Bachelor's thesis

Information and Communications Technology

2021

Markus Vuorinen

BASICS OF A REMOTE DEVELOPMENT ENVIRONMENT

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2021 | 30 pages

Markus Vuorinen

BASICS OF A REMOTE DEVELOPMENT ENVIRONMENT

The need for working remotely is ever-growing in the field of software development. The technology needed for this already exists and there are multiple competing services. However, many smaller companies still struggle to build even the most basic solutions for it or do not even pay attention to it.

This thesis aims to research and explain the basic concepts related to a remote development environment, making it easy to understand and briefly go through the most important steps when implementing the findings from the research in a real-world environment. The research carried out in this thesis used the author’s prior knowledge on the subject as a basis and expanded that to suit the needs of the thesis when needed. Important subjects within the thesis include version control systems, workflow, server security, documentation, style guides and communication. In the practical implementation of the remote development environment, the thesis briefly examines essential tasks such as settings on the core software and how to implement the continuous integration functions.

In the course of this thesis, it became obvious that many of the technologies and methods used in the thesis are useful for any development environment and there is no reason not to implement them to an existing one or create a completely new one based on this thesis. The modern solutions for setting up and using a remote development environment have improved significantly compared to the author’s prior experiences on the subject.

The thesis concludes that the subjects discussed in the thesis cover the most important parts for understanding how to set up a remote development environment. Subjects such as version control systems and server security demonstrate that the existing systems and knowledge can be used as-is or with minor changes while documentation and communication focus on the developer’s side of collaborating remotely. Finally, the subjects on workflow and the development environment itself demonstrate the structure of the system and how to create one.

KEYWORDS:

git, software development, remote work, continuous integration

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintäteknologia

2021 | 30 sivua

Markus Vuorinen

ETÄTYÖKEHITYSYMPÄRISTÖN PERUSTEET

Tarve etätyöskentelylle kasvaa ohjelmistokehityksen alalla. Tarvittava teknologia on jo olemassa ja tarjolla on monia kilpailevia palveluita, mutta monet yritykset yrittävät rakentaa kaikkein yksinkertaisempia ratkaisuja tai eivät edes kiinnitä asiaan huomiota.

Opinnäytetyön ensimmäisenä tavoitteena oli tutkia ja selittää peruskäsitteet, jotka liittyvät etätyökehitysympäristöihin, ja tehdä niistä ymmärrettävämpiä. Toinen tavoite oli lopuksi soveltaa aikaisemmin kerättyä tietoa kehitysympäristön pystyttämisessä. Tärkeitä aiheita opinnäytetyön tutkimuksessa on versionhallintajärjestelmät, työnkulku, turvallisuus, dokumentointi, tyyliohjeet ja kommunikaatio. Kehitysympäristön pystyttämisvaiheessa opinnäytetyö käy ohimennen läpi tärkeimmät vaiheet, kuten versionhallintajärjestelmän asetukset ja jatkuvan integraation toiminnat.

Työn tuloksena selvisi, että opinnäytetyössä tehtyä tutkimusta voi soveltaa uuden etätyökehitysympäristön rakentamisen lisäksi muihin kehitysympäristöihin. Tehdyn tutkimuksen käytännöllisyyttä demonstroitiin pystytettäessä kehitysympäristöä.

ASIASANAT:

git, ohjelmistokehitys, etätyö, jatkuva integraatio

CONTENTS

FIGURES

LIST OF ABBREVIATIONS

CD	Continuous Deployment
CI	Continuous Integration
COVID-19	Coronavirus disease 2019
CVS	Concurrent Versions System
IBM	International Business Machines Corporation
OP	Osuuspankki, a Finnish bank
SSH	Secure Shell
VCS	Version Control System
VPN	Virtual Private Network
YAML	YAML Ain’t Markup Language
.yml	File extension of YAML file

