**Name:** Rupin Malik and Kartik Menon

**UID** 2020300035 and 2020300040

**Project Name:** WeChange **Aim:** Version Control

# Theory:

## How version control helps high performing development and DevOps teams prosper

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage

changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have

collected and refined through careful effort. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

Software developers working in teams are continually writing new source code and changing

existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature

while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the

same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So testing and development proceed together until a new version is ready.

Good version control software supports a developer's preferred workflow without imposing one particular way of working. Ideally it also works on any platform, rather than dictate what

operating system or tool chain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the frustrating and clumsy mechanism of file locking - giving the green light to one developer at the expense of blocking the progress of others.

Software teams that do not use any form of version control often run into problems like not knowing which changes that have been made are available to users or the creation of incompatible changes between two unrelated pieces of work that must then be painstakingly untangled and reworked. If you're a developer who has never used version control you may have added versions to your files, perhaps with suffixes like "final" or "latest" and then had to later deal with a new final version. Perhaps you've commented out code blocks because you want to disable certain functionality without deleting the code, fearing that there may be a use for it later. Version control is a way out of these problems.

Version control software is an essential part of the every-day of the modern software team's professional practices. Individual software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version

control also gives them even on small solo projects. Once accustomed to the powerful benefits of version control systems, many developers wouldn't consider working without it even for

non-software projects.

## Benefits of version control systems:

Using version control software is a best practice for high performing software and DevOps teams. Version control also helps developers move faster and allows software teams to preserve efficiency and agility as the team scales to include more developers.

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is

called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source. Regardless of

what they are called, or which system is used, the primary benefits you should expect from version control are as follows.

A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files. This history should also include the author, date and written notes on the purpose of each change.

Having the complete history enables going back to previous versions to help in root cause

analysis for bugs and it is crucial when needing to fix problems in older versions of software. If the software is being actively worked on, almost everything can be considered an "older version" of the software.

**Branching and merging:** Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from

each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.

**Traceability:** Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause

analysis and other forensics. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can

enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

While it is possible to develop software without using any version control, doing so subjects the project to a huge risk that no professional team would be advised to accept. So the question is not whether to use version control but which version control system to use.

There are many choices, but here we are going to focus on just one, Git. Learn more about other types of version control software.

## About TortoiseSVN:

TortoiseSVN is a really easy to use Revision control / version control / source control software for Windows. It is based on Apache™ Subversion (SVN)®; TortoiseSVN provides a nice and easy user interface for Subversion.

It is developed under the GPL. Which means it is completely free for anyone to use, including in a commercial environment, without any restriction. The source code is also freely available, so you can even develop your own version if you wish to.

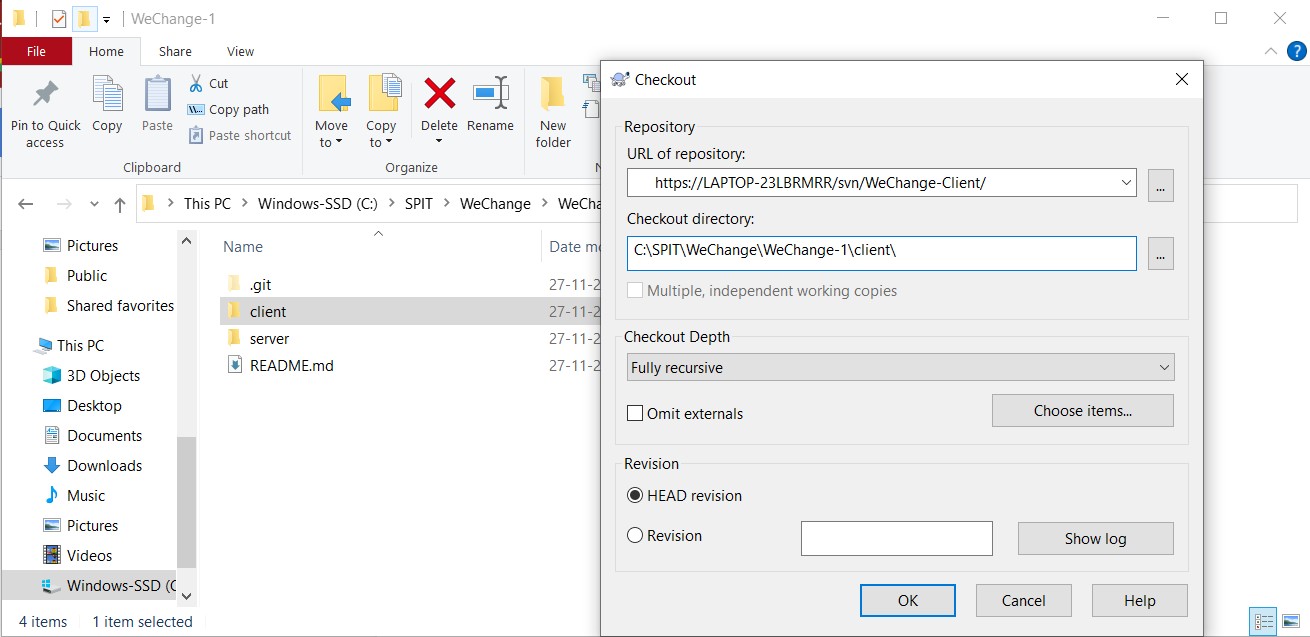
Since it's not an integration for a specific IDE like Visual Studio, Eclipse or others, you can use it with whatever development tools you like, and with any type of file.

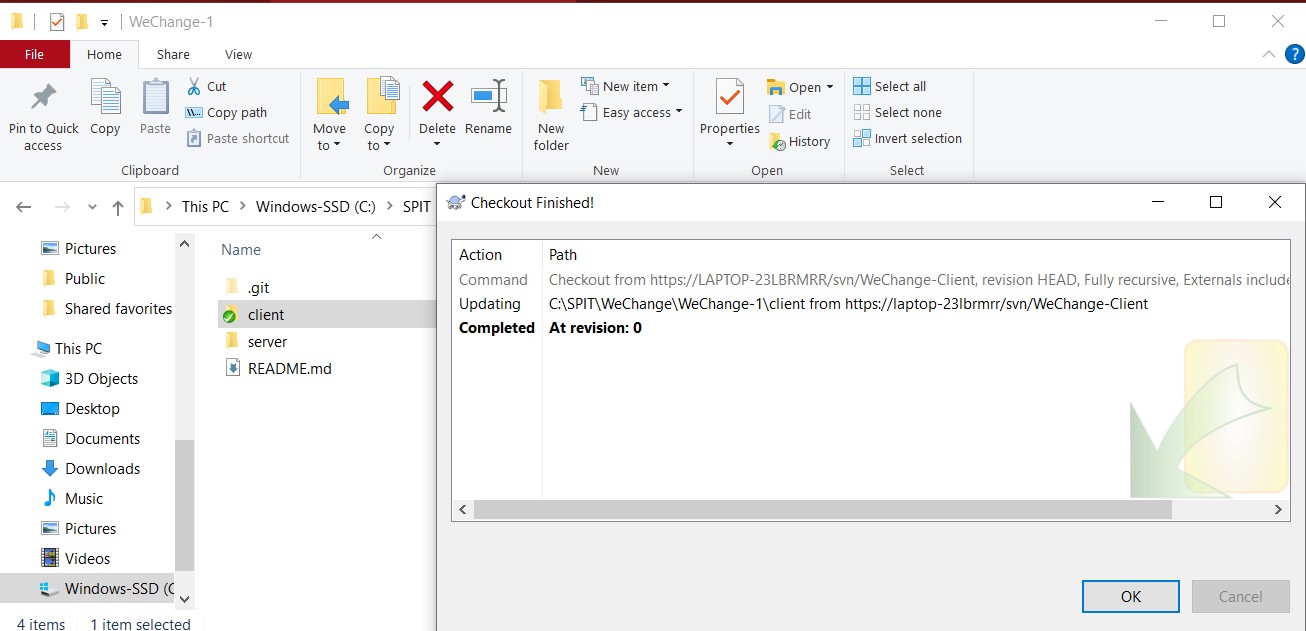
## Tortoise and Visual Svn Server Alternatives are as follows:

* TortoiseGit: TortoiseGit is a Windows Shell Interface to Git based on TortoiseSVN. It's open source and can be built entirely with freely available software
* SVN (Subversion): Apache Subversion which is often abbreviated as SVN, is a software versioning and revision control system distributed under an open source license • Git: Git is a free and open source distributed version control system
* TortoiseHg: TortoiseHg consists of a shell extension, which provides overlay icons and context menus in your file explorer, and a command line program named thg.exe which can launch the TortoiseHg tools.
* GitHub: GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.
* GitLab: GitLab is an open source code repository and collaborative software development platform for large DevOps and DevSecOps projects. GitLab is free for individuals. GitLab offers a location for online code storage and capabilities for issue tracking and CI/CD.
* Bitbucket: Bitbucket is a Git-based source code repository hosting service owned by Atlassian

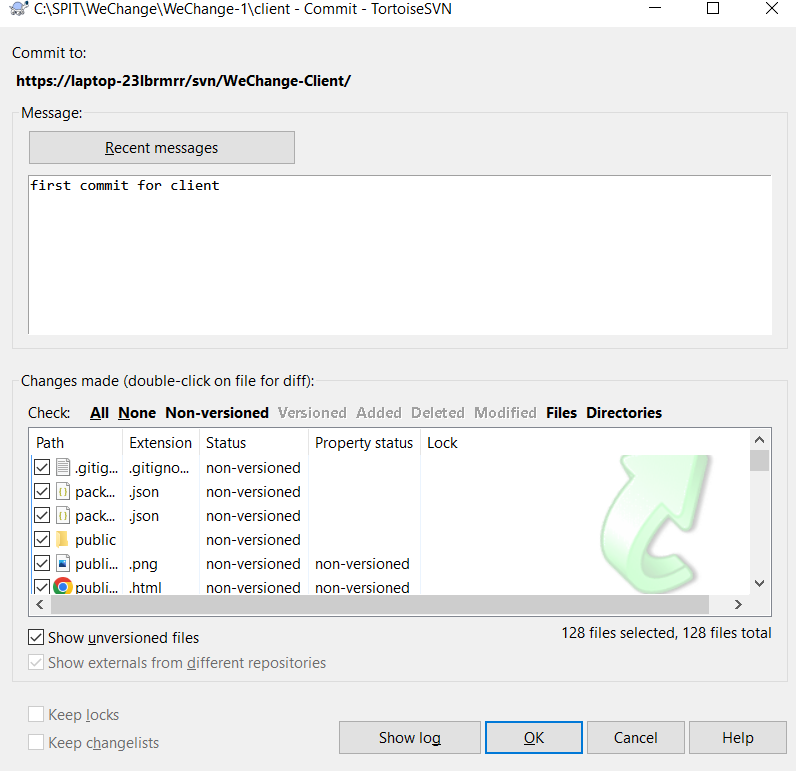
•GitHub Enterprise: GitHub Enterprise Server is a self-hosted platform for software development within your enterprise. Your team can use GitHub Enterprise Server to build and ship software using Git version control, powerful APIs, productivity and collaboration tools, and integrations.

