróbb ság amit kért committed 3 months ago	d(*.*)b committed 6 months ago
little css committed 4 months ago	szeretek kétszer dolgozni committed 6 months ago
egy újabb committed 4 months ago	még egy fix committed 6 months ago
elkeseredett próbálkozás committed 4 months ago	fix the fix committed 6 months ago
ismét committed 4 months ago	

Basic concepts

- **Commit types (scope)**
  - **Build** - changes to the build process
  - **changes related to the CI -** (Continuous Integration) process
  - **Chore** - build process changes
  - **Docs** - documentation changes
  - **feat** - new feature
  - **Fixed** - bug fix
  - **perf** - code change that improves the performance
  - **refactor** - a change in code that neither fixes a bug nor adds a new function, only refactoring
  - **Revert** - withdrawals
  - **style** - space, formatting, missing semicolons, etc.
  - **Test** - add tests

Basic concepts

- **Commit rules**
  - "**scope: message**" format
  - english, present tense
  - not longer than 50 characters
  - short, to the point
  - **what why**, and not what how

  - Examples:
    - `build: Update webpack configuration`
    - `ci: Add GitHub Actions workflow for automated testing`
    - `chore: Update dependencies using npm audit fix`
    - `feat: Add user authentication feature`
    - `fix: Fix issue with form validation not working on Safari`
    - `revert: Revert "feat: Add user authentication feature„`
    - `test: Add unit tests for login component`

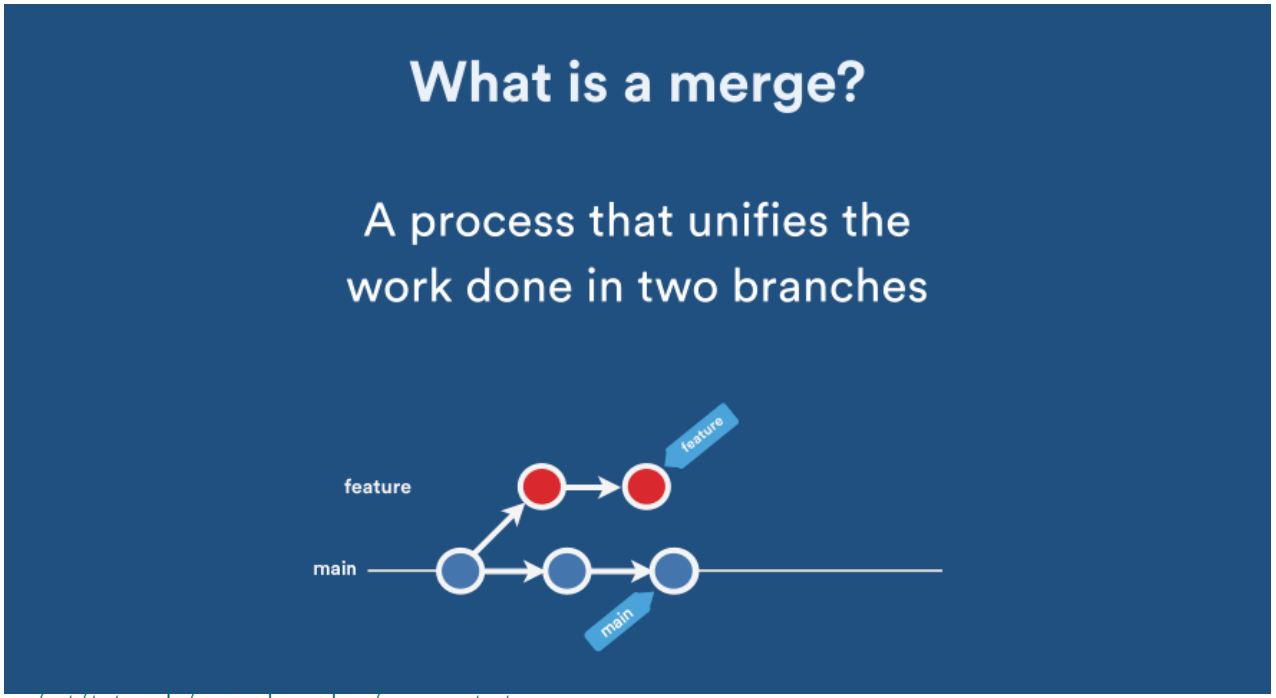
Basic concepts

- **Revision:** version
- **Checkout:** branch switch, commit restore
- **Head:** indicates the most recent commit (version), the top of the current branch
- **Snapshot:** the current state of files at a given point in time
- **Clone:** download the content of the repo (locally on the hard drive)
  - to a specified folder

Basic concepts

- **Merge:** merging, merging branches
  - new commit (changes to both branches)
  - the history of branches will remain
  - show the conflicts

(animation)

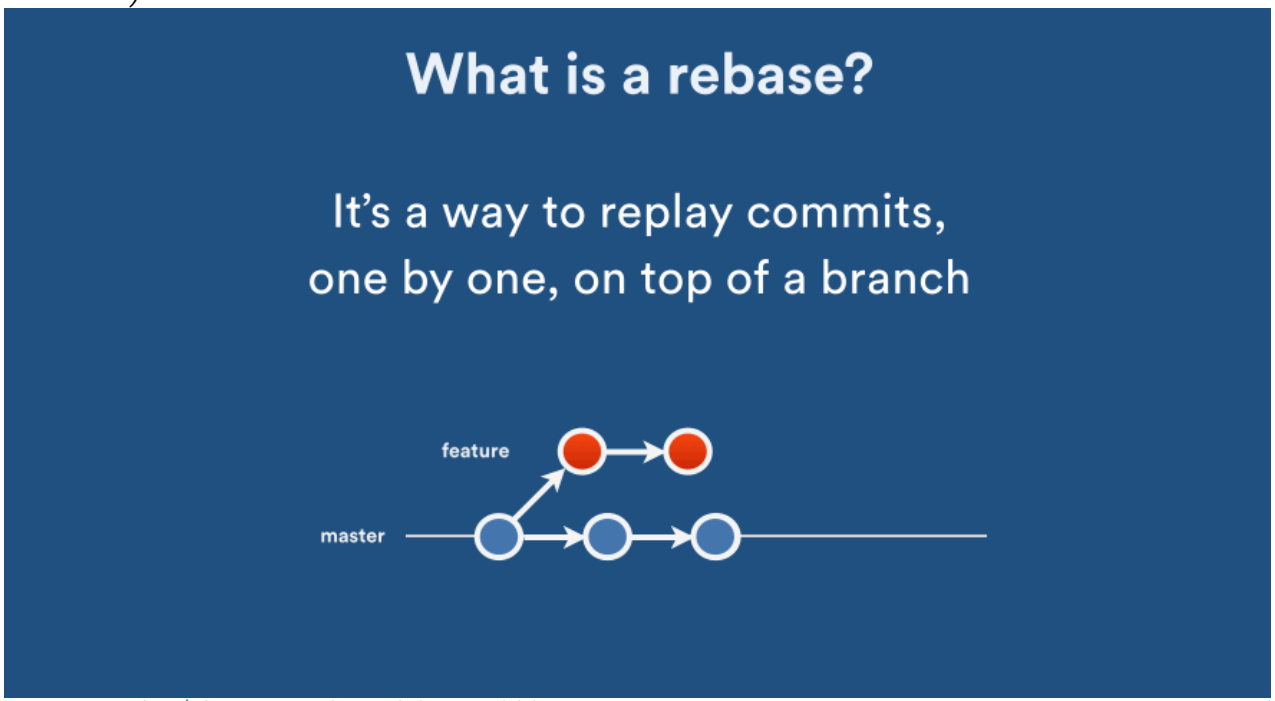


Forrás: https://www.atlassian.com/git/tutorials/using-branches/merge-strategy

Basic concepts

- **Rebase:** validate commits in the current branch to another branch
  - commits to the other branch
  - suspends this process in case of conflict (manual resolution)

