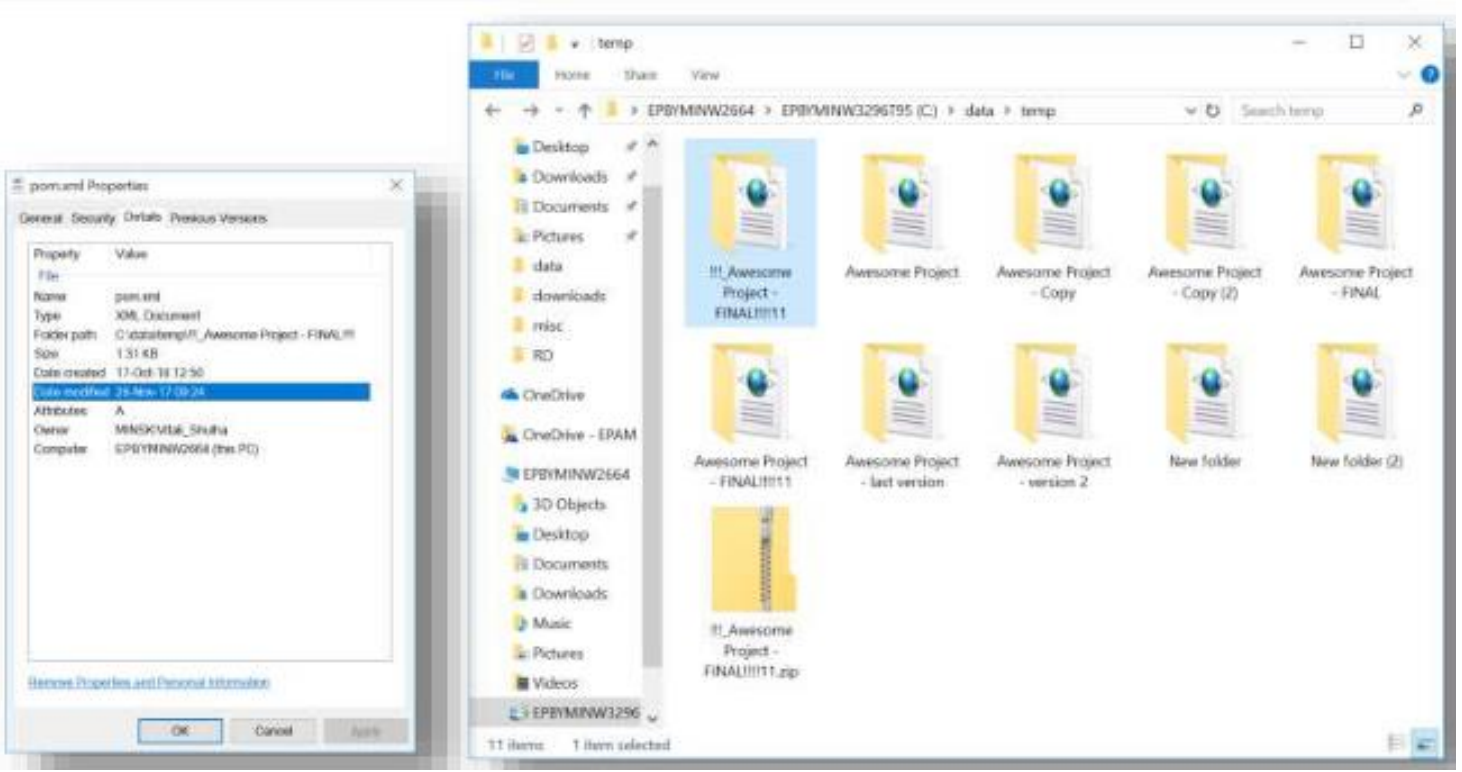
Git essentials

Stefan Zhauryd
Instructor

Module 4 Part 1
Version Control Systems

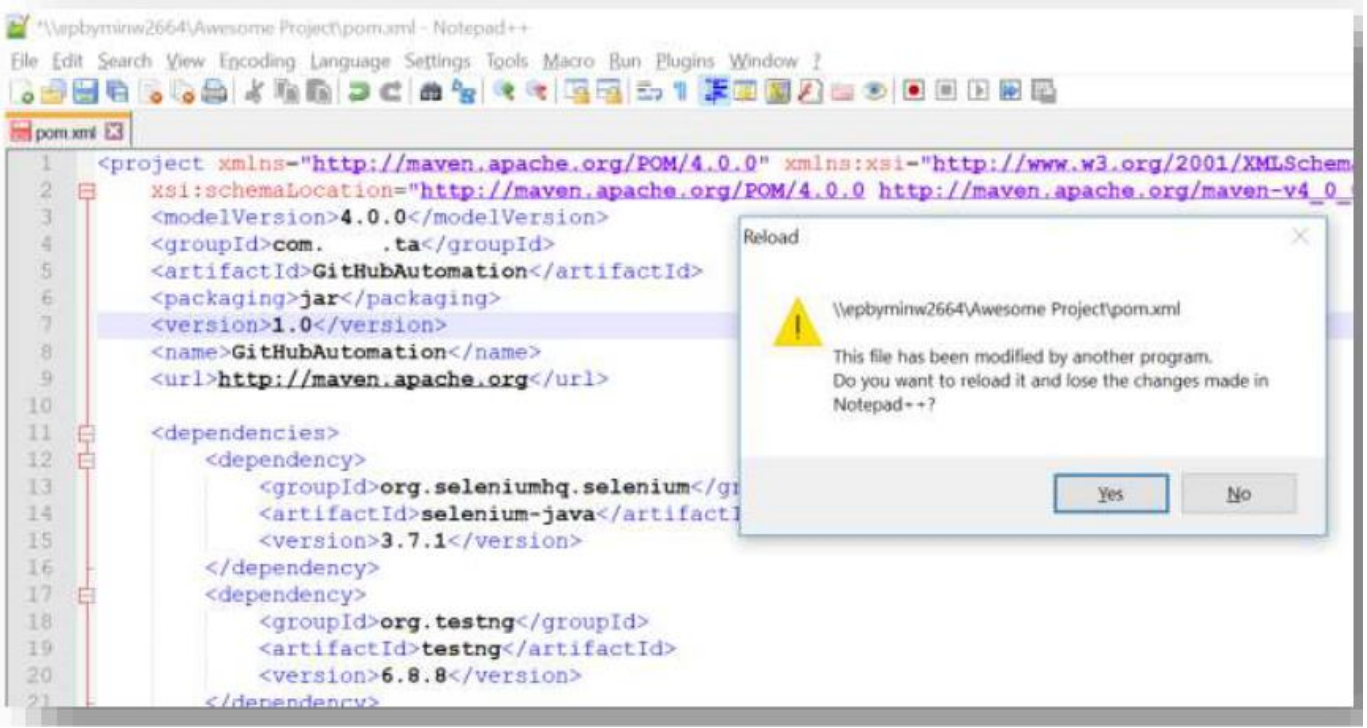
Version Control Systems

Standalone work - Simple user



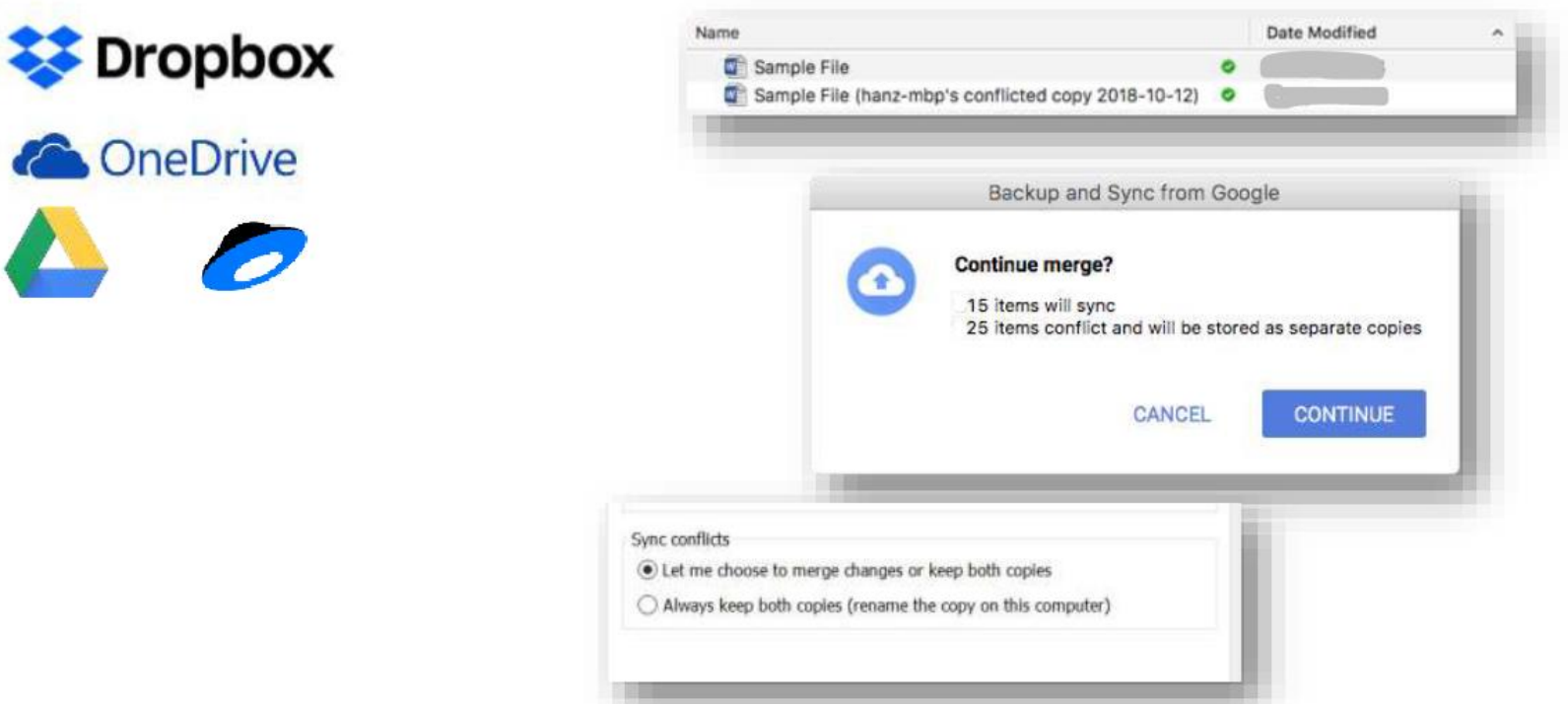
Version Control Systems

Team work - Network share



Version Control Systems

Standalone&Team work - Cloud Tech



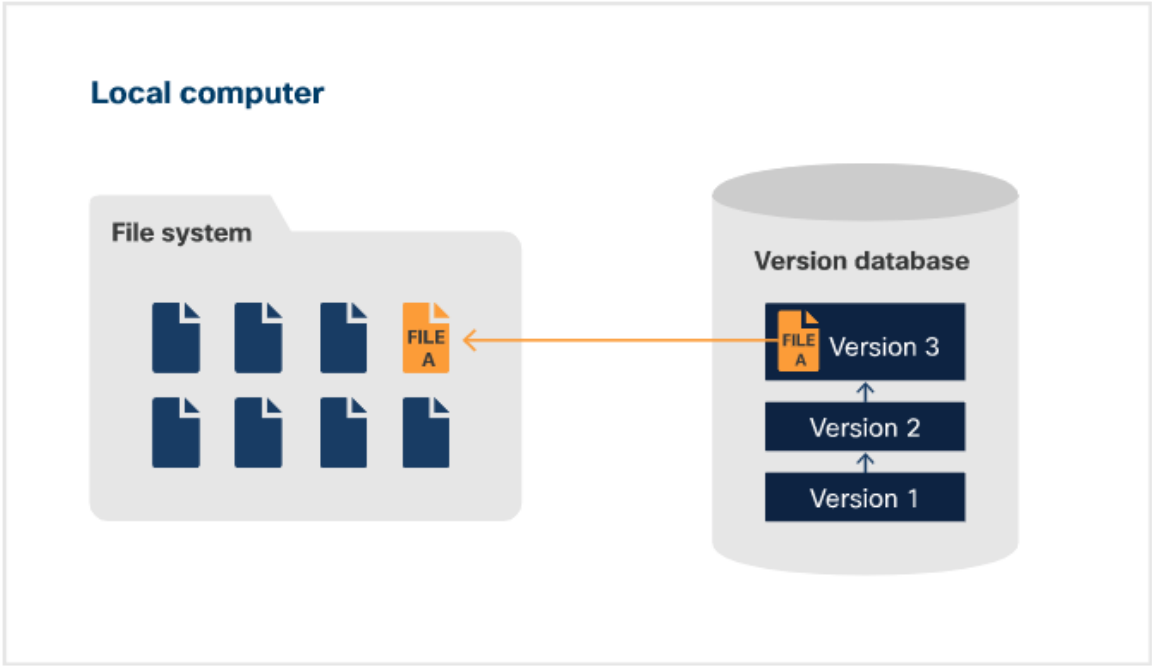
Types of Version Control Systems

- Version control, also called version control systems, revision control or source control, is a way to manage changes to a set of files in order to keep a history of those changes.