1 INTRODUCTION

Remote work has been on the rise in software development jobs for over a decade since the writing of this thesis and has seen its sharpest rise in the last couple of years. According to research carried out by GitLab on people working remotely, 56% of the workers taking part in this research had been doing remote work for less than a year and 21% from 1 to 5 years. (GitLab, 2021) This already rising trend saw a massive increase during the COVID-19 pandemic in 2020 and is still not showing any signs of stopping in 2021. In the United States in April 2020 nearly half of the workers worked from home for 5 or more days a week. (Herhold, 2020)

The pandemic has shown companies that remote work is the future of work. Many companies such as Dropbox, Salesforce, Slack, Spotify, Twitter, and VMware have announced their plans to switch to remote work permanently and others like Facebook have decided that half of their employees will work remotely starting from 2021. (Courtney, 2021) Besides, in companies that have started to move from the traditional 9-to-5 office work hours, nearly half of the employees that have started working remotely prefer it over working in an office with the main reasons being the lack of commute and more flexible schedules. (Herhold, 2020)

With such a large increase in the remote workforce for software development, the need for new remote development environments rises and although the technology is not new and has had plenty of time to mature, the new adopters of remote development technologies have to learn how to use them. The first goal of this thesis is to gather the basic information on what would be important to know before setting up a remote development environment. This environment should be a secure and easy to use remote development environment that does not slow down the workflow compared to a locally done development. The second goal is to use that knowledge to set up a barebones remote development environment on an existing development server so the development can start remotely and briefly going through the important steps needed for the setup.

The thesis divides the information gathered for the first goal into separate subject categories. Chapter 2 first introduces version control systems that work as a basis for the environment and explains the choices made for the practical part of thesis regarding it. Chapter 3 goes through the project workflow and Chapter 4 addresses the importance of good server security. Moving from the back-end and software to the developer, Chapter 5 goes through the documentation and style guide of the project being developed. Chapter 6 discusses why communication is so important when collaborating remotely with others and how it could be implemented. The practical part of the second goal is addressed in Chapter 7 and the thesis ends with a conclusion in Chapter 8.

The thesis uses diverse sources from software's official documentation, reputable website articles, and research conducted by large companies. In addition to the outside sources, the author has also used his over 3 years of work experience in the field.

2 VERSION CONTROL SYSTEM

A version control system (VCS) is the most important part of a development environment. It is a "*system that records changes to a file or set of files over time so that you can recall specific versions later.*" (Chacon, 2014) This is important since the developer does not have to think about losing progress if any mistakes are made and just revert back to a working version if some experiment did not work.

2.1 Common version control systems

The two most common types of VCS are centralized version control systems such as CVS, Subversion or Perforce, and distributed version control systems such as Git or Mercurial.

2.1.1 Centralized Version Control System

The Centralized Version Control System works by having one centralized server that holds all the code and version control history for that code as shown in Figure 1 below. From that server, multiple developers can check out files to their computers, work on them and submit them back to the server. This creates a serious single point of failure. If the connection to the server is lost, no one can work on anything else other than what they had already checked out and in case of data loss, everything else except small pieces of code that developers had on their computers is lost.

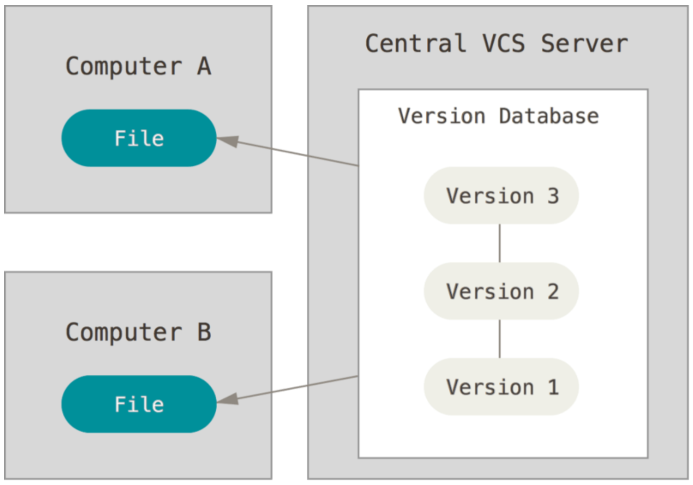


Figure 1. Centralized Version Control (Chacon, 2014).

