Software Engineering

Prof. Dr. Christoph Schober

Department of Applied Computer Science,
Deggendorf Institute of Technology

christoph.schober@th-deg.de

Version Control Systems

Version Control Systems

1. Fundamentals

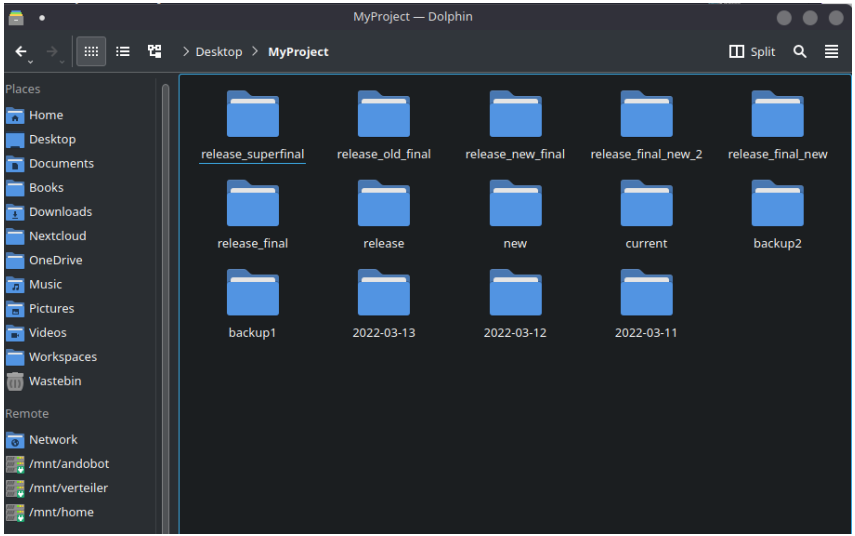
Problem: Managing code by hand without well-defined processes and proper tools is impractical (actually nearly impossible) in practice:

- During hot phases of any project, files[1] are updated with high frequency.
- Projects are handled in teams, i.e., collaborators require access to the current version of all files and distribute changes to a file in a timely manner.
- Multiple versions of the code base need to be maintained, e.g., the version shipped to the customer or the version from ”yesterday”.

[1] (e.g., source code, configuration, etc.)

Simple approaches to tackle the problem (not recommended):

- Sending files around via E-Mail.
- Common folder on a network drive.
- Common folder on a cloud storage (e.g., Dropbox).


MyProject — Dolphin
Desktop
MyProject
Split
Places
Home
Desktop
Documents
Books
Downloads
Nextcloud
OneDrive
Music
Pictures
Videos
Workspaces
Wastebin
Remote
Network
/mnt/andobot
/mnt/verteiler
/mnt/home
release_superfinal
release_old_final
release_new_final
release_final_new_2
release_final_new
release_final
release
new
current
backup2
backup1
2022-03-13
2022-03-12
2022-03-11

Result → Chaos: Files will be lost, time will be wasted

- *Increased risk of errors:* Programmers may make changes to the code that cause unintended consequences, which are hard to track down and fix.
- *No version control:* Keeping track of which version is the most up-to-date or which features were introduced in which version is challenging.
- *Limited collaboration*: Multiple programmers working on the same codebase may lead to potential conflicts and overwriting of each other's work.
- *No safety-net*: Accidental deletions, hardware failures, or other issues can cause in permanent loss of code.

Version Control Systems

2. Definition

Version Control System (VCS)

A software tool that helps programmers to manage changes to their codebase over time, allowing them to collaborate more efficiently.

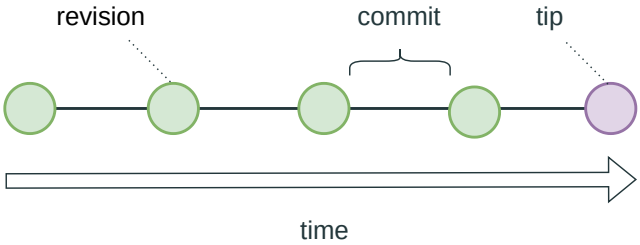
Core features:

- Records changes to a file, including who made the change, the date and time when the change was performed, and for what reason the change was made.
- Allows reverting files to a previous state / version.
- Supports the independent and simultaneous development of multiple features by multiple programmers at the same time.