- Benefits of version control are:
  - Enables collaboration
  - Accountability and visibility
  - Work in isolation
  - Safety
  - Work anywhere
- There are three types of version control systems:
  - Local
  - Centralized
  - Distributed

Types of Version Control Systems (Contd.)

Local Version Control System (LVCS)

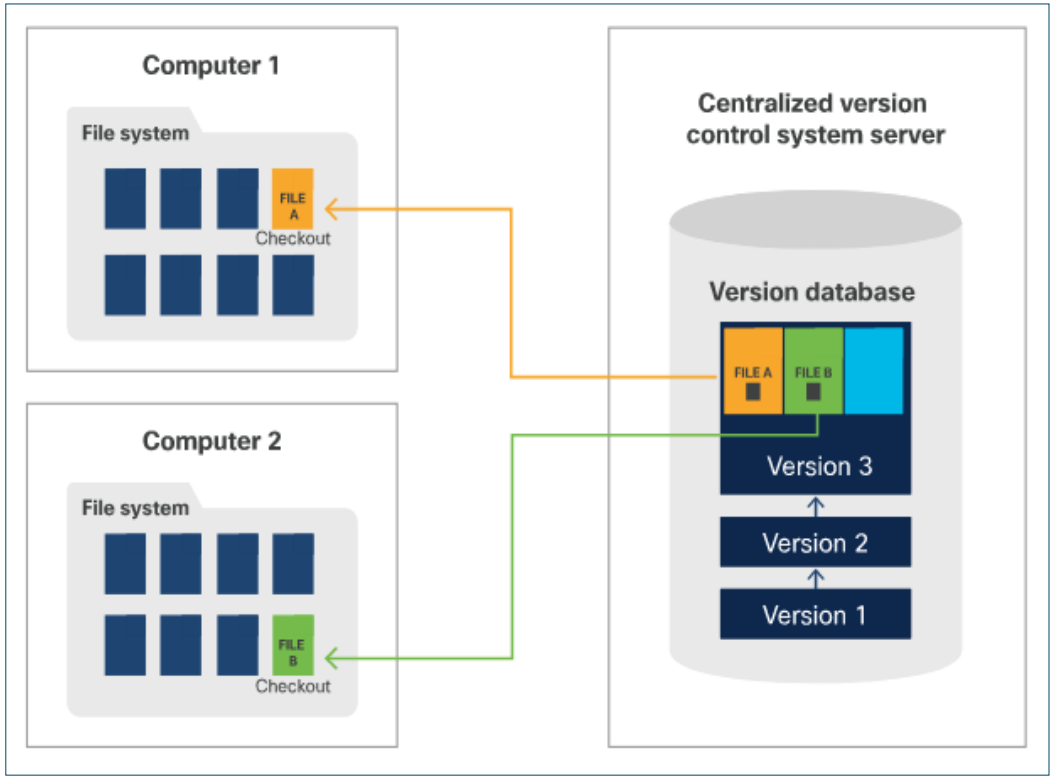
- LVCS uses a simple database to keep track of all of the changes to the file.
- In most cases, the system stores the delta between the two versions of the file.
- When the user wants to revert to the file, the delta is reversed to get to the requested version.