(animation)



Source: https://medium.com/pranayaggarwal25/git-merge-rebase-d8b91825bbb1

Basic concepts

- **Merge vs. Rebase:**
  - **The merge:**
    - creates a new commit in which it merges the 2 branches
    - all history remains intact, so the original fork and merge process can be seen in the commit story
    - preserves the entire history of development
    - if we want to preserve the original development story and it is important to see where the changes branched out
  - **The rebase:**
    - moves the commits of the development branch to the latest state of another branch without rewriting
    - does not create a separate merge commit
    - if you want a clean, linear commit story
  - Important: conflicts can arise, be careful!

Alapfogalmak

- **Why was rebase introduced?**
  - the merge alone is sufficient from a functional point of view, but it is not enough to maintain a good version history
  - the rebase was introduced to make **the commit history more understandable, linear, and easier to manage**
  - Merge has side effects:
    - the history becomes full of branches
    - many "technical" merge commits appear
    - it becomes difficult to trace what was done and why
    - git log → difficult to read
    - code review → more difficult
    - the commit history gets "polluted"

Alapfogalmak

- **Why was rebase introduced?**
  - **rebase does not merge branches; instead, it "replays" the commits onto another branch**

Merge

```
A---B---C-------M
     \           /
      D----E----F
```

Rebase

```
A---B---C---D'---E'---F'
```

feat: login form
Merge branch 'main' into feature
feat: validation
Merge branch 'main' into feature
fix: edge case
Merge branch 'main' into feature

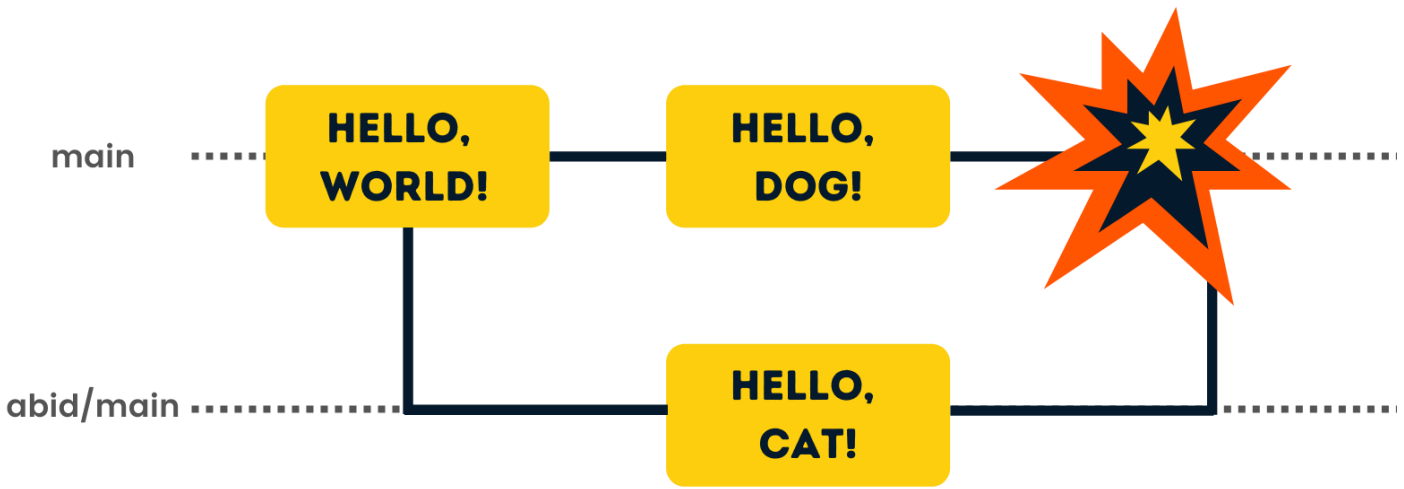
feat: login form
feat: validation
fix: edge case

Alapfogalmak

Case	Merge	Rebase
Public branch (main, develop)	✅	❌
Own feature branch	✅	✅
Clear commit-history required	❌	✅
„I want to play it safe”	✅	❌

Basic concepts

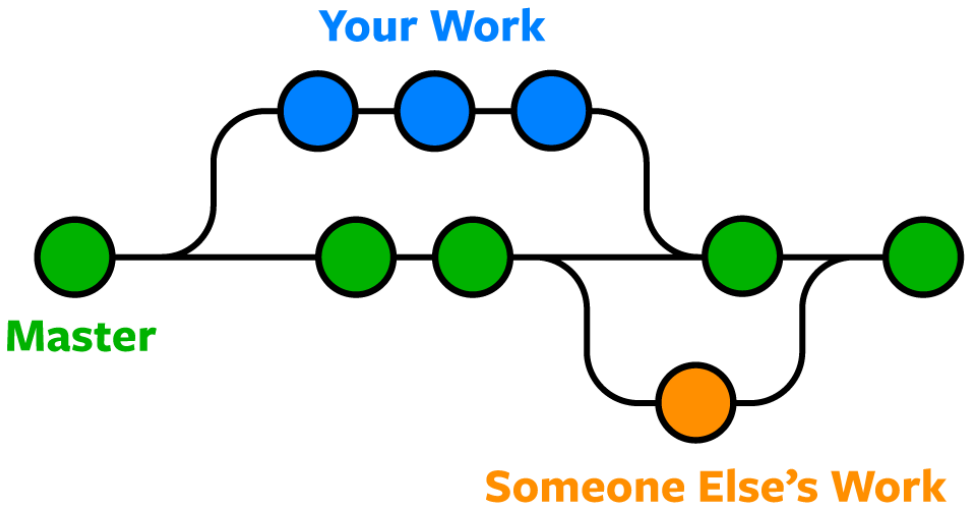
- **Conflict:** a phenomenon that occurs during the merging of development branches (branches), modifications made to the same part of the data by several actors, cannot be merged automatically
  - if it occurs → should be resolved
  - resolution: conflict by flags manually or in IDE



Forrás: https://www.datacamp.com/tutorial/how-to-resolve-merge-conflicts-in-git-tutorial

Basic concepts

- **Branch:** development branch, development should not take place only on a "thread"
  - the most common branches: master (main), develop, feature, release, hotfix, etc.