Terminology:

- *Repository*: An archive of the codebase that is being worked on. Besides code, it also manages resources such as documentation, notes, and others.
- *Working directory*: Local copy of the files managed by a repository.
- *Checkout*: Obtain a local working copy from a repository.
- *Commit*: Update a repository with local changes from the working directory.
- *Revision:* Controlled version of the files in a repository.

Definition

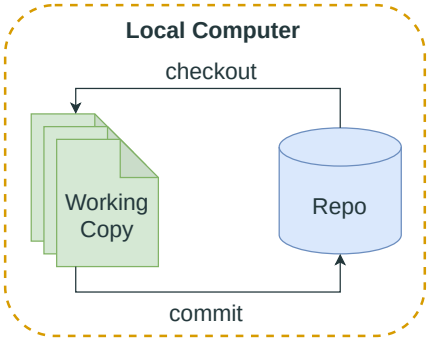


Version Control Systems

3. Architecture

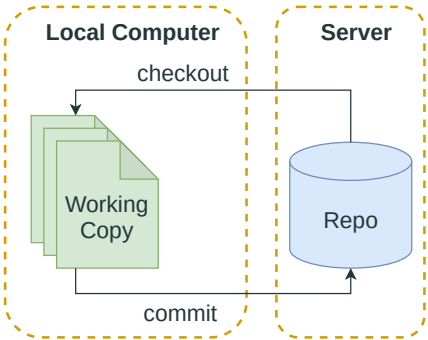
Local version control:

- Records all changes to a local repository.
- Can roll back to previous versions.
- Only works with a single computer.
- Manual synchronization between team members.

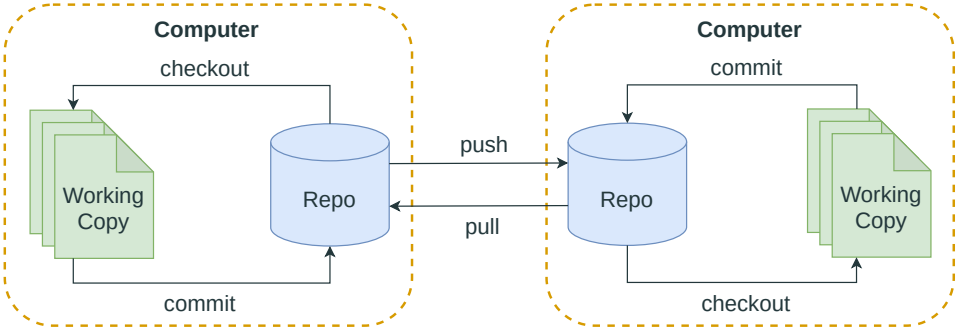


Centralized version control:

- Records all changes to a remote repository.
- Can roll back to previous versions.
- Can distribute changes to team members.
- Central server is single point of failure.
- Network connectivity is required.
- Network constitutes a bottleneck.

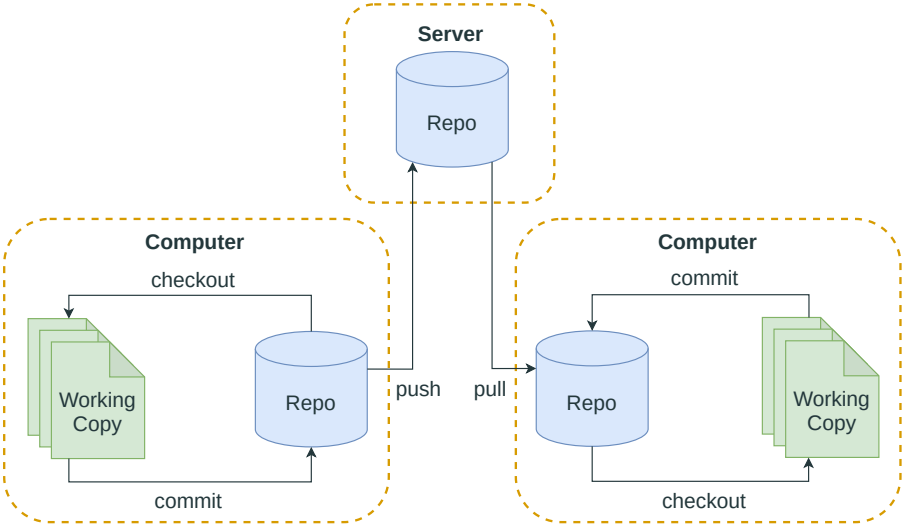


Distributed version control:



Distributed version control:

- Records all changes to a local repository.
- Can roll back to previous versions.
- Synchronizes and merges between team members.
- Arbitrary synchronization hierarchy possible.
- Central server conceptually indistinguishable from other clients.