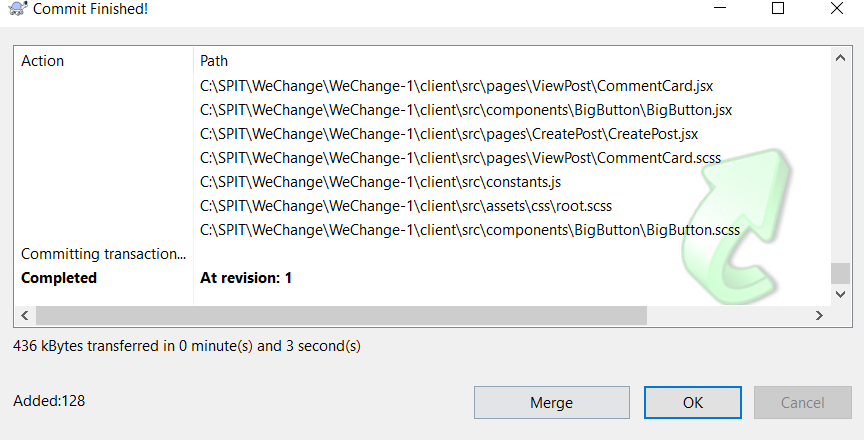
# Implementation:

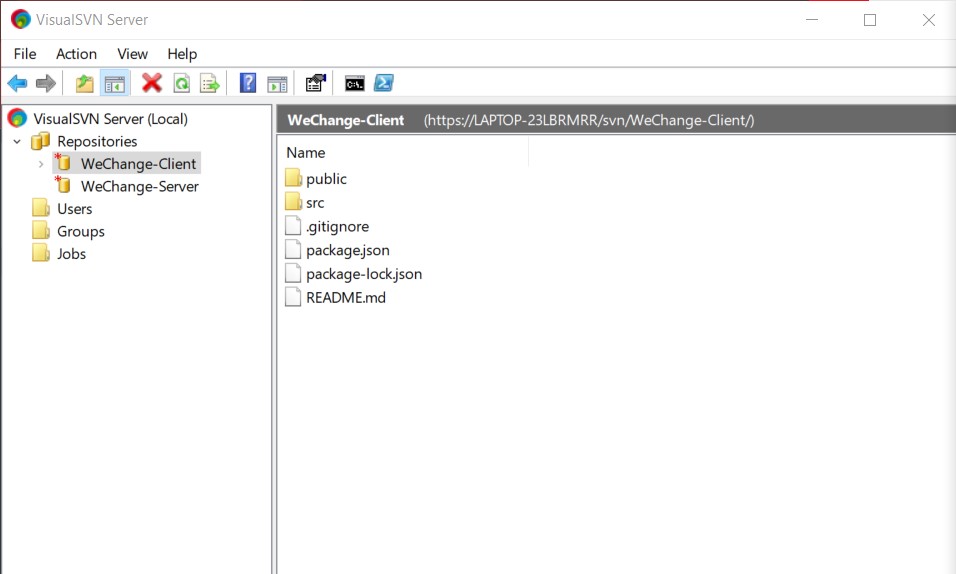
Checkout Client folder:



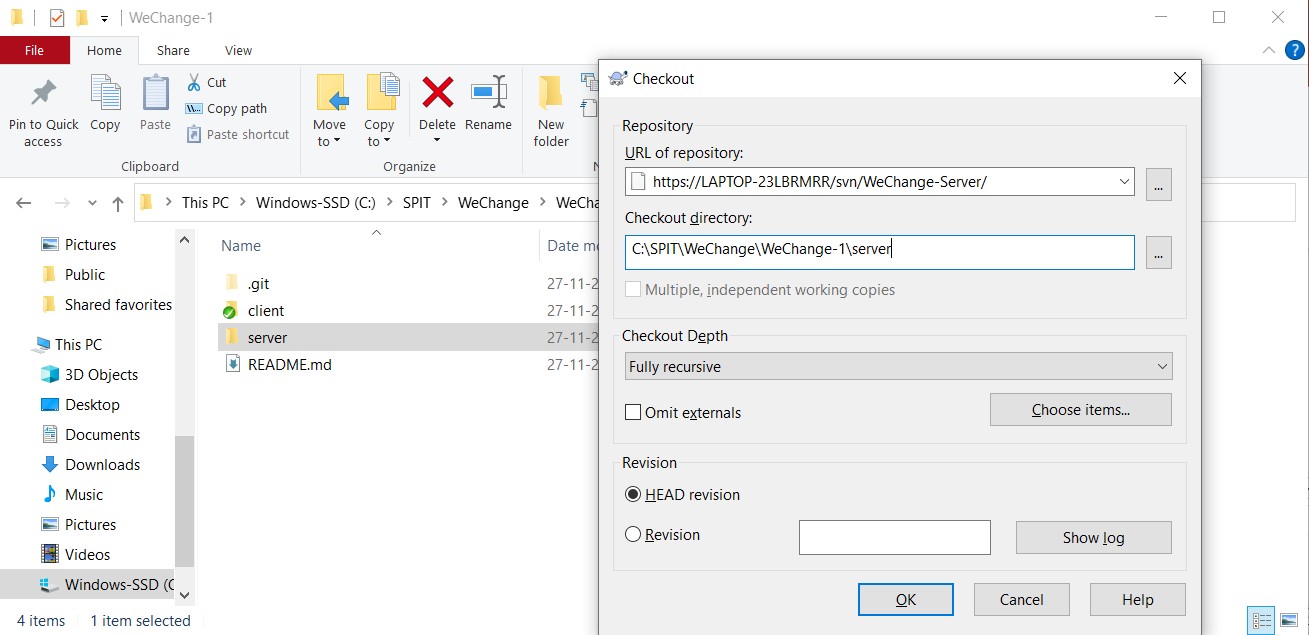
Committing the files in Client folder to VisualSVN servers:

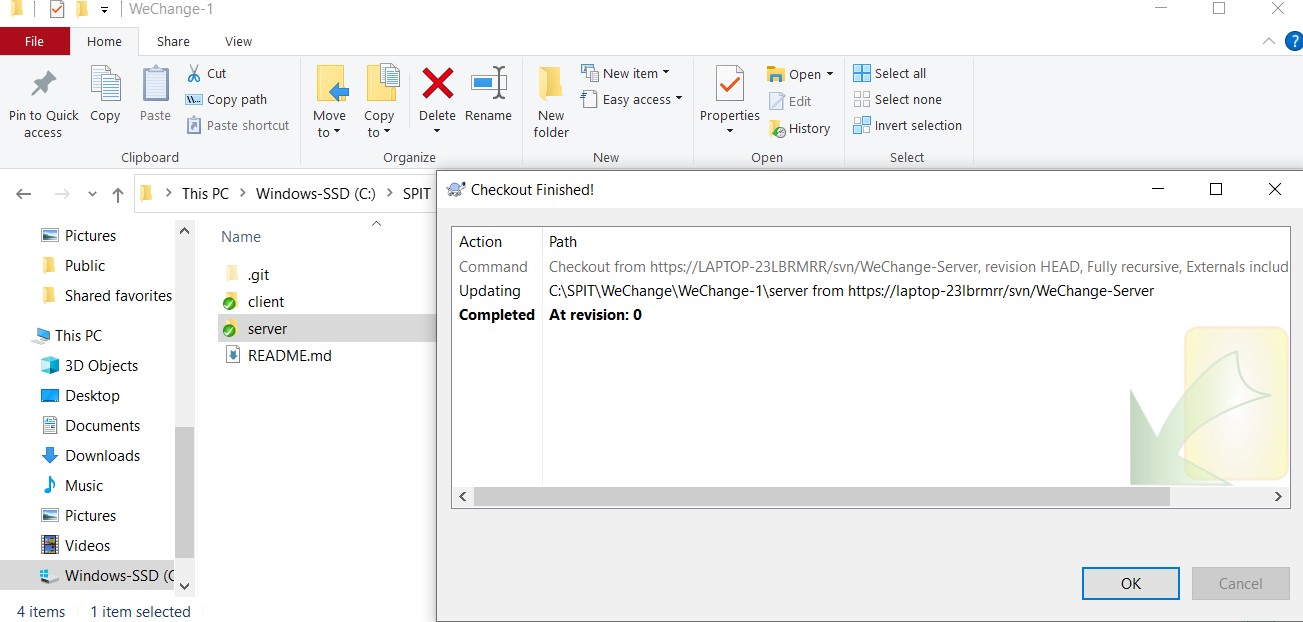


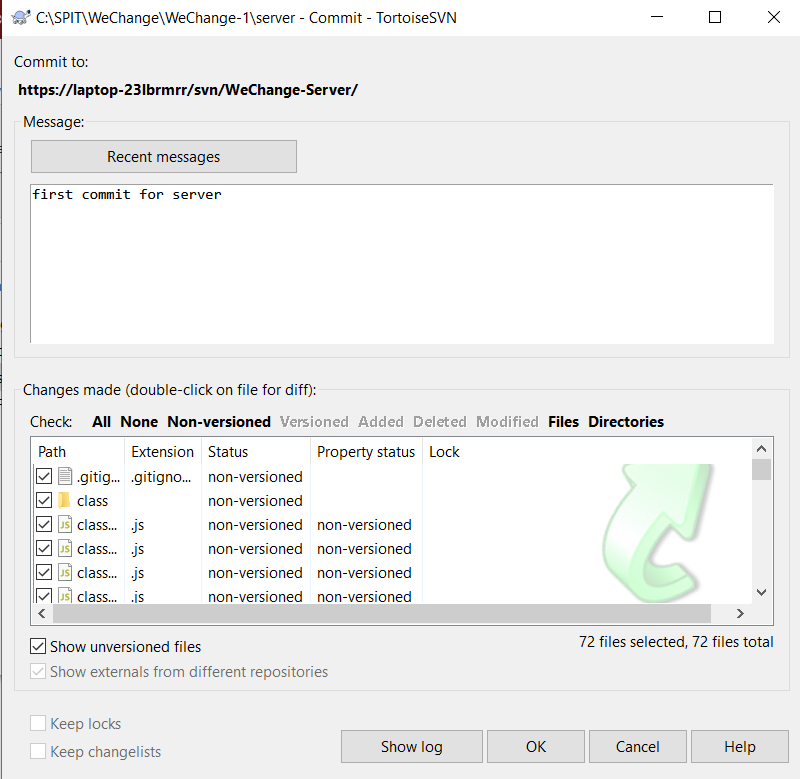


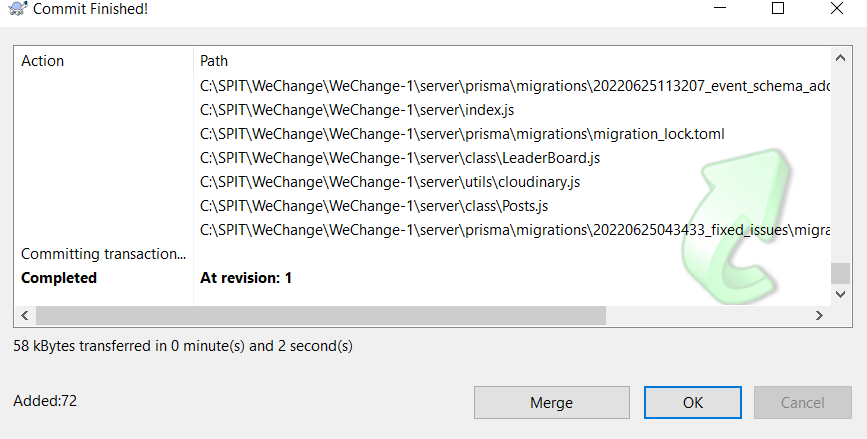
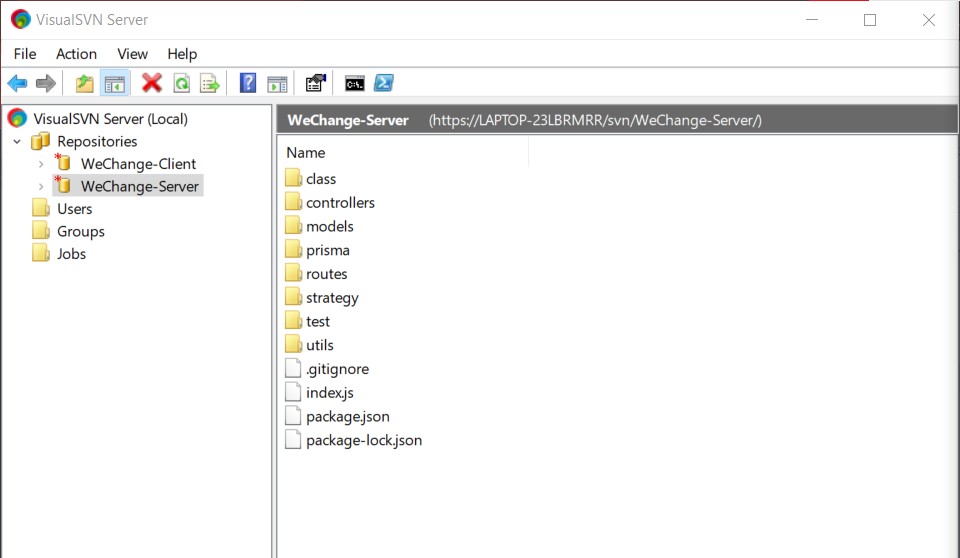


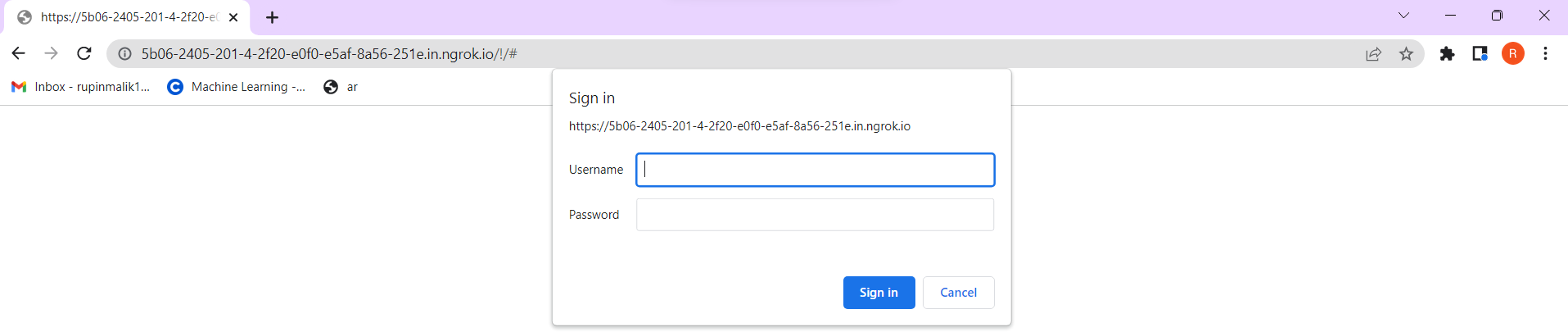
Checkout Server folder:

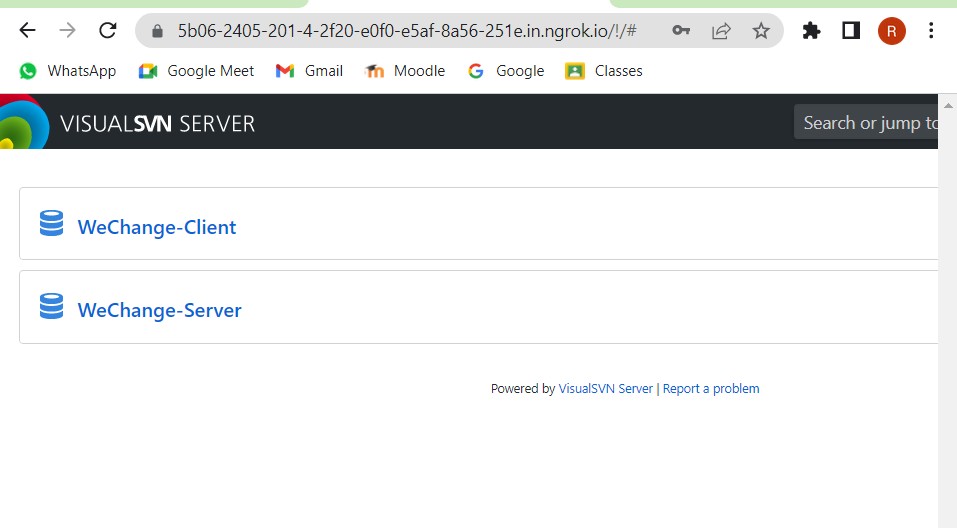


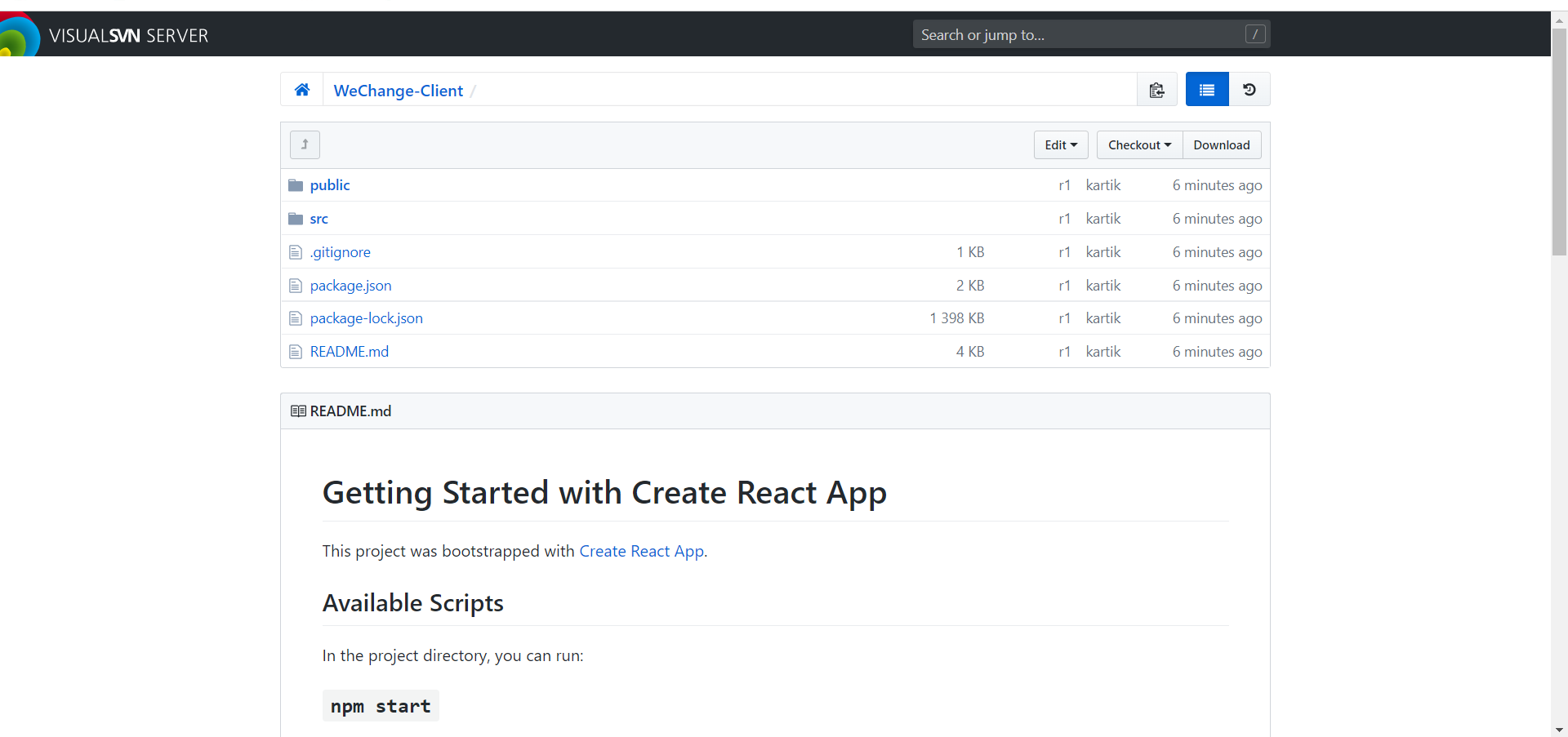


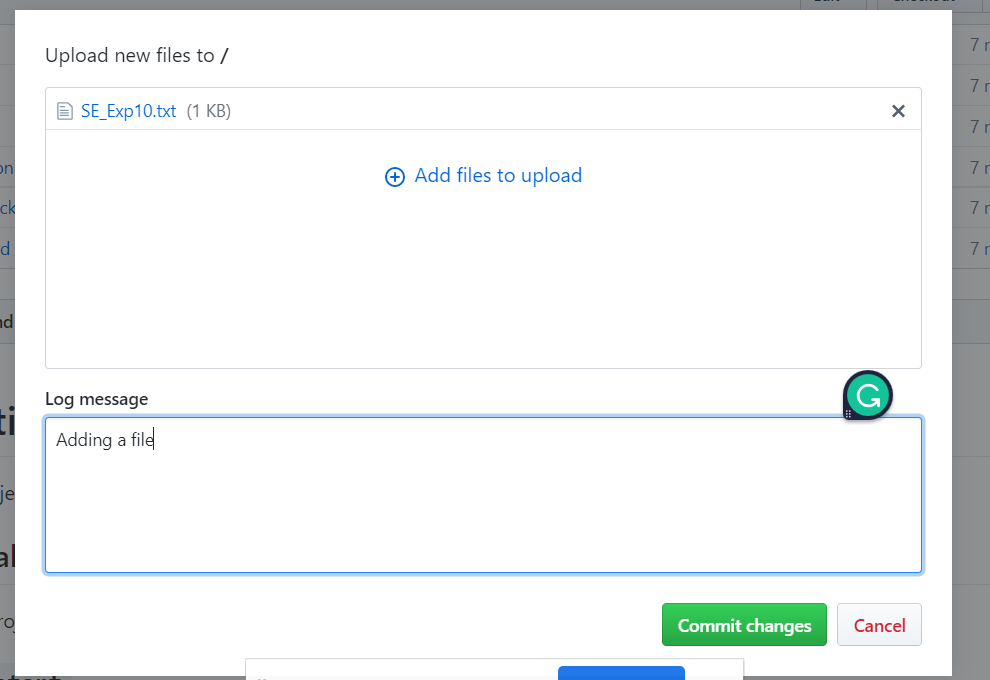
Committing files in Server folder in VisualSVN servers