  - If possible, spend as little time as possible on a branch
    - until then, a lot of things happen at the master → a lot of conflicts
    - → small (1-day) tasks/tickets

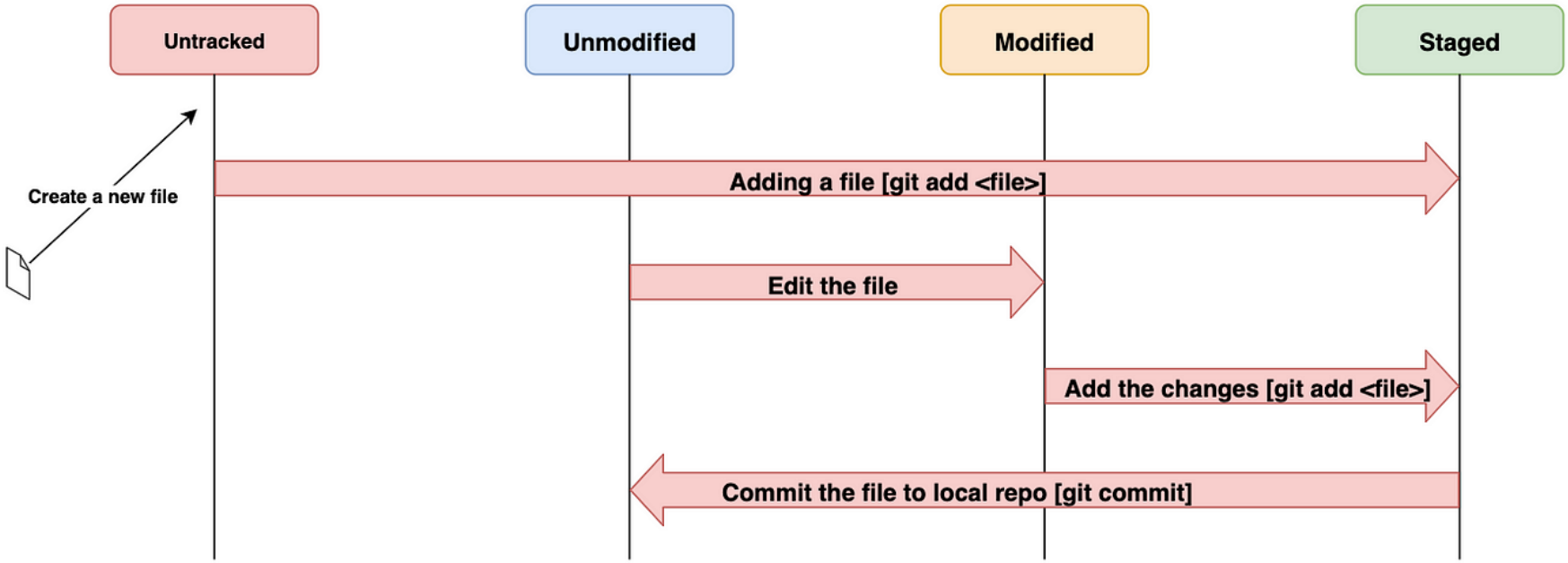
Source: https://www.nobledesktop.com/image/gitresources/git-branches-merge.png

File statuses

- **Untracked:** files whose changes are not tracked

- **Tracked:** files whose changes are tracked
  - modified: the file has been modified since the last commit
  - unmodified: the file has not been modified since the last commit
  - staged: the file has changed since the last commit, its change will be stored during the next commit

File statuses

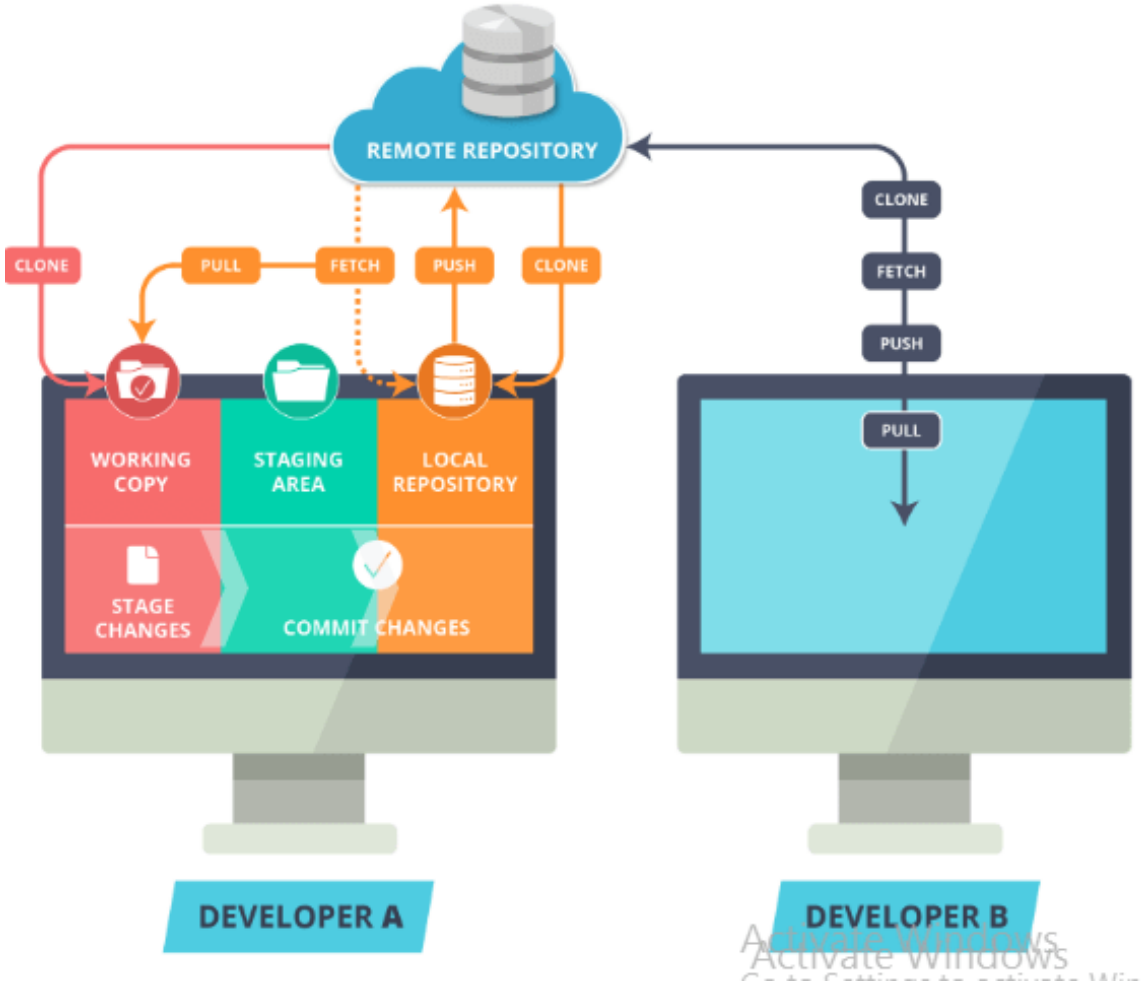


Source: https://miro.medium.com/v2/resize:fit:1100/format:webp/1*VlEZHRPT7vumT_505jLaHg.png

Types of repositories

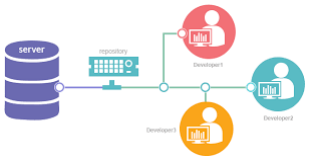
- **local:** changes to the local file system, local repo
- **remote:** changes to the remote server, remote repo

- **public:** anyone can access it
- **private:** only authorized users can access