Types of Version Control Systems (Contd.)

Centralized Version Control System (CVCS)

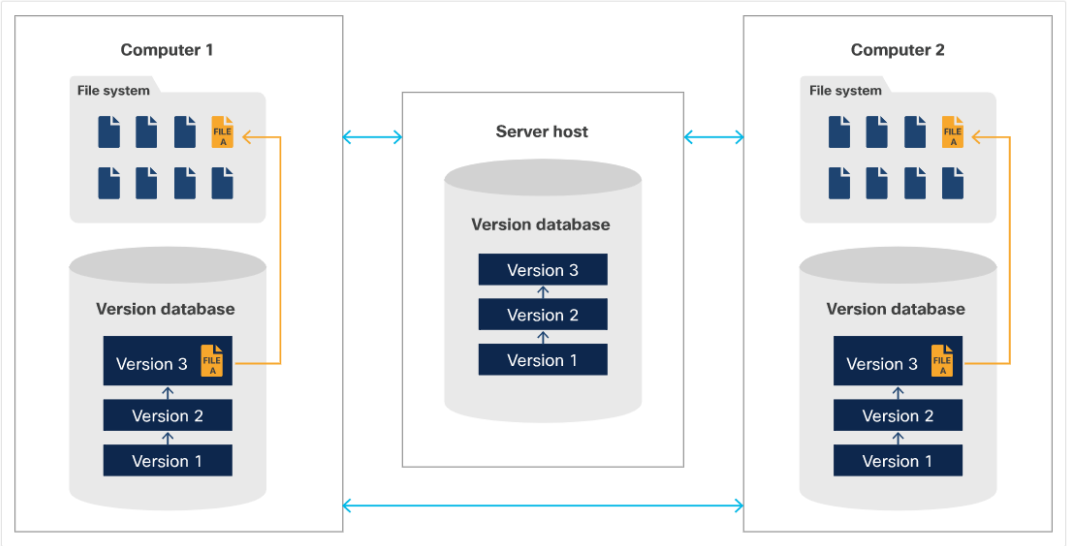
- CVCS uses a server-client model.
- The repository is stored in a centralized location, on a server.
- In CVCS, only one individual can work on a particular file at a time.
- An individual must check out the file to lock it and make the required changes and check in once done.



Types of Version Control Systems (Contd.)

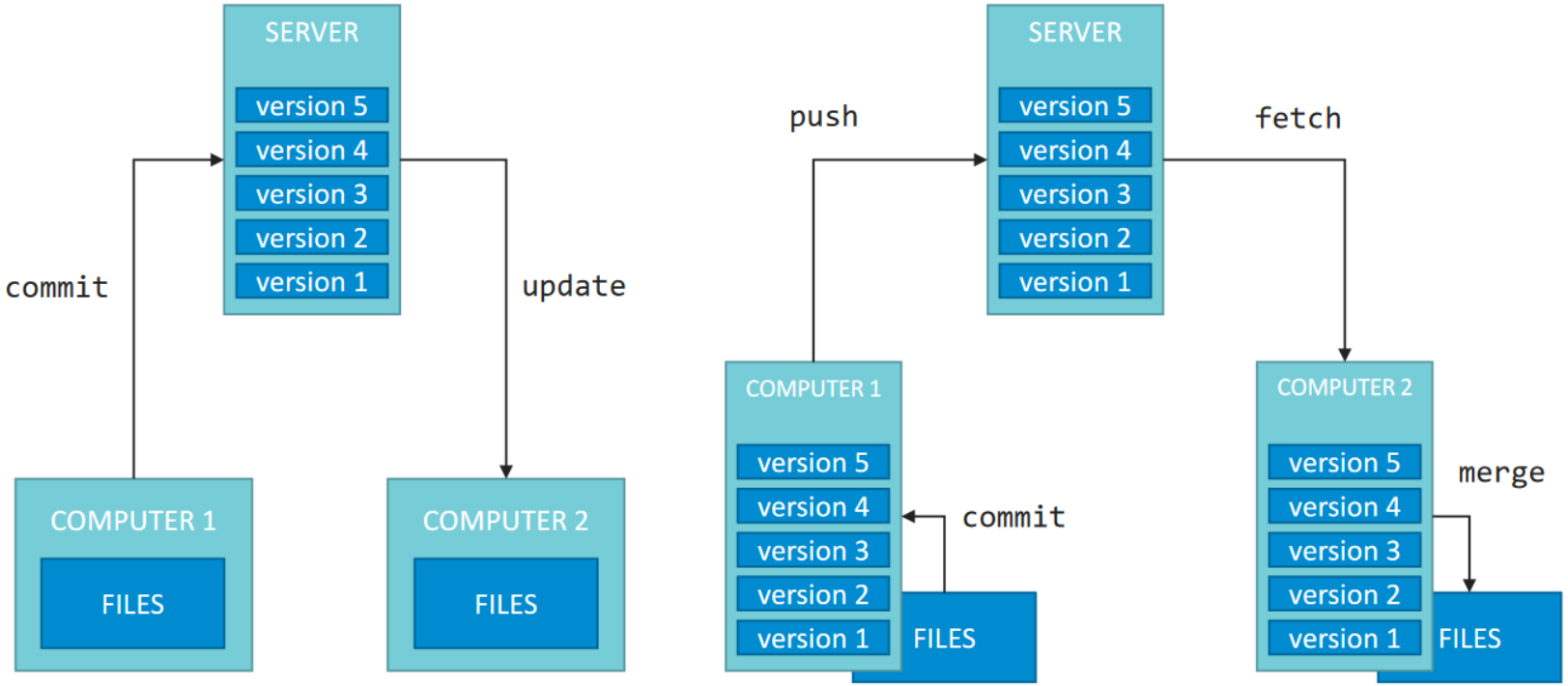
Distributed Version Control system (DVCS)

- DVCS is a peer-to-peer model.
- The repository can be stored on a client system, but it is usually stored in a repository hosting service.
- In DVCS, every individual can work on any file, at the same time, because the local file in the working copy is being modified. Hence, locking is not required.
- When the individual has made the changes, they push the file to the main repository that is in the repository hosting service, and the version control system detects any conflicts between file changes.