Server
Repo
Computer
checkout
Working Copy
Repo
commit
push
pull
Computer
commit
Repo
Working Copy
checkout

Distributed version control is state-of-the-art:

- Repositories are kept local at each user.
- Repositories are synced (regularly) with the central server.

Main advantages:

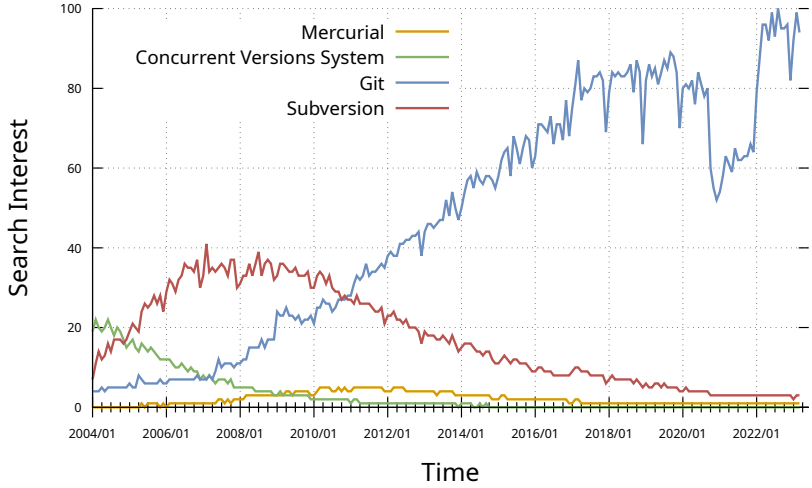
- Interactions with the local repository are very fast.
- All VCS features are available offline.
- Encourages developers to commit to VCS in a higher frequency.
- Reduced load on the central server as synchronizations are performed only when a piece of work is finished.

Version Control Systems

4. Tools

Available VCS software tools:

- *Concurrent Versions System*
  - First centralized version control system (1989)
  - Supports versioning of multiple files in file-by-file fashion.
- *Subversion*
  - Developed to address the shortcomings of CVS (2000)
  - Major centralized system
- *Git*
  - Created by Linus Torvalds for the Linux Kernel (2005).
  - Most popular open-source distributed versioning system.
- *Mercurial*
  - Developed in parallel with Git
  - Similar features



[1]Source: Google Trends

Git[2]

A free and open source **distributed version control system** designed to handle everything from small to very large projects with speed and efficiency.

Properties:

- Distributed VCS: Local repositories on each computer.
- Supports non-linear development very efficiently.
- Cryptographically secure development history.
- Available on all modern operating systems.

[2] `https://git-scm.com/`

`Git` CLI commands to interact with a local repository:

Command	Description
`git init`	Create a new local repository
`git checkout`	Obtain a working copy from the local repository
`git add`	Add a file or directory to the repository
`git rm`	Remove a file or directory from the repository
`git commit`	Store changes in the local repository
`git log`	See the commit history
`git status`	Displays the state of the staging area[3]
`git diff`	Shows changes made to the working directory
`git stash`	Temporarily shelves changes

[3] The staging area comprises the files that will be part of the next commit.

To exclude files from version control, create a file named `.gitignore`, fill it with patterns to exclude, and add it to the VCS using the `git add` command.

Example:

```
# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
*.__pycache__
```

Git Tag

Label used to mark a specific point in the Git history of a codebase, typically used to signify a release or a significant milestone.

Command:

Command	Description
`git tag`	Manages tags in the repository

Version Control Systems

5. Branching

Branch

A separate, parallel duplication of the files under version control that allows developers to make changes without affecting the main codebase.