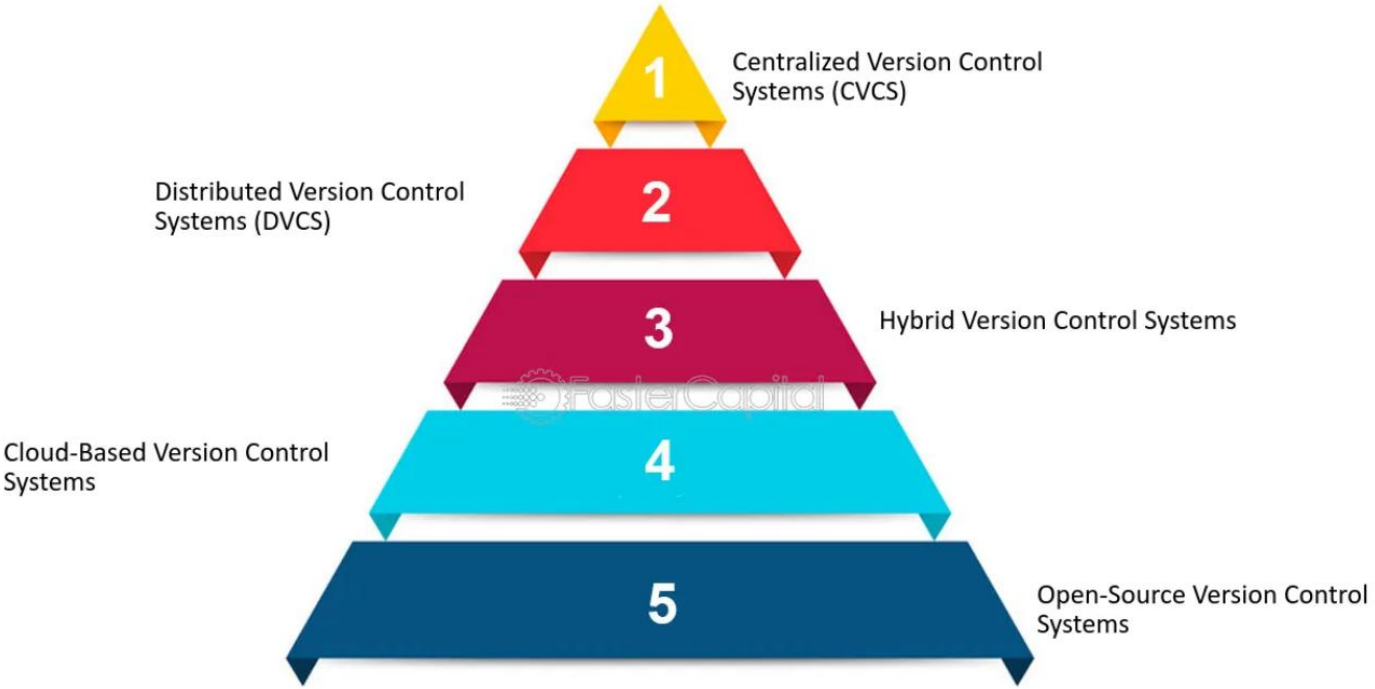


Forrás: https://www.c-sharpcorner.com/article/git-and-github-version-control-local-and-remote-repository/Images/Git%20And%20Github%20Version%20Control10.png

Version Control System Types

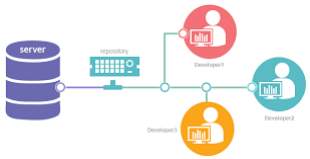


- **Basically, there are 2 types**
  - **Centralized (CVCS)**
  - **Decentralized system (DVCS)**



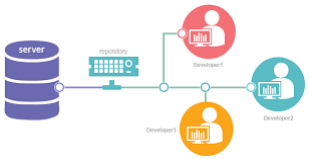
Forrás: https://fastercapital.com/topics/types-of-version-control-systems.html

Version Control System Types

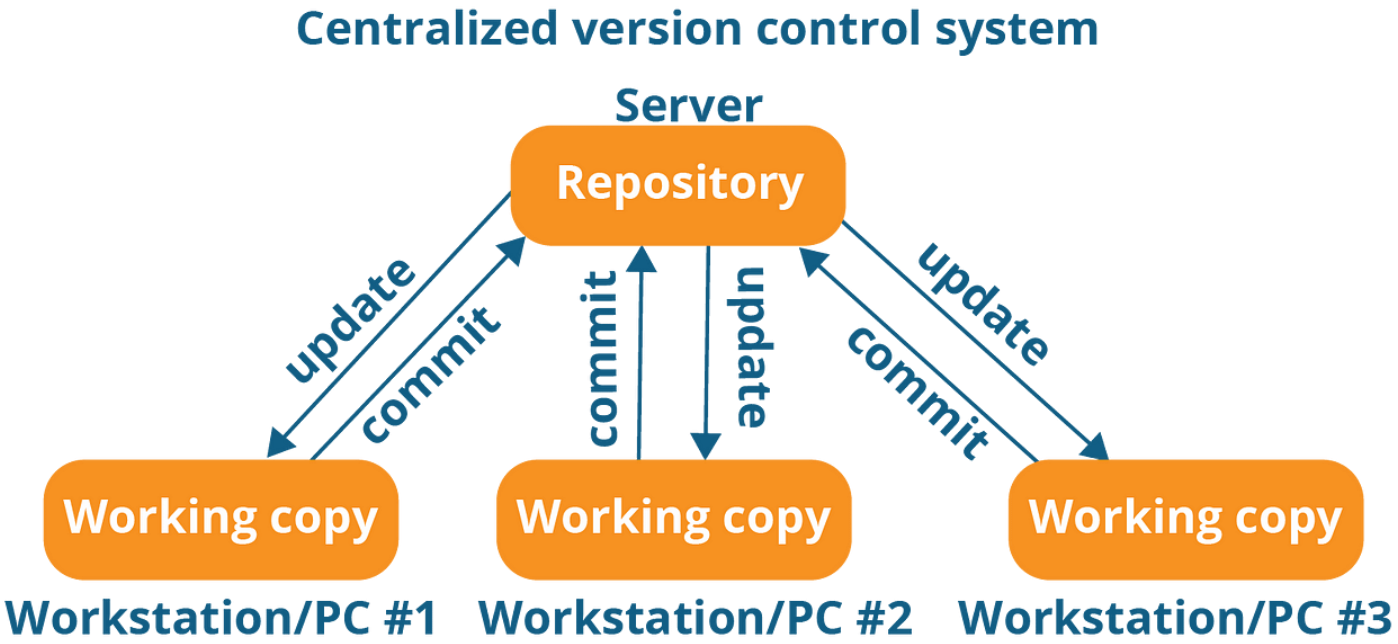


- **Centralized (CVCS)**
  - **common central repo**
    - all developers are directly connected to this
  - unified version control
    - developers always work on the central repo version
  - **all operations on the server**
    - there is always a connection to the server
    - after every commit, an update is needed for the changes to appear for everyone
    - all changes must be uploaded to the central repo, so it is strongly tied to the availability of the central server

Version Control System Types

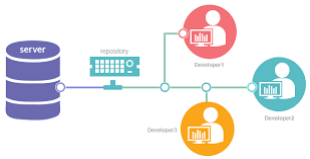


- **Centralized (CVCS)**



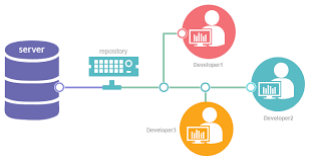
Forrás: https://faun.pub/centralized-vs-distributed-version-control-systems-a135091299f0