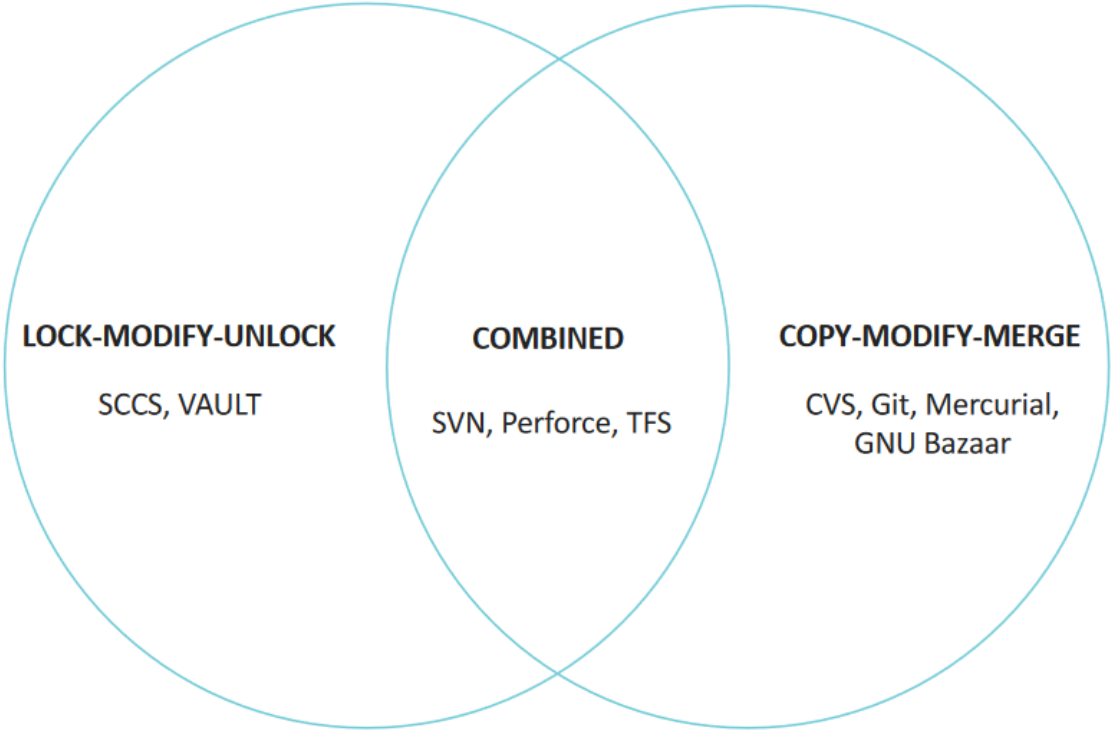


Version Control Systems

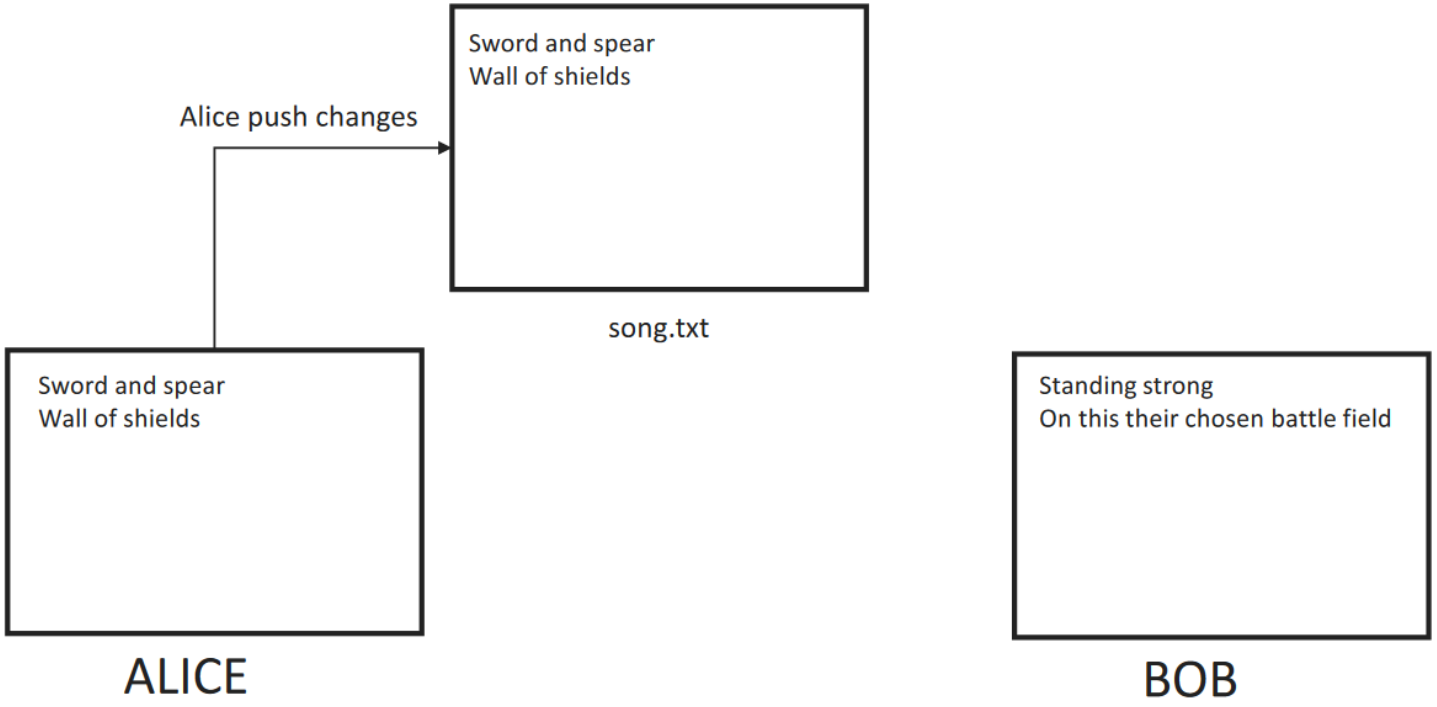
CVCS vs DVCS



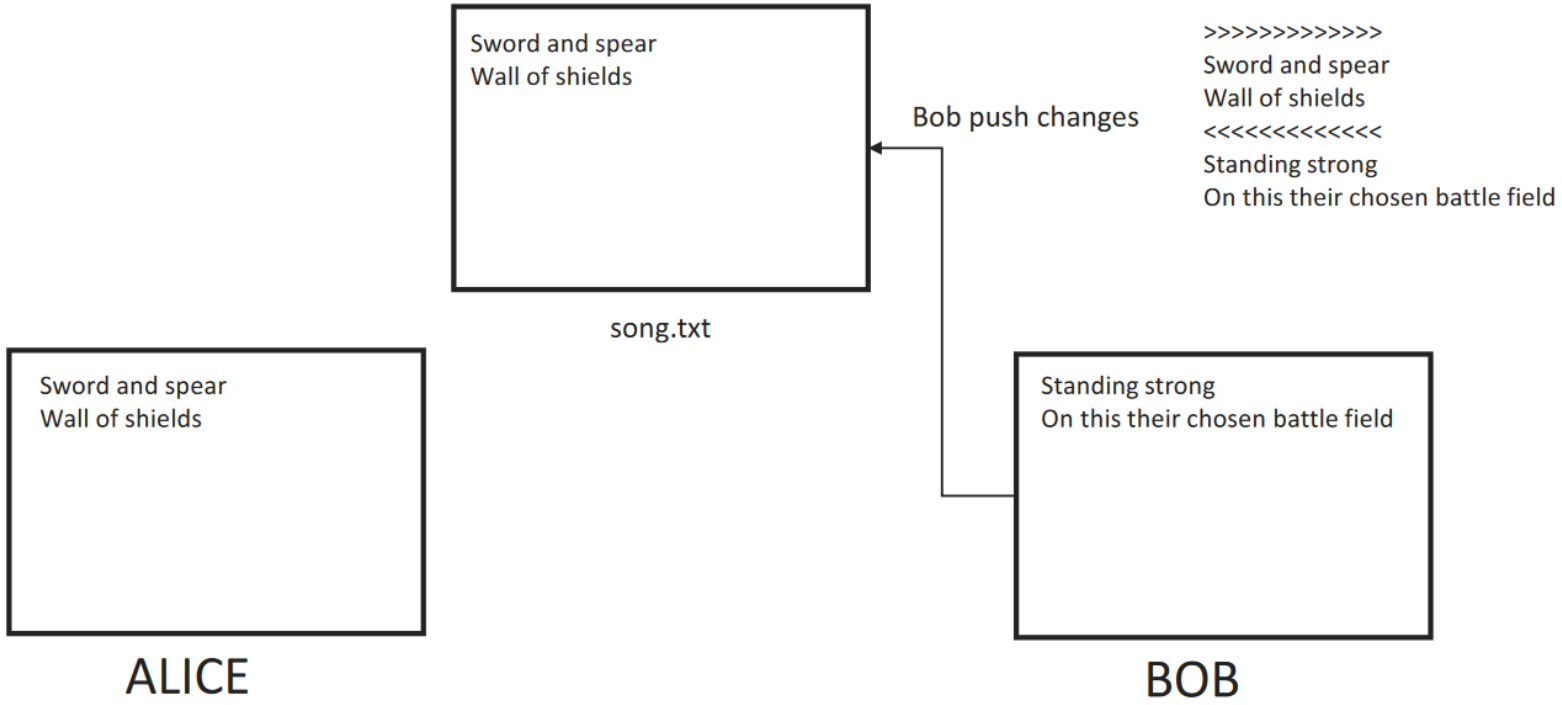
Types of Version Control Systems



Copy-modify-merge strategy

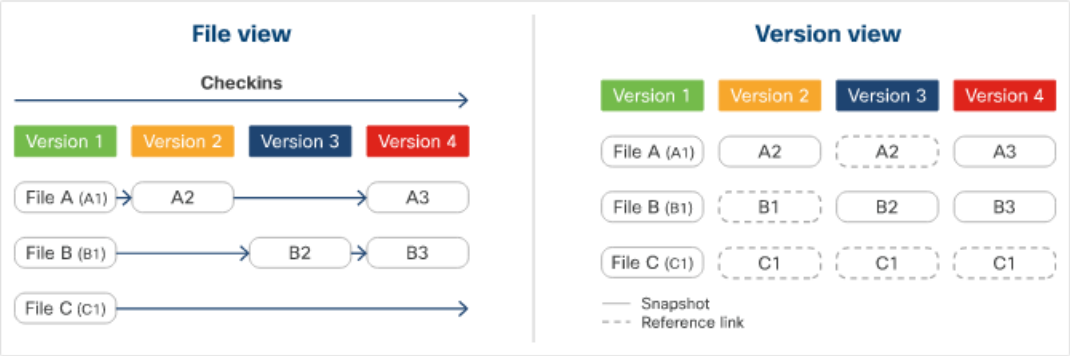


Copy-modify-merge strategy



Git

- Git is an open source implementation of a distributed version control system that is currently the latest trend in software development.
- A Git client must be installed on a client machine. It is available for MacOS, Windows, and Linux/Unix.
- One key difference between Git and other version control systems is that Git stores data as snapshots instead of differences (the delta between the current file and the previous version).



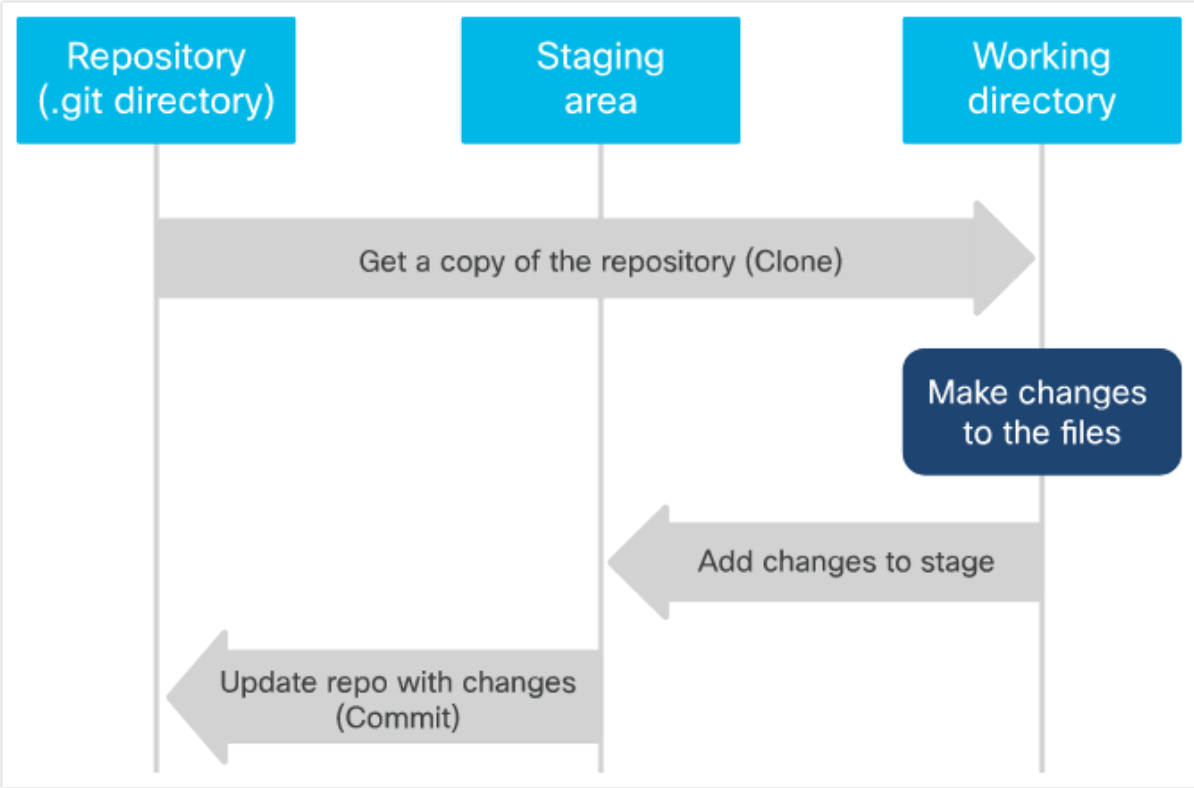
- If the file does not change, Git uses a reference link to the last snapshot in the system instead of taking a new and identical snapshot.

Git (Contd.)

- **Git is organized by 3s**- three stages and three states.

- The three stages are:
  - Repository (the .git directory)
  - Working directory
  - Staging area

- The three states are:
  - Committed
  - Modified
  - Staged
  - Untracked



Local vs. Remote Repositories

- Git has two types of repositories: **local** and **remote**.
- A **local repository** is stored on the file system of a client machine, which is the same one on which the git commands are being executed.
- A **remote repository** is stored somewhere other than the client machine, usually a server or repository hosting service.
- A remote repository with Git continues to be a DVCS because the remote repository will contain the full repository, which includes the code and the file history.
- When a client machine clones the repository, it gets the full repository without requiring to lock it, as in a CVCS.
- After the local repository is cloned from the remote repository or the remote repository is created from the local repository, the two repositories are independent of each other until the content changes are applied to the other branch through a manual Git command execution.