Motivation for branching:

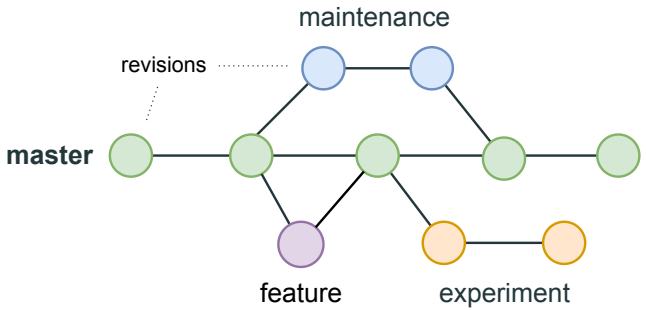
- *Separation of maintenance and development*: One developer fixes a bug and another developer implements a feature for the next version.
- *Exploring & experimenting*: One developer works on an idea that changes lots of code and another developer continues implementing features.

`Git` commands to manage branches with a local repository:

Command	Description
`git branch`	Creates a branch in the local repository. Deletes a branch when executed with the `-d` argument.

Branching practice:

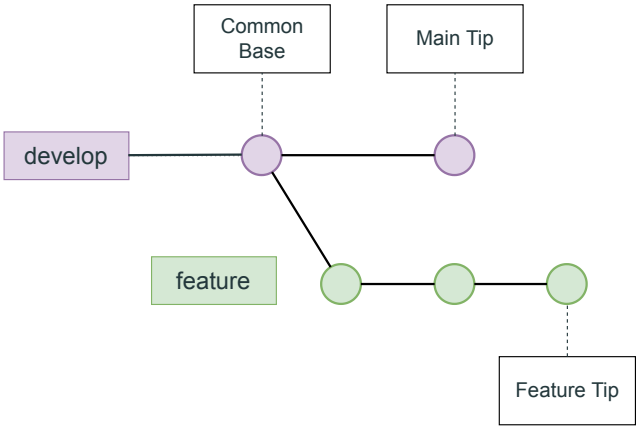
- Separate branches are created for developing features.
- Separate branches are created for hotfixes.
- Branches may be merged into master later on.



`Git` commands to integrate changes from one branch to another:

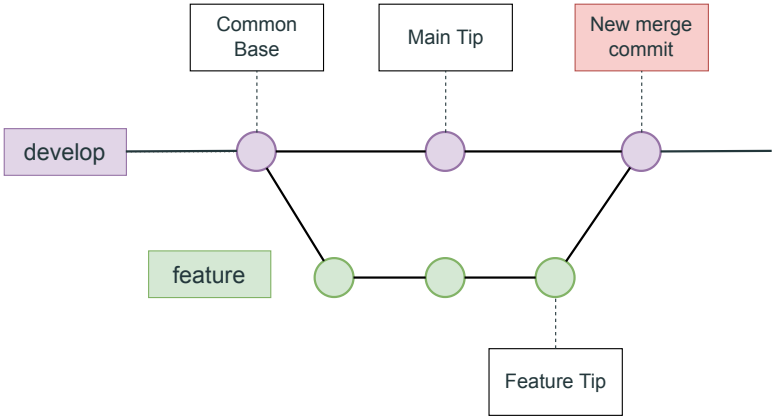
Command	Description
`git merge`	Puts a forked history back together again by creating a merge commit that contains all changes from the merged branch
`git rebase`	Rewrite commits from one branch to another. That is, when branch `A` is rebased onto branch `B`, all commits in `A` that were created after branching away are added to `B` immediately after the common base. All commits in `B` that were created after branching away are **reapplied** to `B` immediately after the commits that were added from `A`. *Note:* Reapplied commits are new commits.