Version Control System Types

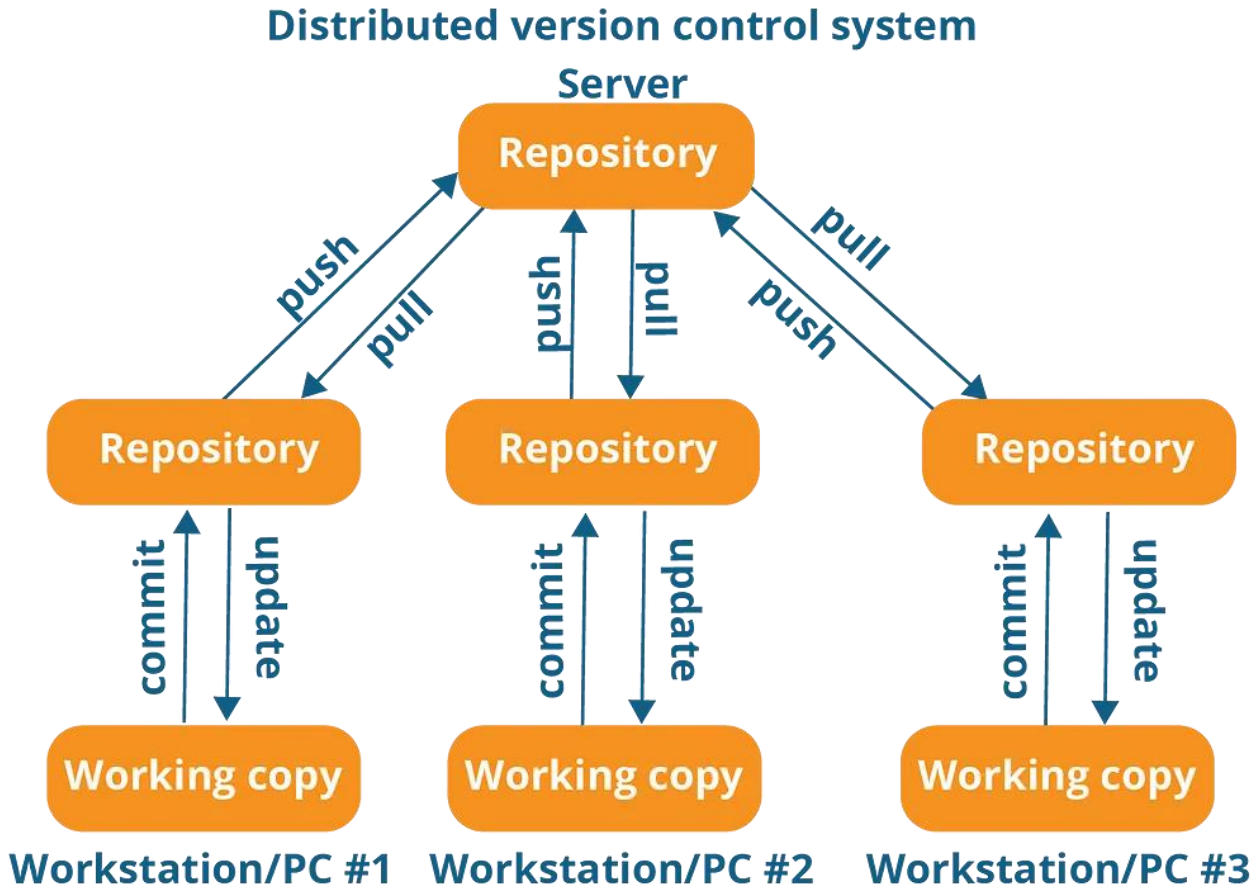


- **Decentralized system (DVCS)**
  - **no central storage**
    - everyone has their own local repo
    - user machines in containers (local repo)
    - but there can also be a remote repo next to it
    - developers work in their own local repository and only sync changes to remote repositories when they're ready
  - **no need to communicate with a central server**
    - fast
    - not dependent on server access → more flexibility

Version Control System Types

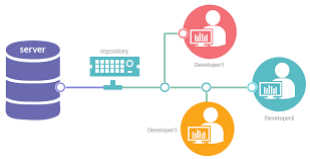


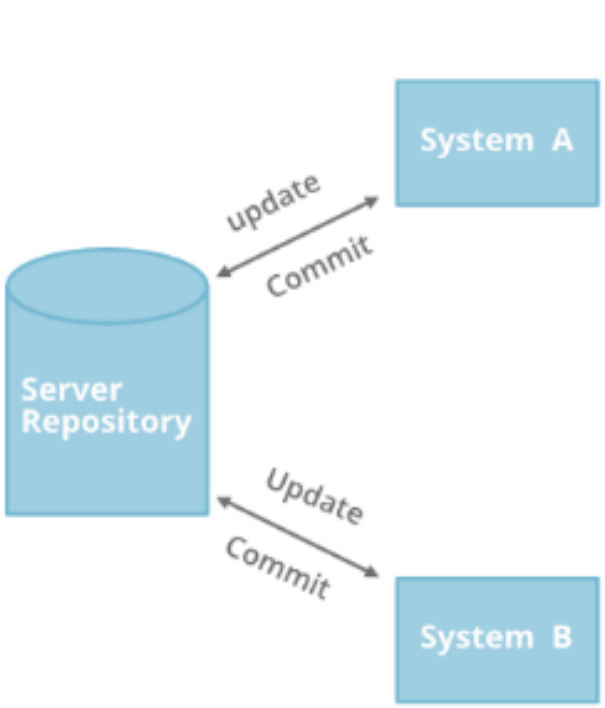
- **Decentralized system (DVCS)**

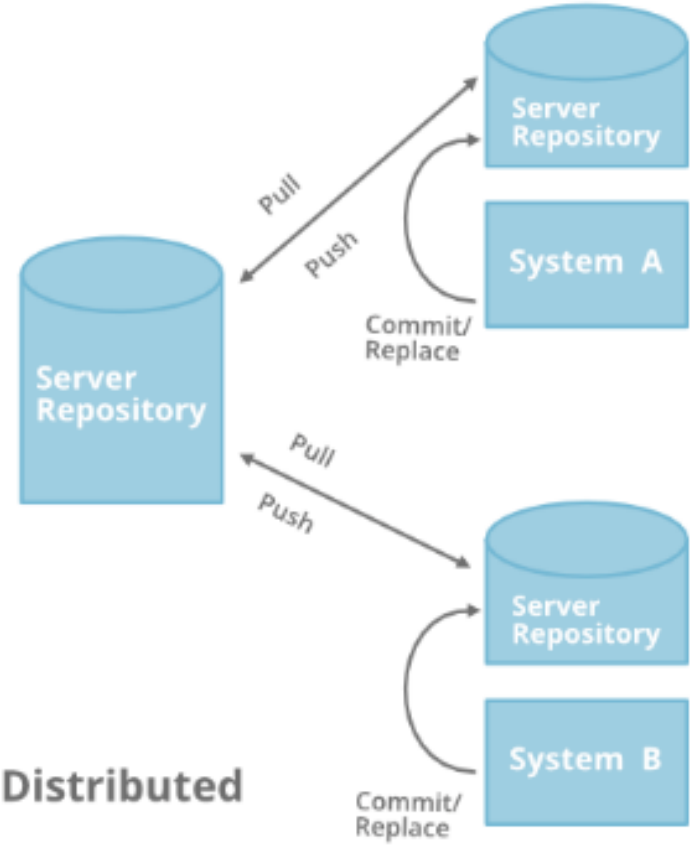


Forrás: https://faun.pub/centralized-vs-distributed-version-control-systems-a135091299f0