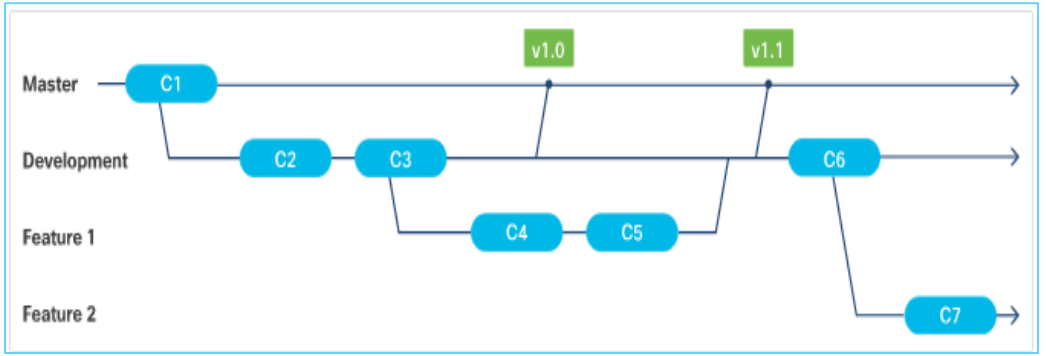
What is Branching?

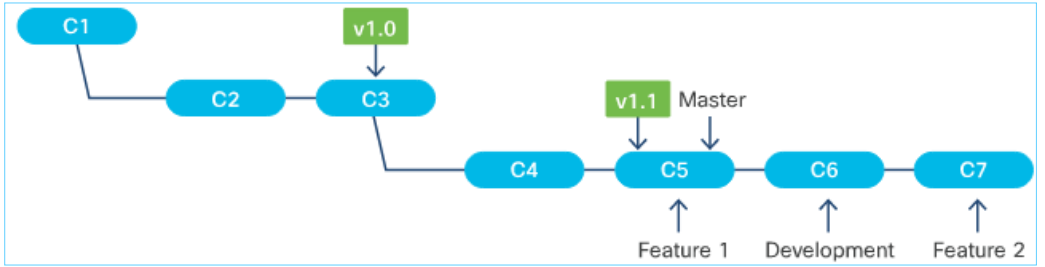
- Branching enables users to work on code independently without affecting the main code in the repository. When a repository is created, the code is automatically put on a branch called Master.

- Branches can be local or remote, and they can be deleted and have their own history, staging area, and working directory.

- Git's branch creation is lightweight, and switching between branches is almost instantaneous.

- When a user goes from one branch to another, the code in their working directory and the files in the staging area change accordingly, but the repository (.git) directories remain unchanged.





GitHub and Other Providers

- **Git and GitHub are not the same**.
- While Git is an implementation of distributed version control and provides a command line interface, GitHub is a service provided by Microsoft that implements a repository hosting service with Git.
- In addition to providing the distributed version control and source code management functionality of Git, GitHub provides additional features such as:
  - code review
  - documentation
  - project management
  - bug tracking
  - feature requests
- GitHub introduced the concept of the ‘pull request’, which is a way of formalizing a request by a contributor to review changes such as new code, edits to existing code, etc., in the contributor's branch for inclusion in the project's main or other curated branches.

Download and install git

https://git-scm.com/downloads

Setup using default options*

Create **github** account

https://github.com/signup?source=login

Git tutorials

https://learngitbranching.js.org/?locale=ru_RU

https://githowto.com/ru

https://www.atlassian.com/git/tutorials

Git Commands

commands to get help:

```
$ git help <verb>
```

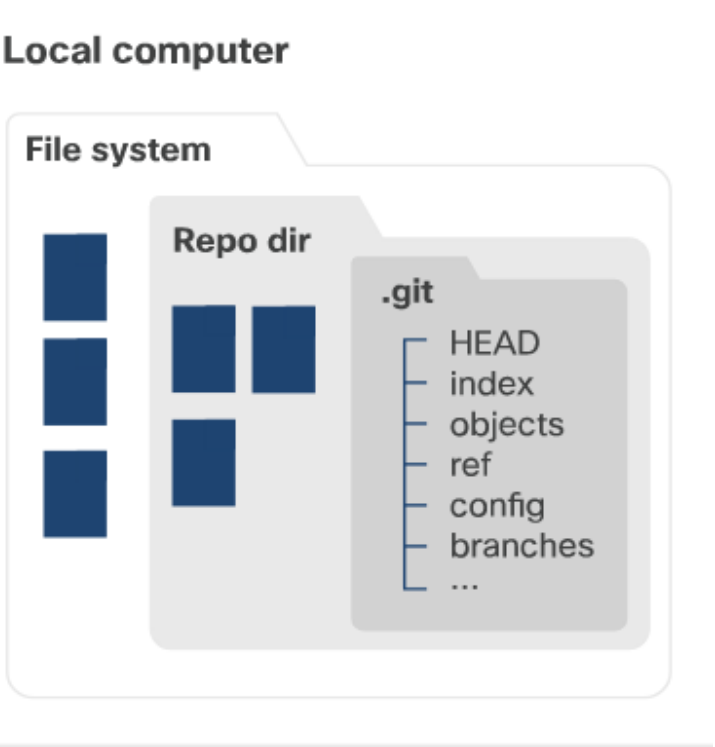
Setting up Git

- To configure Git, use the --global option to set the initial global settings.

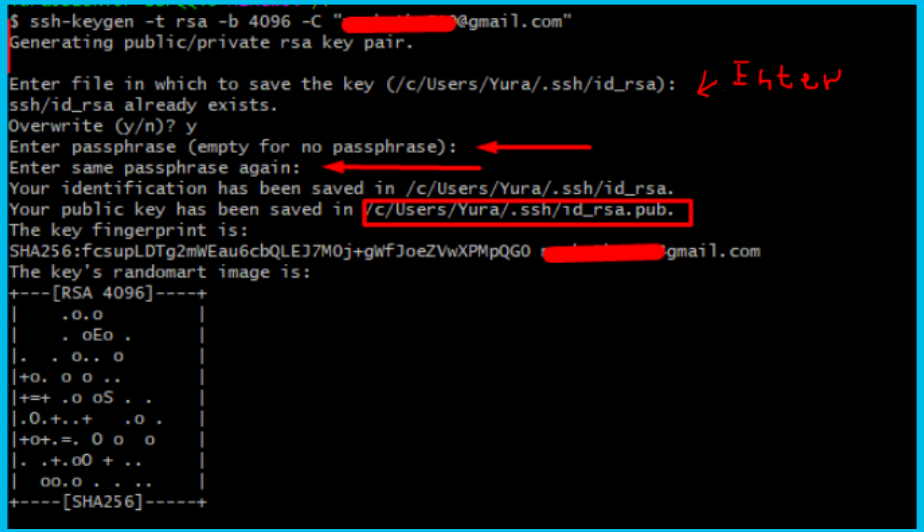
Command: **git config --global** key value

Create a New Git Repository

- Git provides a **git init** command to create an empty Git repository, or make an existing folder a Git repository.
- When a new or existing project becomes a Git repository, a hidden .git directory is created in that project folder.
- The .git directory is the repository that holds the metadata such as the compressed files, the commit history, and the staging area. In addition, Git also creates the master branch.



Watching the video to configure git
FOR YOUR HOME WORK



- **generate SSH key pair**

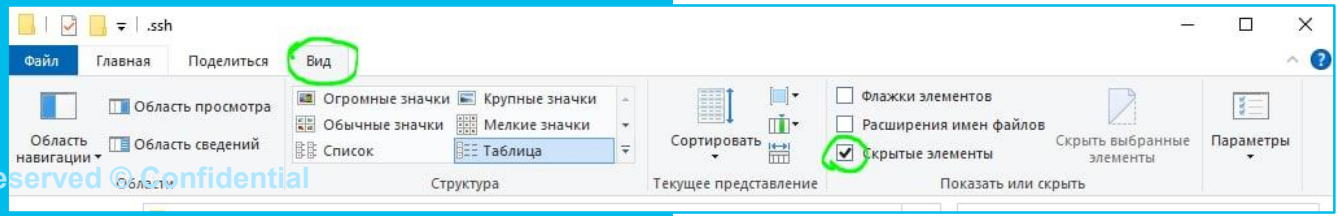
```
ssh-keygen -t rsa -C your_email@lala.com
```

- upload **public key** to your github profile
- **config your git username and email**

```
git config --global user.name "Your Name"

git config --global user.email "Your email"

git config --global core.editor "code --wait"
```

- **review these settings by**

```
git config --list
```



Git Commands (Contd.)