2.1.2 Distributed Version Control System

In a distributed version control system, every developer fully mirrors the whole repository and its version control history as shown in Figure 2 below. This acts as a full backup in case of data loss in the main server and it enables a different kind of possibilities in development when collaborating with others on the same code at the same time.

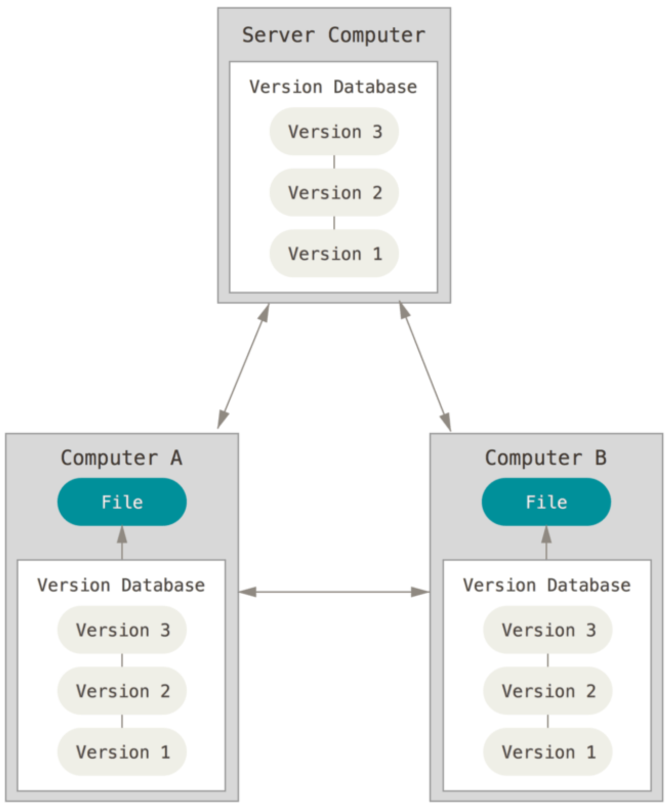


Figure 2. Distributed Version Control (Chacon, 2014).

2.2 VCS of Choice and Software

Due to earlier experience and features such as offline availability and faster commits (Backlog, 2020), Git was chosen as the VCS of choice. Earlier experience is based on using the Gogs Git-repository service that is a lightweight software program with web-based graphical UI and user management tools. However, it alone did not meet the requirements for the new development environment, thus requiring to either supplement lacking Continuous Integration and Continuous Deployment (CI/CD) functionality with other software, such as Jenkins; or switching to another Git-repository service with those functions built in.

For this thesis GitLab was chosen as it has all the needed functionality to build and manage Git-repository with multiple developers and CI/CD pipeline effortlessly. It also has multiple other functions not listed as requirements that can help enhance the development environment in the future, such as integrations for development tools already in use and many options for easier introduction and integration of new tools into the system in use.

3 WORKFLOW

A good workflow is an important part of an efficient development environment. It can be divided into small parts that make sense as a single task thus making everything easier to understand and manage. Developers always know what task is next on a workflow and all the important tasks get done.

3.1 Basic Git Workflow

Git uses a branching model that allows the creation of a copy called a branch from the main source code that is being developed called a master branch. This: “*encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.*” (Git, 2021) making it easier to try new things without affecting any of the working code. When adding the code on a new branch to the master branch, those branches need to be merged.

When starting to use Git, a repository – where the source code being developed is stored – needs to be created or an already existing repository taken into use. When creating a new repository some code needs to be added into that repository. After writing some code, it needs to be tracked within the version control system by “adding” the code to the system. After the code is being tracked, any committed changes made to it to take a snapshot of the code. This enables a comparison of differences between the new and old versions of the file and when needed an old version can be loaded. When a piece of code is ready to be added to the branch being worked on a “push” is needed to add the code to it. When starting by using an already existing repository or updating a local branch with changes made by someone else, just “pull” the new code to the local branch and start making commits. Figure 3 below demonstrates how to get code to and from the Git repository.