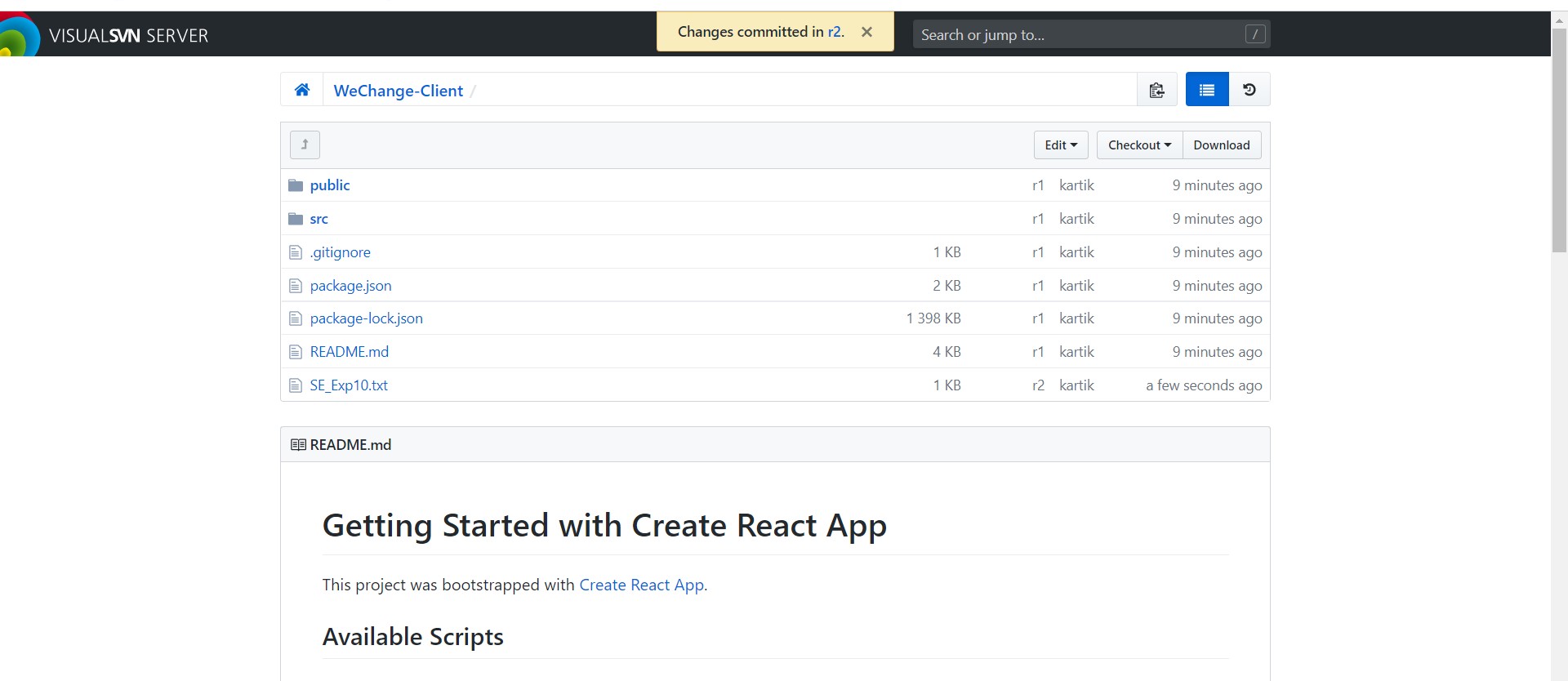




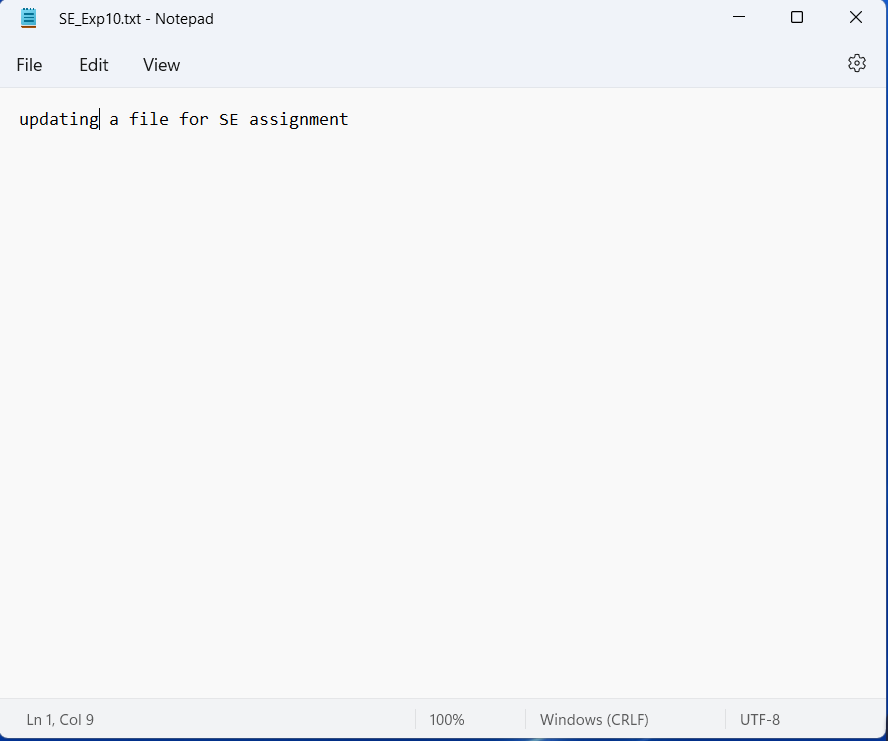


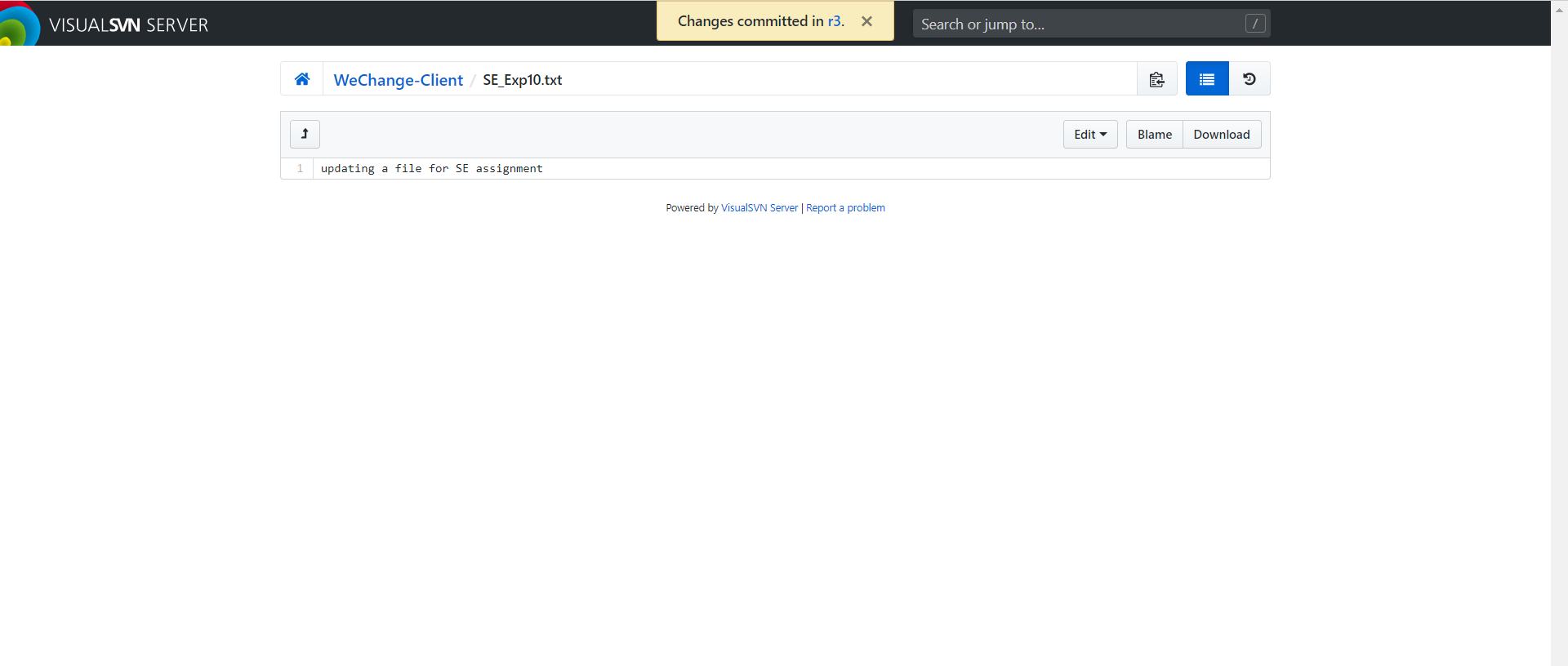


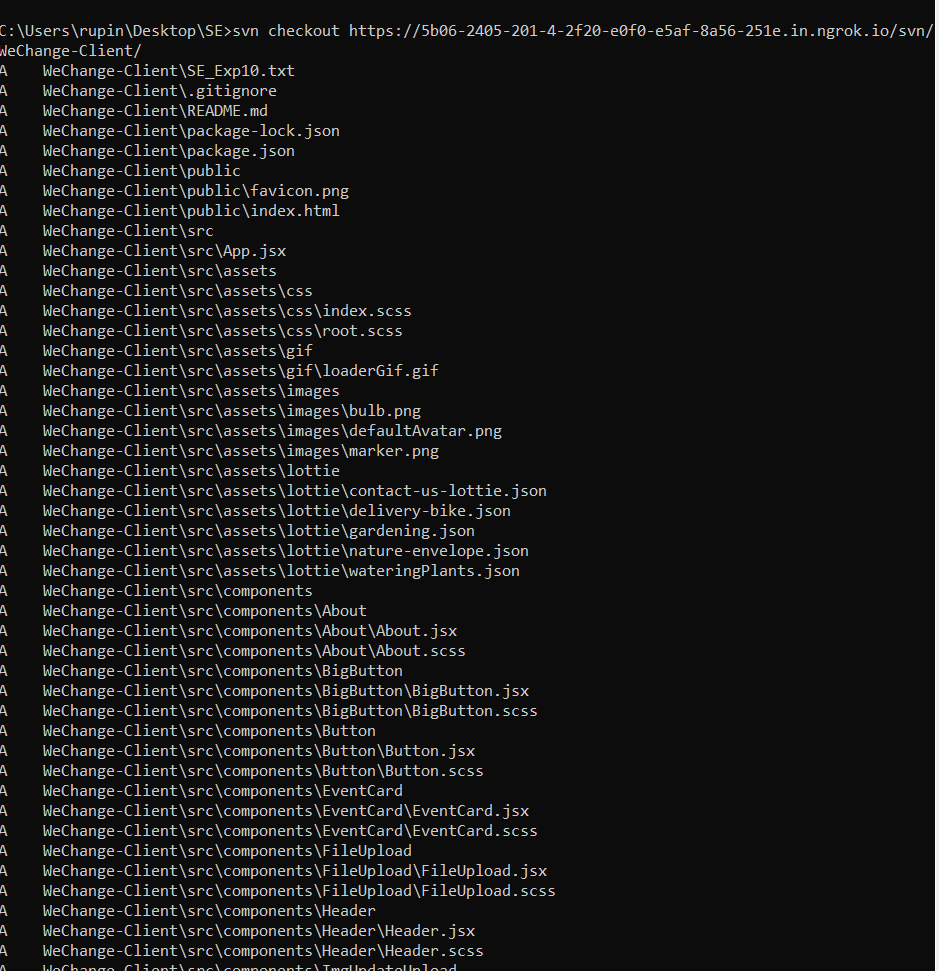
Adding a file:

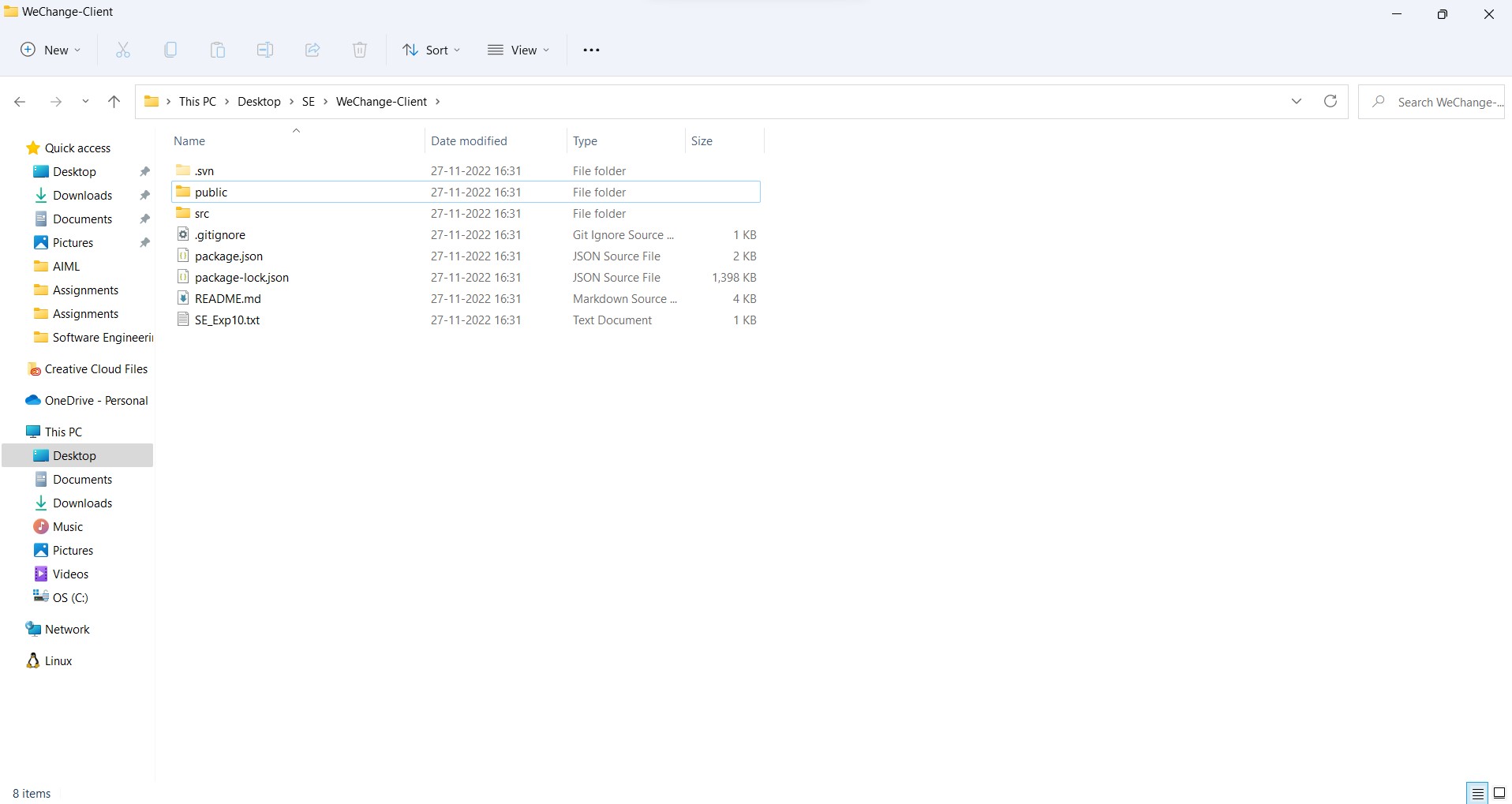