Command: **git init**

- To make a new or existing project a Git repository, use the following command:

  $ git init <project directory>

  where the <project directory> is the absolute or relative path to the new or existing project.

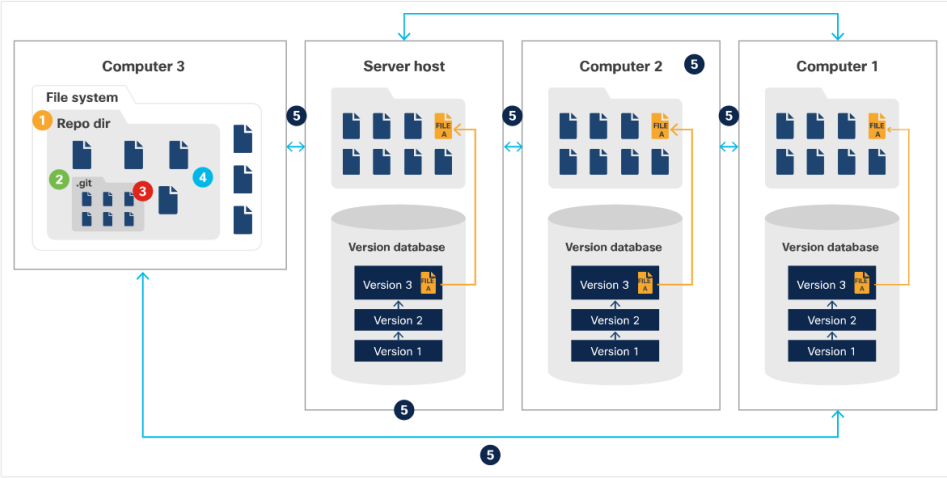
- For a new Git repository, the directory in the provided path will be created first, followed by the creation of the .git directory.

********Get an Existing Git Repository**

- ******Command**: **git clone <repository> [target directory]**

  where <repository> is the location of the repository to clone.

- Git supports four major transport protocols for accessing the <repository>: Local, Secure Shell (SSH), Git, and HTTP.



Git Commands (Contd.)

View the Modified Files in the Working Directory

- Git provides a git status command to get a list of files that have differences between the working directory and the parent branch.
- **Command**: **git status**

Adding and Removing Files

Adding Files to the Staging Area

- **Command**: **git add**
- This command can be used more than once before the Git repository is updated (using commit).
- Only the files specified in the git command can be added to the staging area
- To add a single file to the staging area:
  $ git add <file path>
- To add multiple files to the staging area, where the <file path> is the absolute or relative path of the file to be added to the staging area.
  $ git add <file path 1> ... <file path n>
- To add all the changed files to the staging area: $ git add .

Git Commands (Contd.)

add changes in working directory to the staging area:

```
$ git add <filename> [<filename>]
```

command accepts wildcards as parameters:

```
$ git add *.go
```

dot '.' shortcut is used to add all files:

```
$ git add .
```

by default, add is not applied to removed files, to fix it use --all parameter:

```
$ git add --all .
```

Adding and Removing Files (Contd.)

Removing Files from the Git Repository

- There are two ways to remove files from the Git repository.
- **Option 1**: git rm command is used to remove files from the Git repository and add to the staging area.
  - **Command**: **git rm**
  - To remove the specified file(s) from the working directory and add the change to the staging area, use the following command:
    $ git rm <file path 1> ... <file path n>

    where <file path> is the absolute or relative path of the file to be deleted from the Git repository.

Adding and Removing Files (Contd.)

- **To add the specified file(s) to be removed from the staging area without removing the file(s) itself from the working directory, use the following command:**

  $ git rm --cached <file path 1> ... <file path n>

  The git rm command will not work if the file is already in the staging area with changes.

- **Option 2**: This option is a two-step process. First use the regular filesystem command to remove the file(s) and then add the file to the stage using the Git command.

  $ rm <file path 1> ... <file path n>
  $ git add <file path 1> ... <file path n>

This two step process is equivalent to using the git rm <file path 1> ... <file path n> command. Using this option does not allow the file to be preserved in the working directory.

Updating Repositories

Updating the Local Repository with the Changes in the Staging Area

Command: **git commit**

- This command combines all the content changes in the staging area into a single commit and updates the local Git repository.

- To commit the changes from the staging area, use the following command:
  $ git commit

- To commit the changes from the staging area with a
  message, use the following command:
  $ git commit -m "<message>"

Updating Repositories (Contd.)

Updating the Remote Repository

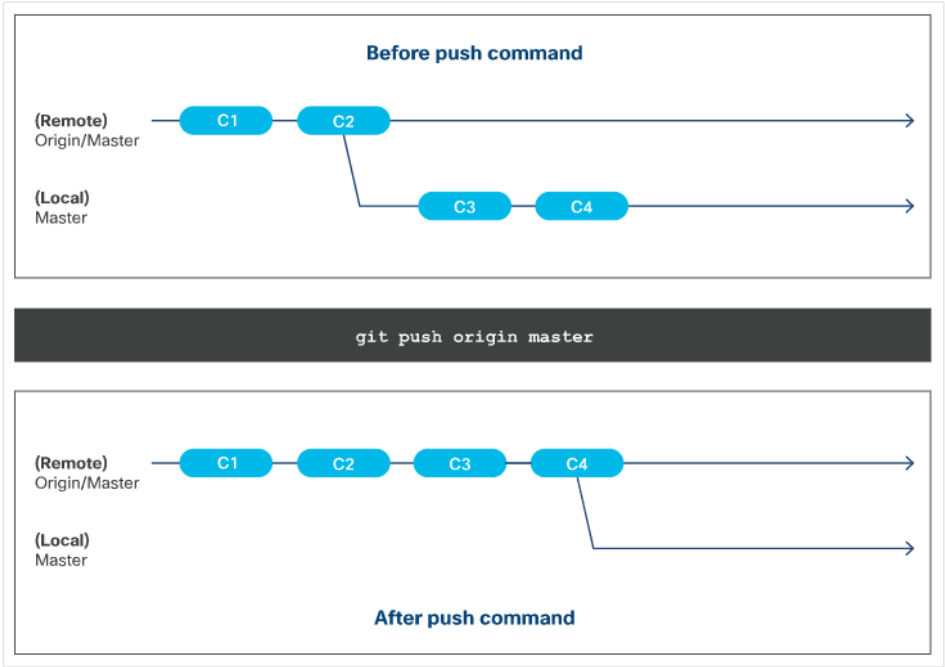
Command: **git push**

Better to use: git push origin

- This command will not execute successfully if there is a conflict with adding the changes from the local Git repository to the remote Git repository.
- To update the contents from the local repository to a particular branch in the remote repository, use the following command:

  $ git push origin <branch name>

- To update the contents from the local repository to the master branch of the remote repository, use the command: **$ git push origin main**



Updating Repositories (Contd.)

Updating Your Local Copy of the Repository

- Local copies of the Git repository do not automatically get updated when another contributor makes an update to the remote Git repository.
- Updating the local copy of the repository is a manual step.