Version Control System Types

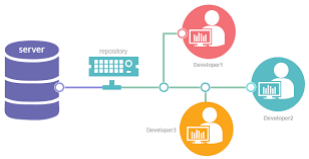




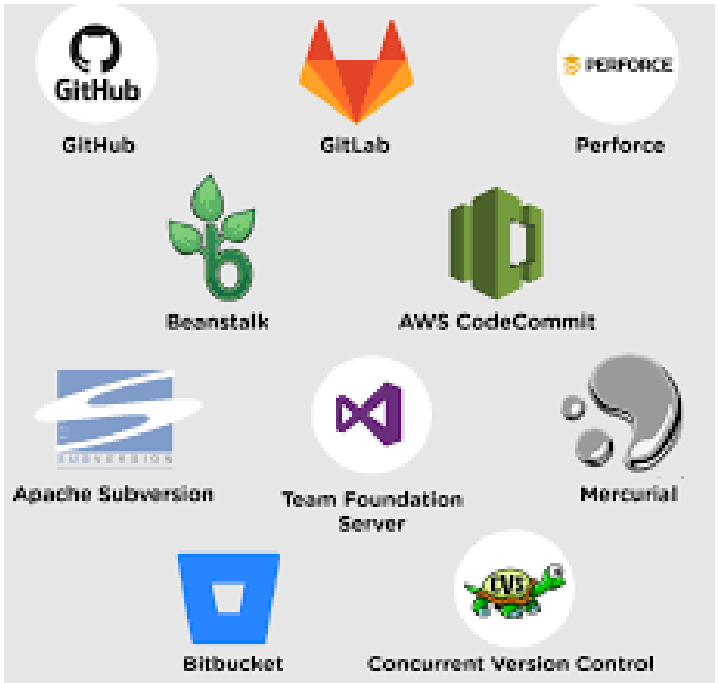


Source: *https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190820174942/CVCS-vs-DVCS.png*

Version Control Systems



- **Basically, 3 larger:**
  - **Git**
  - **Mercurial**
  - **SVN**



Source: https://bacsoftwareconsulting.com/general/top-free-version-control-systems/

Git

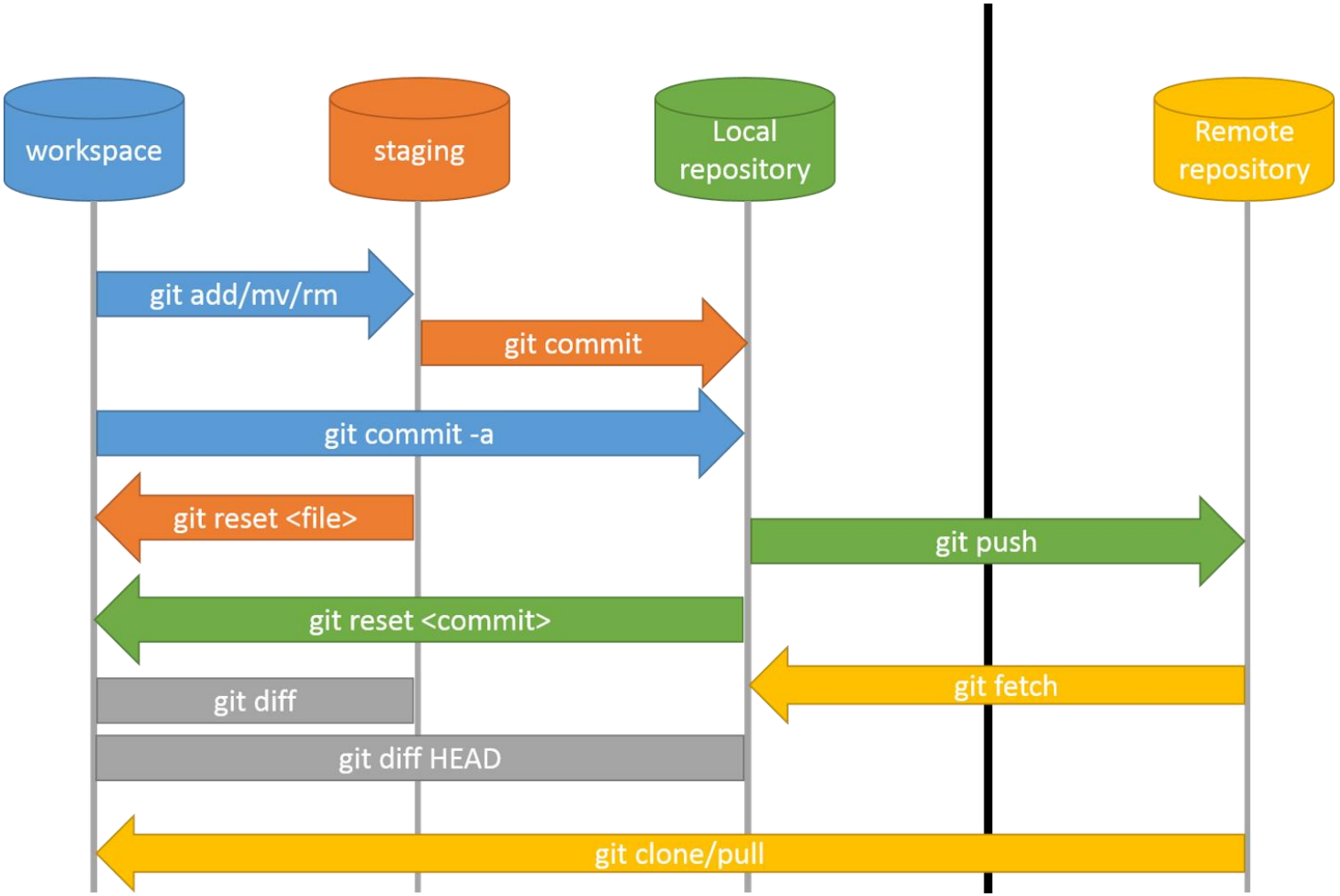
- Distributed version control (DVCS)
  - each developer has a full copy of the repo
- fast because most operations are done locally
- powerful branching and merging, possibility of parallel development
- GitHub, GitLab, Bitbucket support
- command-line and graphical clients (e.g. GitKraken, SmartGit)
- suitable for both large and small projects
- hook support

Git

Git hook-ok

- **automatic execution of a predefined command or script during a specific event (e.g. commit, push, update, etc.)**
- **Client-side (run locally on the developer's machine)**
  - e.g. checks the code style before commit
- **Server-side (run on the server, e.g. GitHub, GitLab)**
  - e.g. after a push, it sends a Slack message or starts a CI/CD pipeline
- .git/hooks/ directory
- shell script or other programming languages
- pre-hook: before the event, post-hook: after the event
- Purpose:
  - automatic checks, code formatting, statistical code analysis, initiation of CI/CD processes, automatic testing after each push, compliance with security rules, filtering out secrets and passwords before commit, etc.