Consider the following example scenario:



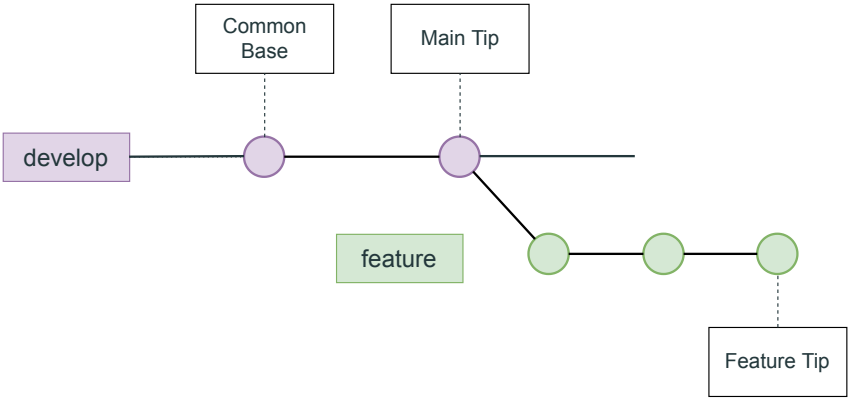
After executing (from `develop`):

```
1 git merge feature
```



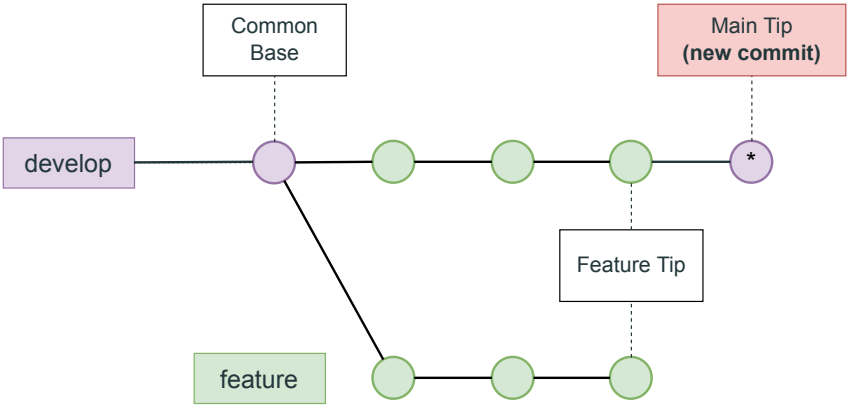
After executing (from `feature`):

```
1 git rebase develop
```



After executing (from `develop`):

```
1 git rebase feature
```



Once `rebase` is understood, an important key learning is when not to do it.

The golden rule of rebase

Never rebase public branches, i.e., branches used by other people.

Best practice:

- Use `rebase`, but use `rebase` with caution.
- Use `rebase` only to rewrite history on private branches.

Version Control Systems

6. Conflict Management

Conflict Management

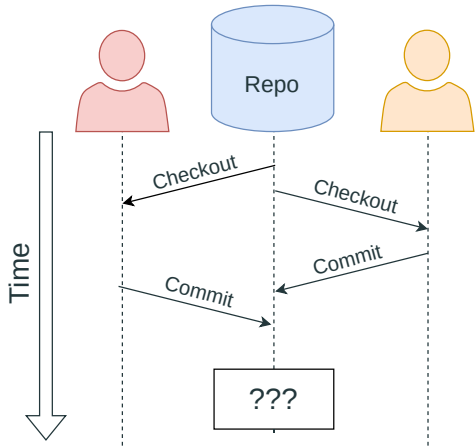
The process of resolving differences between two or more conflicting versions of the same file or code, and merging them into a single, coherent version.

Challenges:

- Some tasks can be easily partitioned, whereas some tasks cannot.
- Changes may be independent even in the same file.
- Conflicts happen nevertheless.

The Lost Update Problem:

- Two changes overlap.
- Without further precautions, the last change ”wins”.
- Previous changes are lost and can easily go unnoticed.