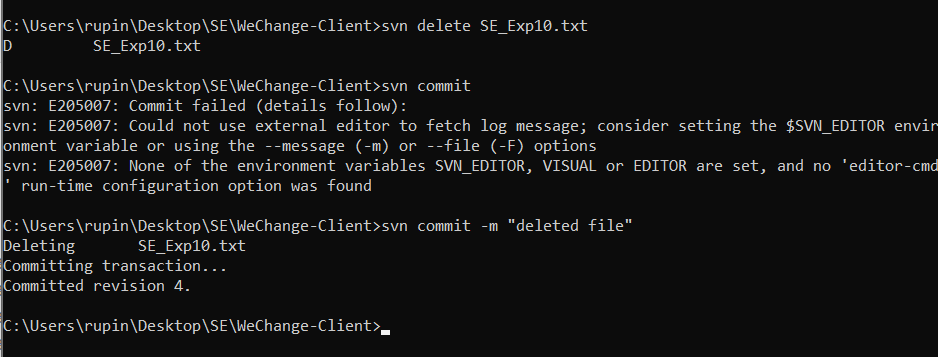
Updating the text file:

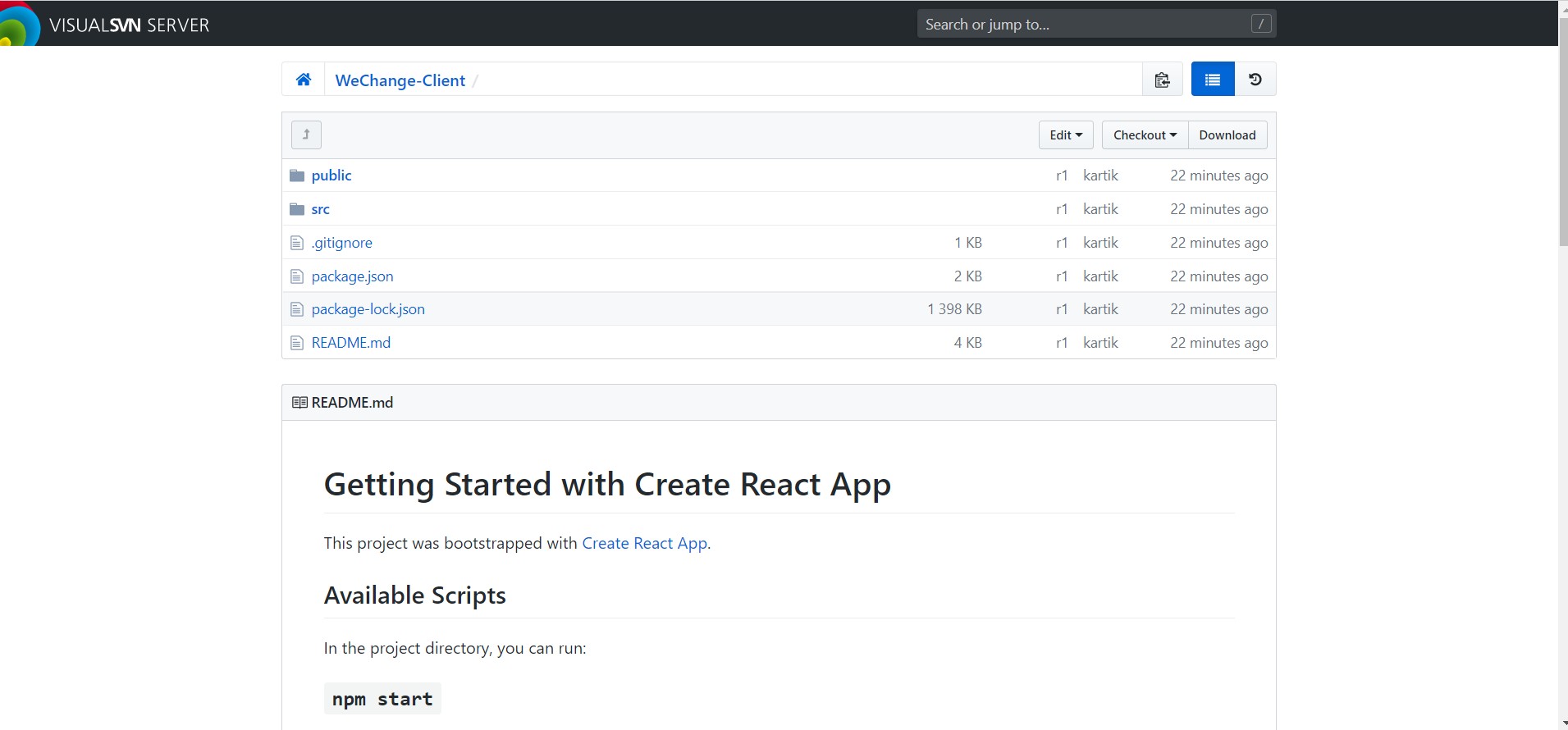


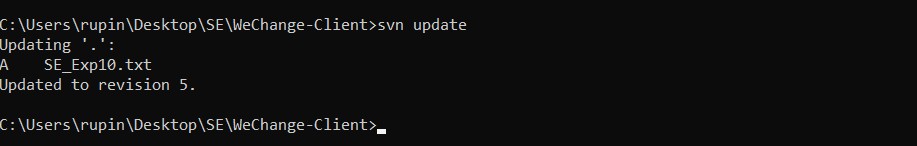


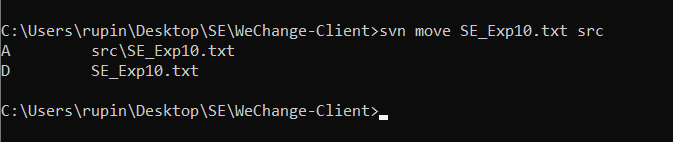
SVN Checkout client:

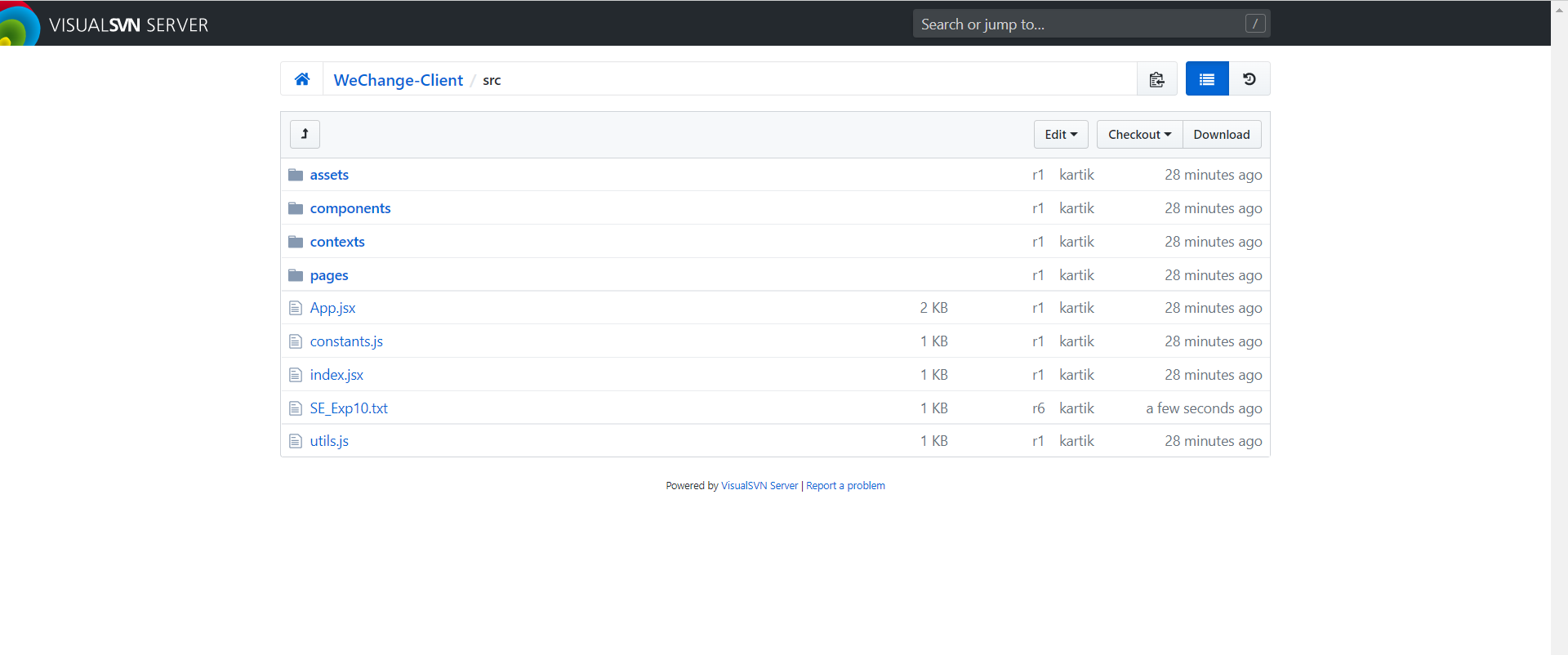


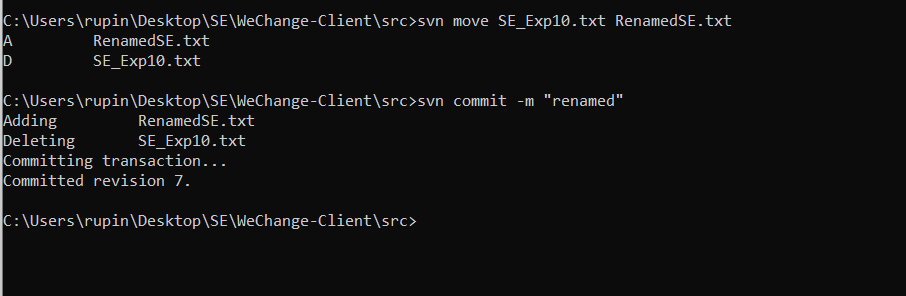
Deleting a file:

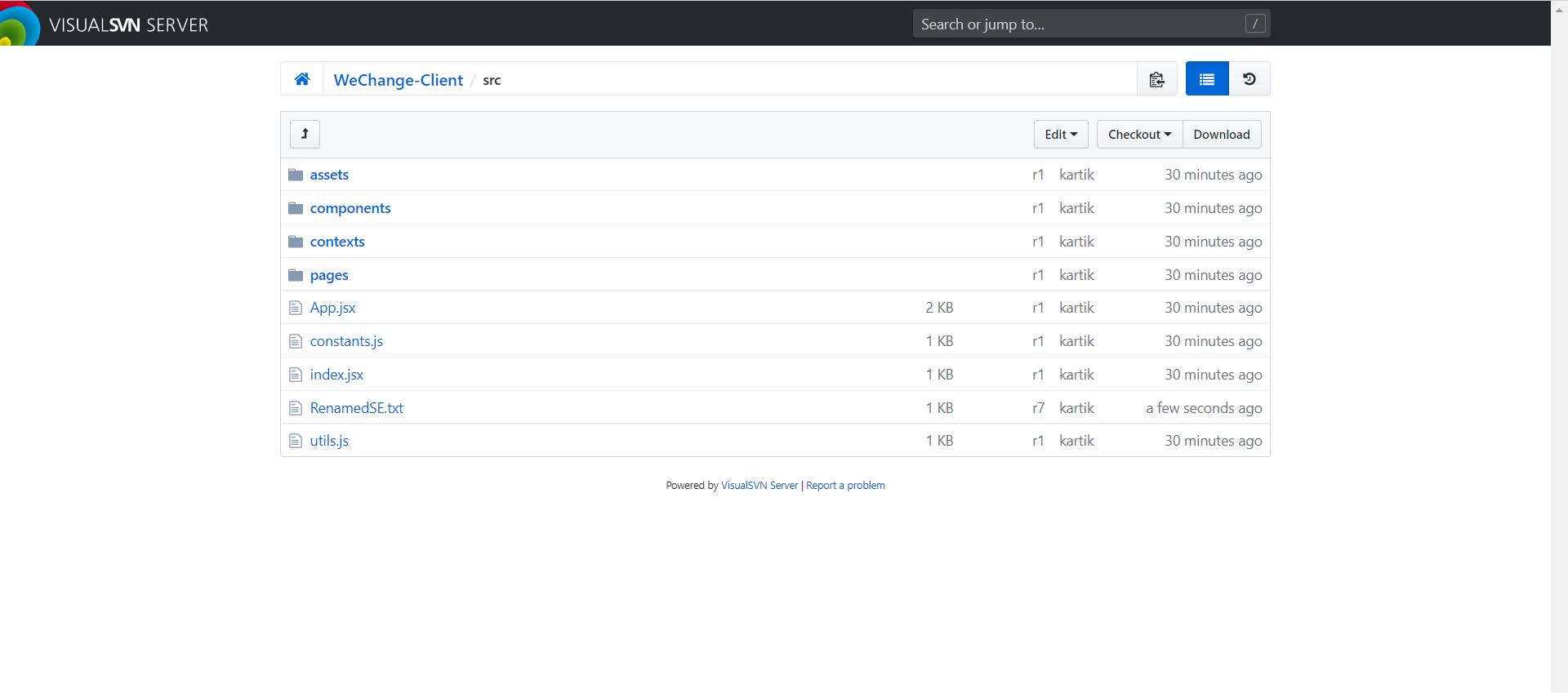


Updating:

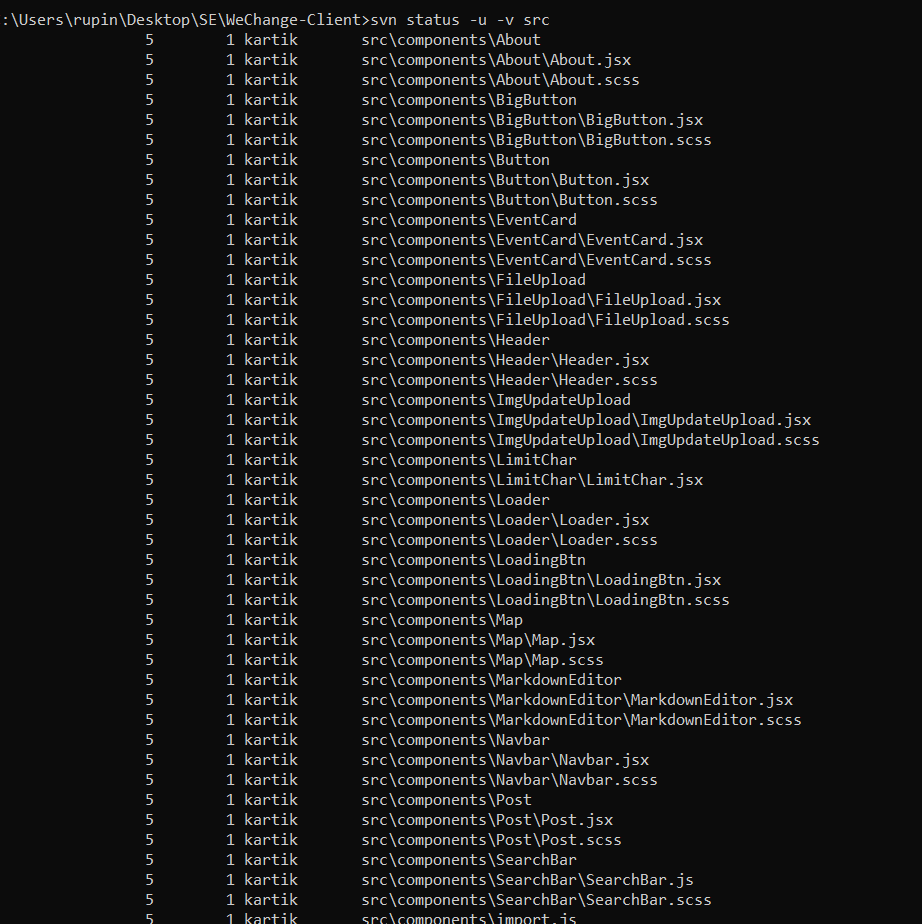
Moving:



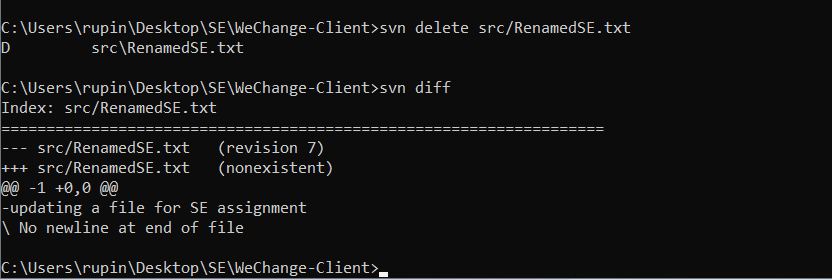
Renamed

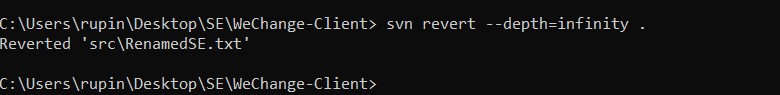


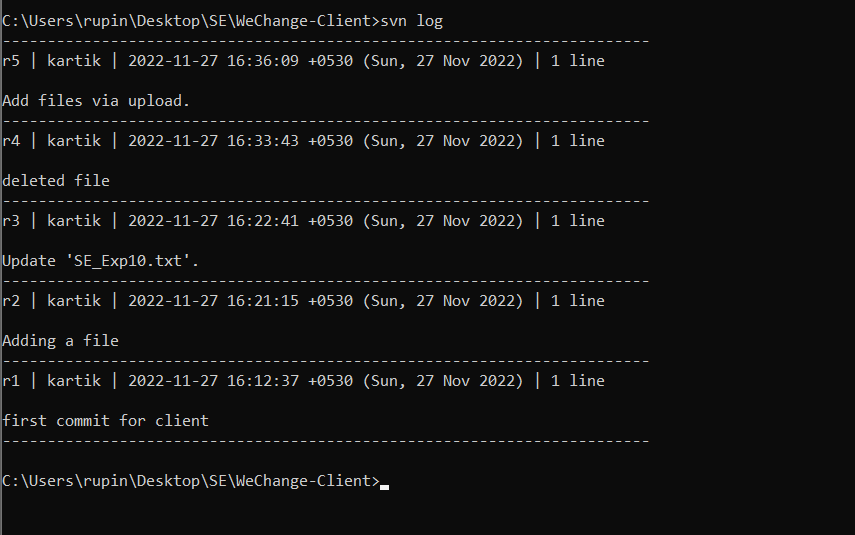
Svn status:



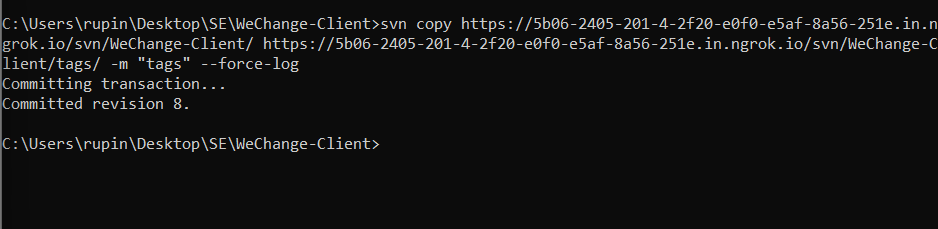
Svn diff:

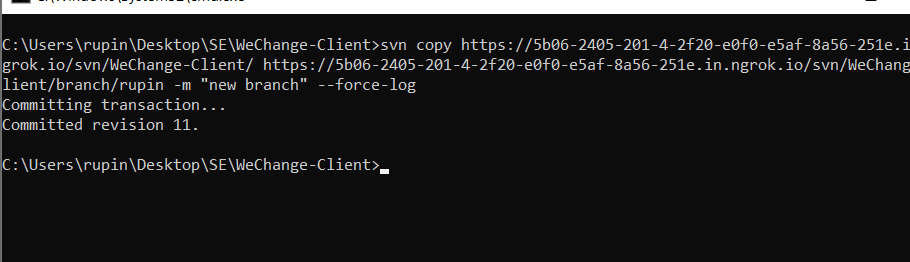


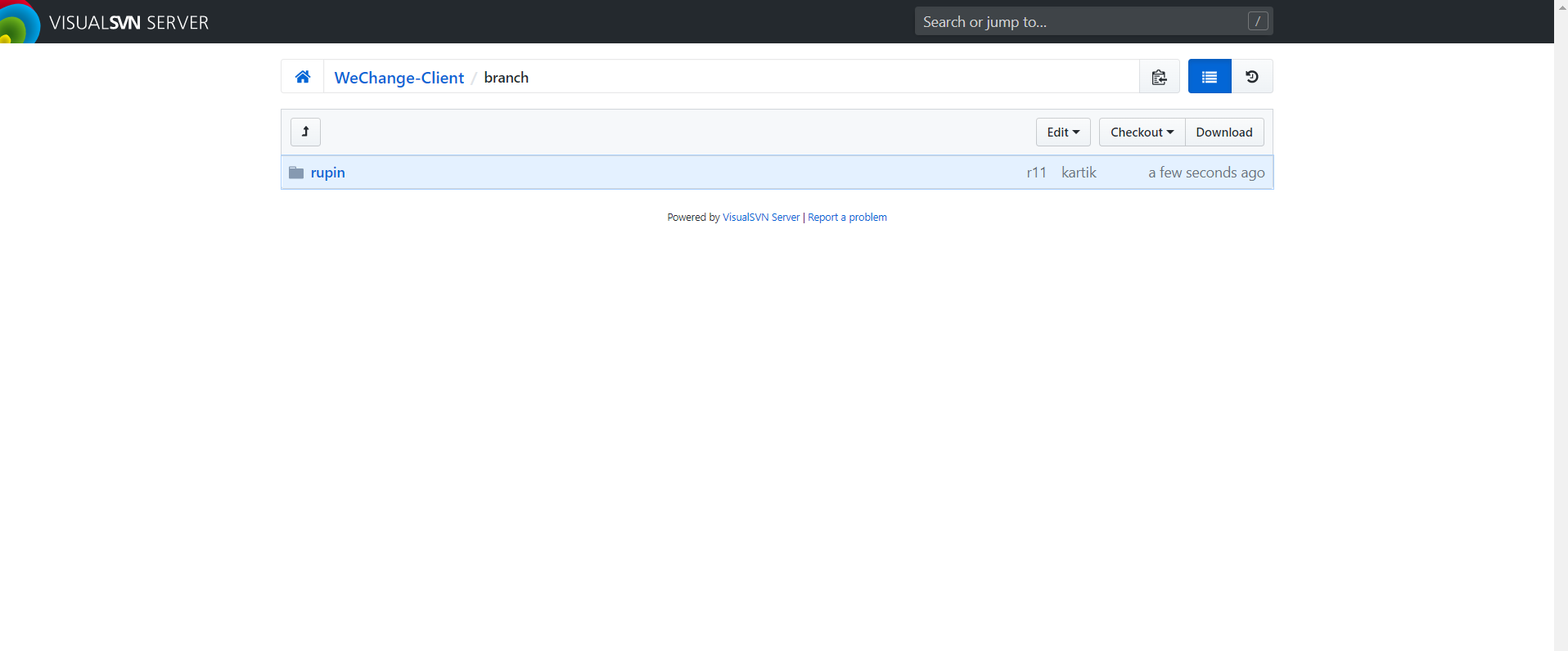
SVN Revert:

SVN Log:

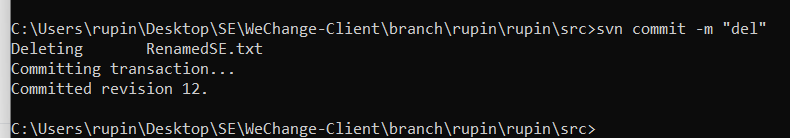
SVN Tags:

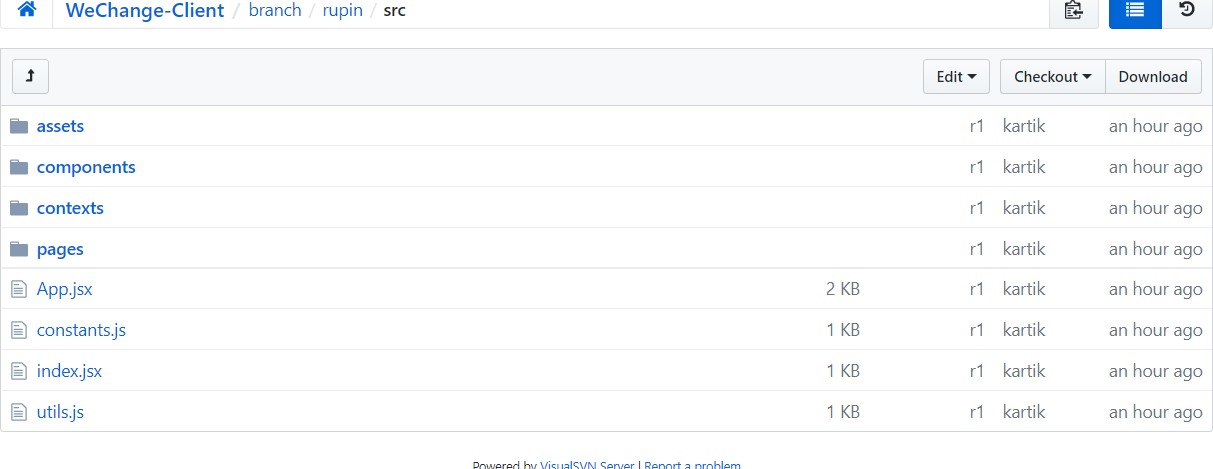


SVN Branch:

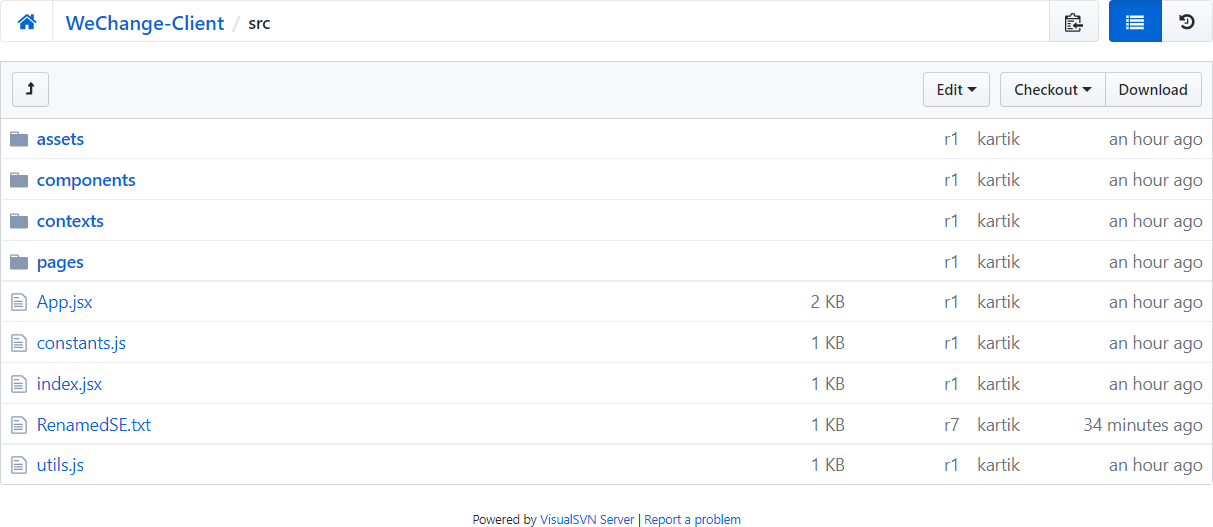


Deleting in branch:

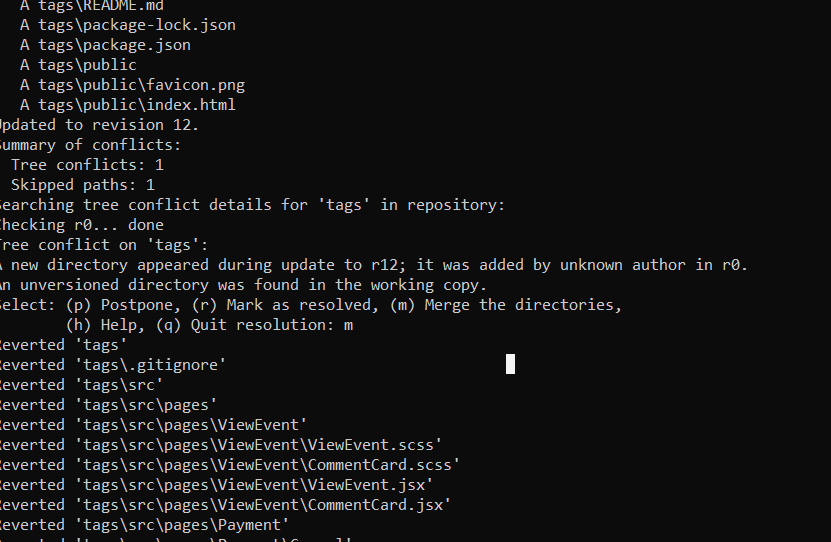


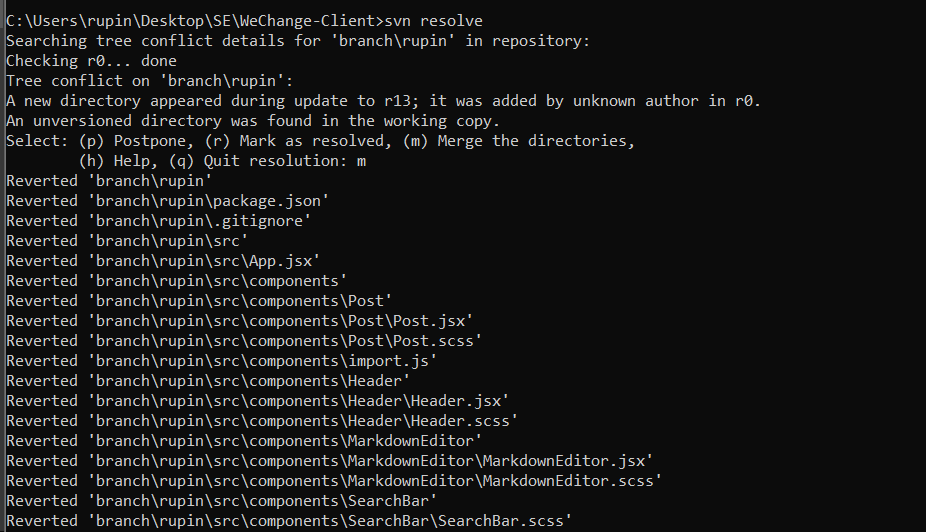


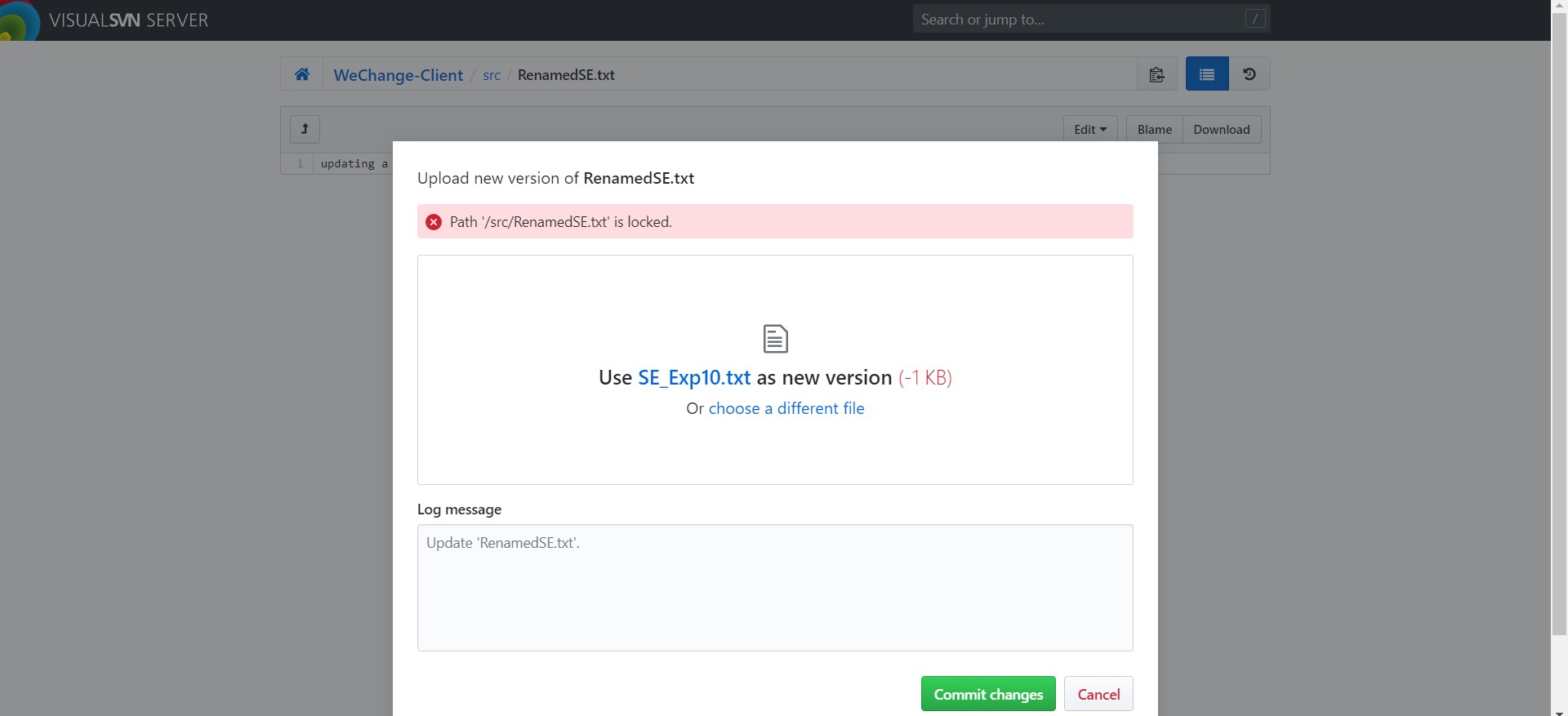
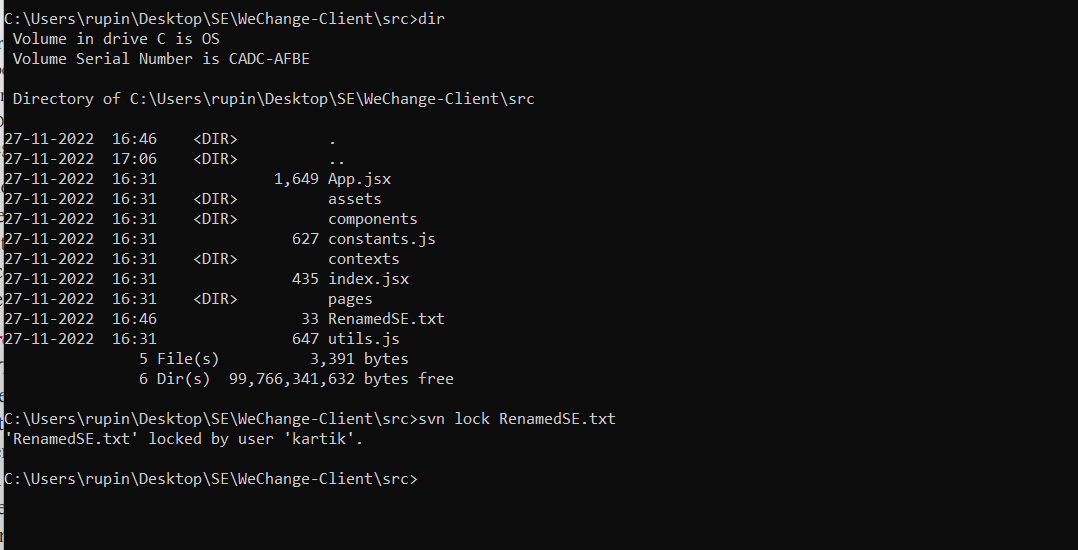
Still there in root folder:



Merging :



SVN Resolve:

SVN Lock

## Conclusion:

* + Successfully installed VisualSVN Server and Tortoise SVN
  + Created 2 users with password authentication
  + Created 2 repositories on VisualSVN Servers, one for website client and other for website server
  + Cloned the repositories on a different machine using ‘ngrok’
  + Performed the following operations: checkout, commit, update, add, edit, delete, rename, move, status, diff, revert, log, tag, branch, merge, resolve and lock
  + Checked whether the changes were reflected on the original machine