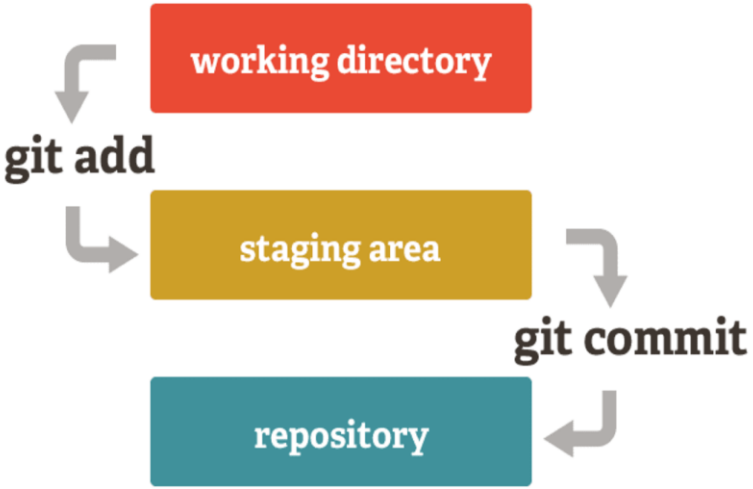
Git structure



Source: https://blog.isquaredsoftware.com/2021/01/coding-career-git-usage/

Git structure

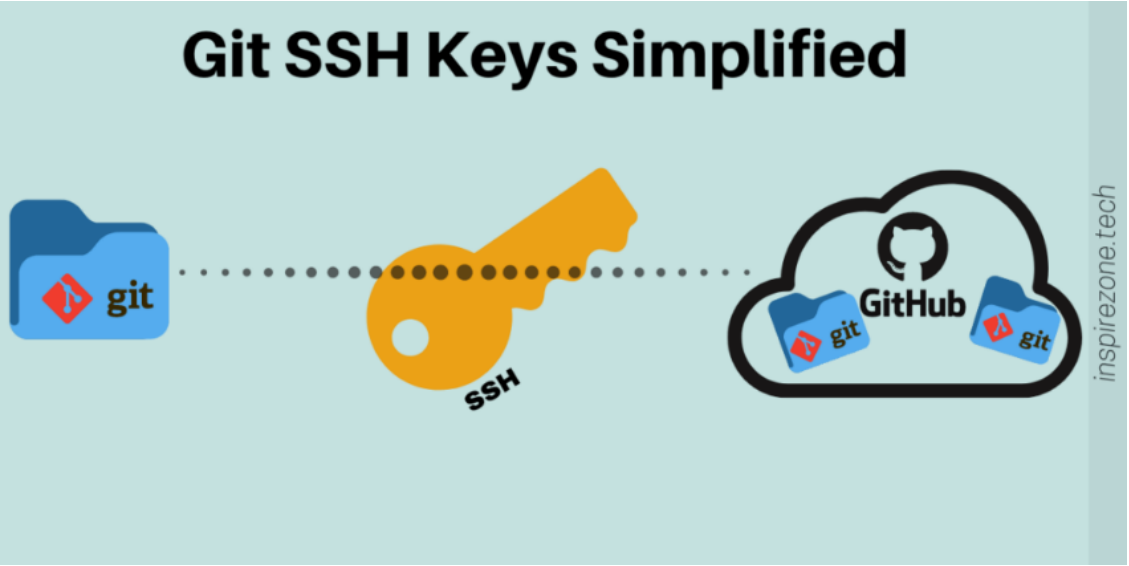
- **Working copy (/directory)**
  - the total number of files that are versioned by the system (.git)
- **Staging area**
  - changes that will be applied to next commit
- **Local repository**
  - local storage
- **Remote repository**
  - remote server



Forrás: https://res.cloudinary.com/practicaldev/image/fetch/s--Si7ksd-d--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https://cdn-images-1.medium.com/max/800/1%2AdiRLm1S5hkVoh5qeArND0Q.png

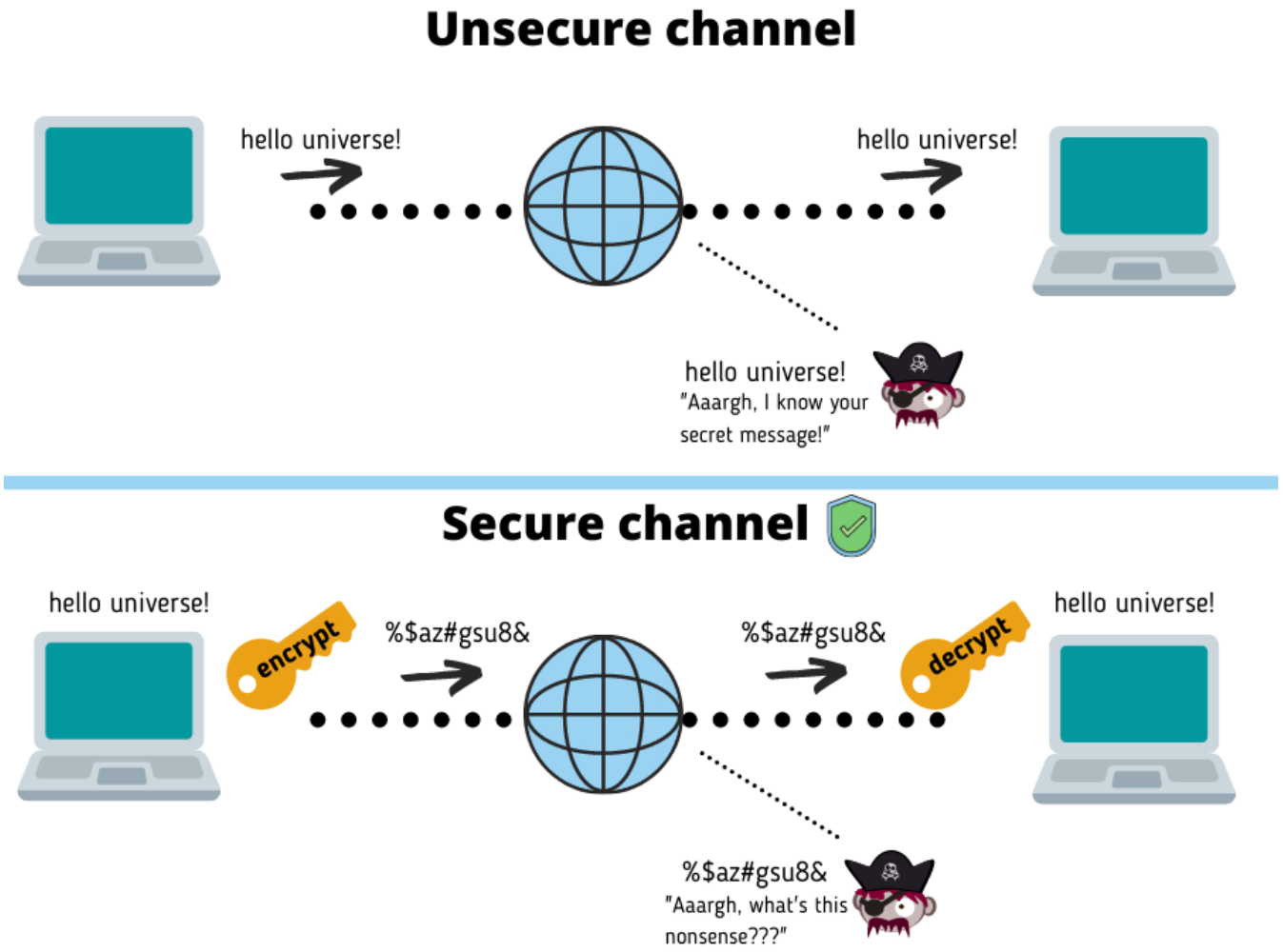
Communication between repositories

- between local repo and remote repo (Github, Bitbucket, etc.)
  - communication only required for synchronization
- HTTPS or SSH (both encrypted)
  - `Git Clone` https://szolgaltato/felhasznalonev/repo-neve.git
  - Generate SSH keys: `ssh-keygen -t rsa`