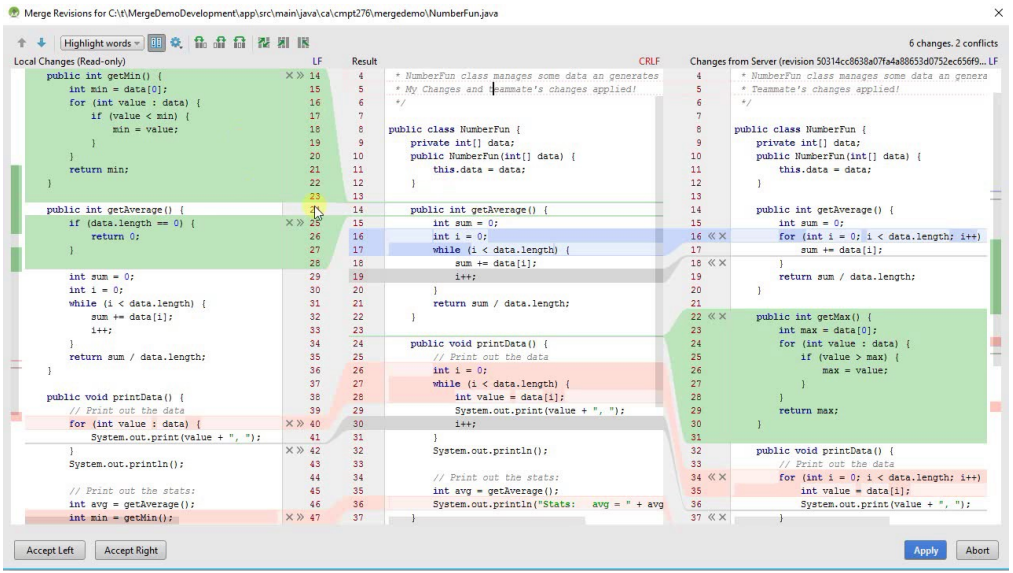
Pessimistic version control: *Lock, Modify, Unlock*

- Avoids concurrent modification
- Produces idle time when lock is not released quickly
- Collaborators rush to acquire Lock
- Potential for deadlock

Optimistic version control: *Copy, Modify, Merge*

- Allows concurrent modification
- Attempts to merge concurrent changes automatically
- If in doubt, raise error and let collaborators resolve manually
- Used in every modern VCS

Conflict Management



Version Control Systems

7. Remote Repositories

`Git` commands[4] to interact with a remote repository:

Command	Description
`git clone`	Create a local copy of a remote repository
`git remote`	Manage remote repositories
`git push`	Update remote repository with local commits
`git pull`	Obtain updates from a remote repository

[4] `https://git-scm.com/docs`

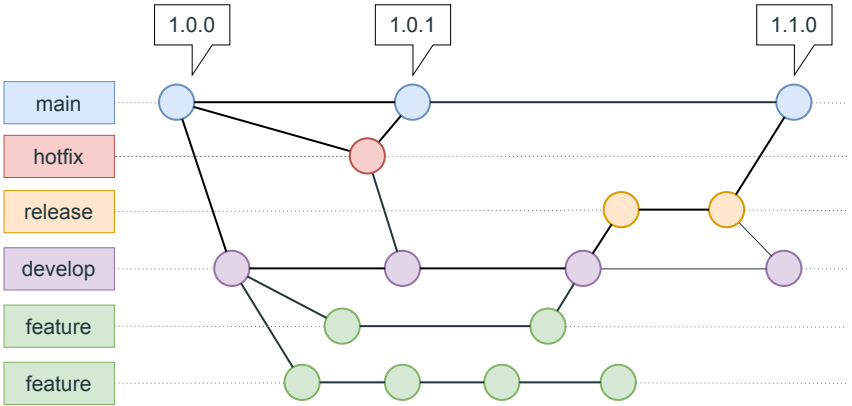
Version Control Systems

8. Branching Models

The *Gitflow* branching model works as follows:

- The **main** branch stores the official release history.
- The **develop** branch serves as integration branch for features.
- Each feature is developed in its own **feature** branch, which is branched off of `develop` and merged back to `develop`.
- Once `develop` has acquired enough features for a release, a **release** branch is forked of `develop`, and once tested (and debugged), merged into `develop` and `main`.
- To patch production releases, **hotfix** branches are forked of `main`, and as soon as the patch is ready, merged back into `main` as well as `develop`.

The *Gitflow* Workflow[5]:



[5] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Version Control Systems

9. Summary

Summary

You should have acquired the following competencies:

- Understand why we need Version Control Systems.
- Use Git as Version Control System.
- Create a working copy from a local or remote repository.
- Update a repository with local changes.
- Exclude files from version control.
- Handle multiple branches.
- Resolve merge conflicts.
- Apply the *Gitflow* workflow.