Command: **git pull**

- When executing the command, the following steps occur:
  - The local repository (.git directory) is updated with the latest commit, file history, and so on from the remote Git repository.
  - The working directory and branch is updated with the latest content from step 1.
  - A single commit is created on the local branch with the changes from step 1.
  - The working directory is updated with the latest content.

Updating Repositories (Contd.)

- To update the local copy of the Git repository from the parent branch, use the following command:

  $ git pull

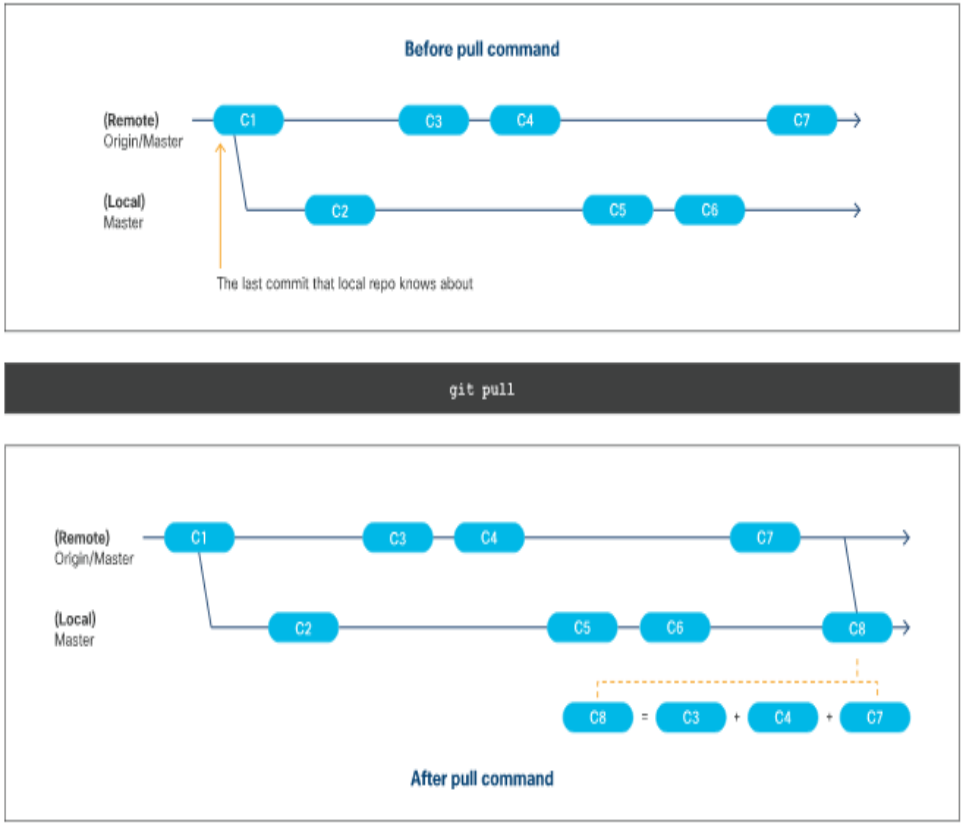
  Or

  $ git pull origin

- To update the local copy of the Git repository from a specific branch, use the following command:

  $ git pull origin <branch>

```
git pull == git fetch + git merge
```



Branching Features

Creating and Deleting a Branch

Option 1: git branch command to list, create, or delete a branch.

$ git branch <parent branch> <branch name>

Option 2: git checkout command to switch branches by updating the working directory with the contents of the branch.

$ git checkout -b <parent branch> <branch name> - git checkout -b Lecture_1

Deleting a Branch

- To delete a branch, use the following command:

  $ git branch -d <branch name>

Get a List of all Branches

- To get a list of all the local branches, use the following command:

  $ git branch Or **$ git branch --list**

.gitignore

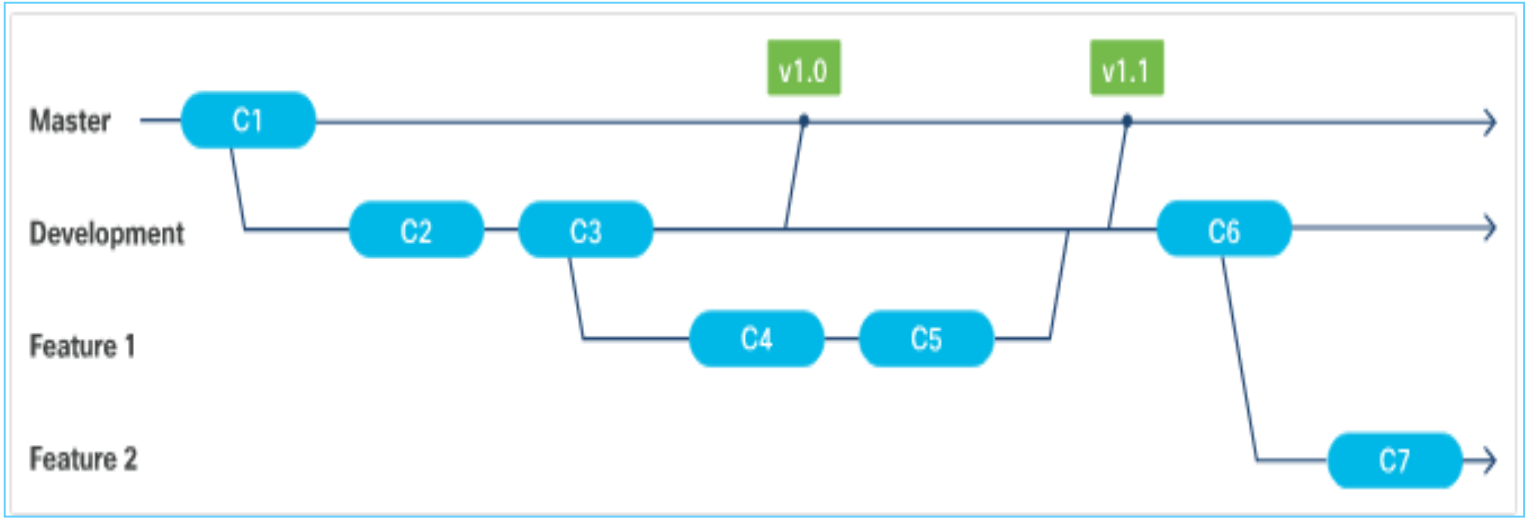
gitignore files specify intentionally untracked files that Git should ignore.

```
# no .log files
*.log

# but do track error.log, even though you're
ignoring .log files above
!error.log

# only ignore the TODO file in the current
directory, not subdir/TODO
/TODO

# ignore all files in the build/ directory
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

https://tyapk.ru/blog/post/gitignore

GIT Flow

main
or



https://www.atlassian.com/ru/git/tutorials/comparing-workflows/gitflow-workflow

https://www.atlassian.com/ru/git/tutorials/making-a-pull-request

Home work gitDemo

Setup git
Create github account
Complete the tutorials

Step by step videos will be published in the telegram channel

Tasks:

- **Install & setup git**
- **Create a github account and follow the steps from video in telegram**
- **Complete the tutorials:**

https://learngitbranching.js.org/?locale=ru_RU

https://githowto.com/ru

ЗАДАНИЯ

Формат сообщения которое вы присылаете мне

(после полного выполнения домашнего задания, только один раз) в Telegram:

Добрый день/вечер. Я Вася Пупкин и

это мои домашние задания к лекции 4 Часть 1 про гит.

Мой github аккаунт:

https://github.com/your_github_username

Мой репозиторий gitDemo:

https://github.com/your_github_username/gitDemo.git

+Скриншоты выполненных туториалов

Крайний срок сдачи 03/10 в 21:00 (можно раньше, но не позже)

https://docs.github.com/articles/using-pull-requests

Create your possibilities. Bye bye.