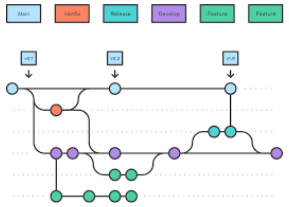
Source: https://inspirezone.tech/git-ssh-keys-simplified/

Communication between repositories



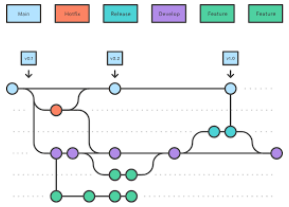
Source: https://inspirezone.tech/git-ssh-keys-simplified/

Gitflow



- Workflow model
  - Vincent Driessen, 2010, blog post
- defines a structured and **well-defined workflow**
- it is based on the **use of different types of branches**, which represent different stages of development
  - Master
  - Develop
  - Feature
  - Release
  - Hotfix

Gitflow

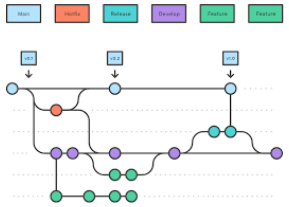


- **Master branch:** this branch contains **stable versions** that are ready for release

- **Develop branch: the main branch of development** in which active development work takes place. New features and improvements are developed on the develop branch

- **Feature branches:** each **new feature or development** is developed on a separate feature branch from the develop branch. When the feature is complete, the feature branch is returned to the develop branch

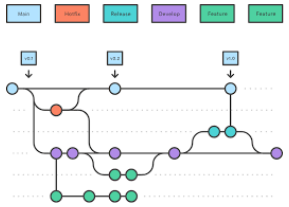
Gitflow

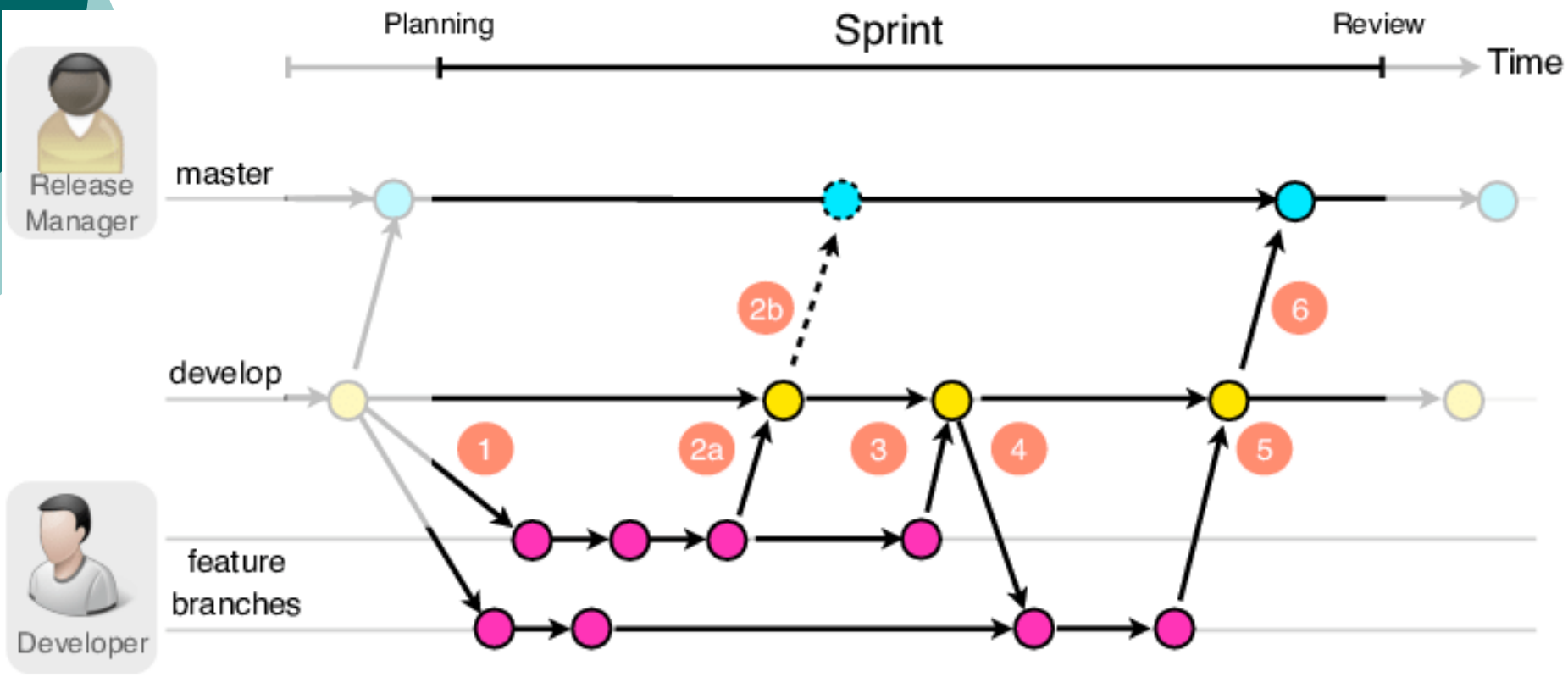


- **Release branches:** when a **new version is ready** for release, a release branch is created from the develop branch, where the final refinements and fixes for the release are made. The release branch will eventually be merged with the master branch, and the release will be created

- **Hotfix branches:** when **sudden and urgent fixes** are needed in an already released version, a hotfix branch is created from the master branch. After completing the fix, this will be returned to both the master and develop branches

Gitflow





Source of the figure:
Krusche, Stephan & Alperowitz, Lukas. (2014). Introduction of Continuous Delivery in Multi-Customer Project Courses. 36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings. 10.1145/2591062.2591163.

Mercurial

- Fully distributed: each developer has a full repo copy
- Simple set of commands: e.g. hg commit, hg pull, hg push
- Internal data management: it doesn't use hash-based object storage like Git, but instead uses a more stable storage model that prevents corrupt repositories
- Directly supports file-level changes: stores file-level revisions, while Git takes snapshots of entire trees
- It was supported by multiple development platforms: Bitbucket used to natively support Mercurial, but discontinued it in 2020
- Hooks and extensions: supports customizable hooks and plugins, but with fewer options than Git

SVN

- Centralized Version Control (CVCS)
- Simpler model than distributed systems
  - not everyone has a full copy
- Some files can be downloaded separately, you don't have to clone the whole repo
- Better for handling large binaries than Git
- No advanced branch management
  - branching, merging nehezebb
- Use in decline
- It was mainly used in older enterprise systems and game development (e.g. Unreal Engine)

VCSs Comparison

	Git	Mercurial	SVN
Type	Distributed (DVCS)	Distributed (DVCS)	Centralized (CVCS)
Branching, merging	Advanced	Good	Limited
How To Use	Full repo available offline	Full repo available offline	You need a central server
Data storage	Local and remote	Local and remote	Central Server
Field of application	Large projects, open source, startups	Simplified improvements	Old company projects, game development

References

- *Dr. Péter Mileff, Software Development, Version Control, Version Control Systems, note*
  - *https://users.iit.uni-miskolc.hu/~mileff/szf/Verziokezeles_V5.pdf*
- *Mark Groves: Introducing Gitversion control*
- *https://www.atlassian.com/git/tutorials*
- *https://git-scm.com/docs/gittutorial*

Thank you for your attention!