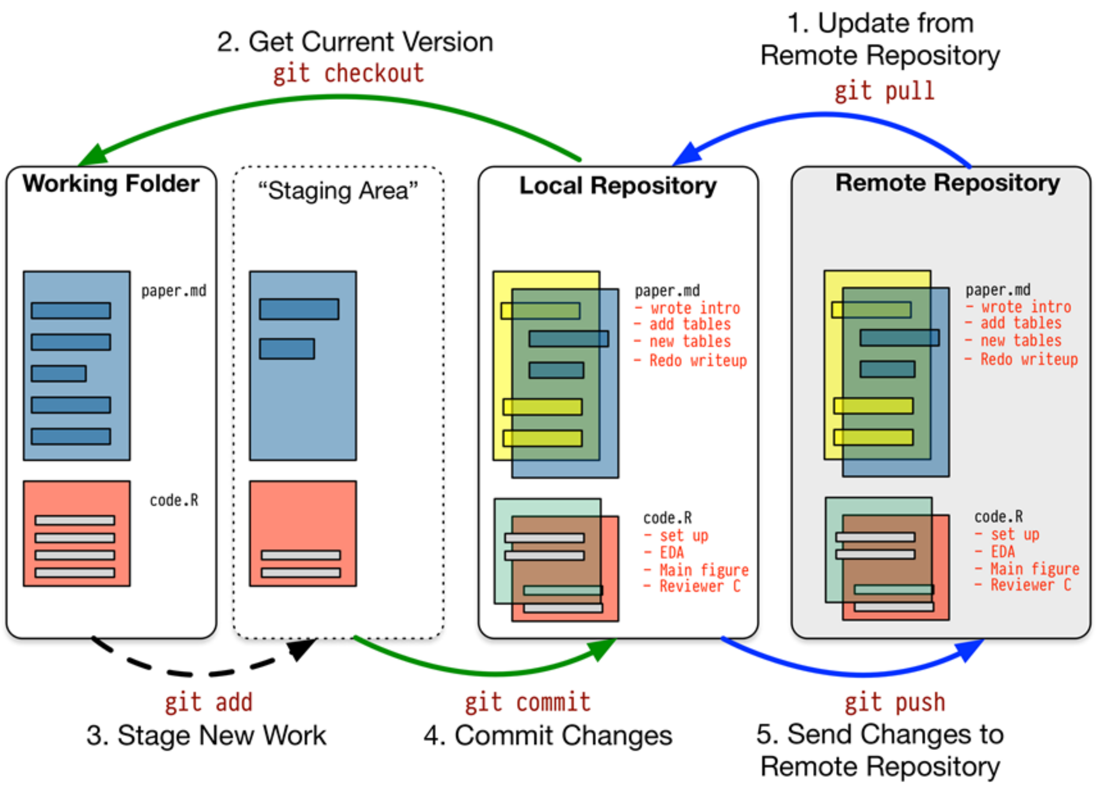


Figure 3. Git Workflow (Ikeanyi, 2018).

3.2 Continuous Integration

When more than one developer is working on the same project, some errors are bound to happen. To catch these errors code needs to be tested every time someone pushes a piece of code. That is why continuous integration is so important. "*Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project.*" (Atlassian, 2021)

By automating the building and testing of the code, a large amount of time can be saved and by writing automated tests for the code. Those tests can be run at any point in time to make sure the whole project works or to help with debugging errors. Automation also enforces consistency in getting all the important tasks done in order when tasks can not be skipped and everything leaves a mark on VCS.

3.3 Code Review

After automated tests of continuous integration, it is important to take some time for code review where another group member reads through the code to make sure it is readable, catch any errors that were not tested, and give feedback so both parties can learn from it. After all possible mistakes are corrected and the code is approved, the code can be merged to the branch meant for deployment.

3.4 Continuous Deployment

The last phase of the workflow is continuous deployment (CD). During it, the code is built for production, tested one last time with production-related tests and possibly reviewed before waiting for human confirmation to be pushed to production. Figure 4 below shows the whole CI/CD pipeline and its order.

The tests usually include running it in a test version of a whole production environment, performance tests, and testing larger parts of the system such as network, containers and database.

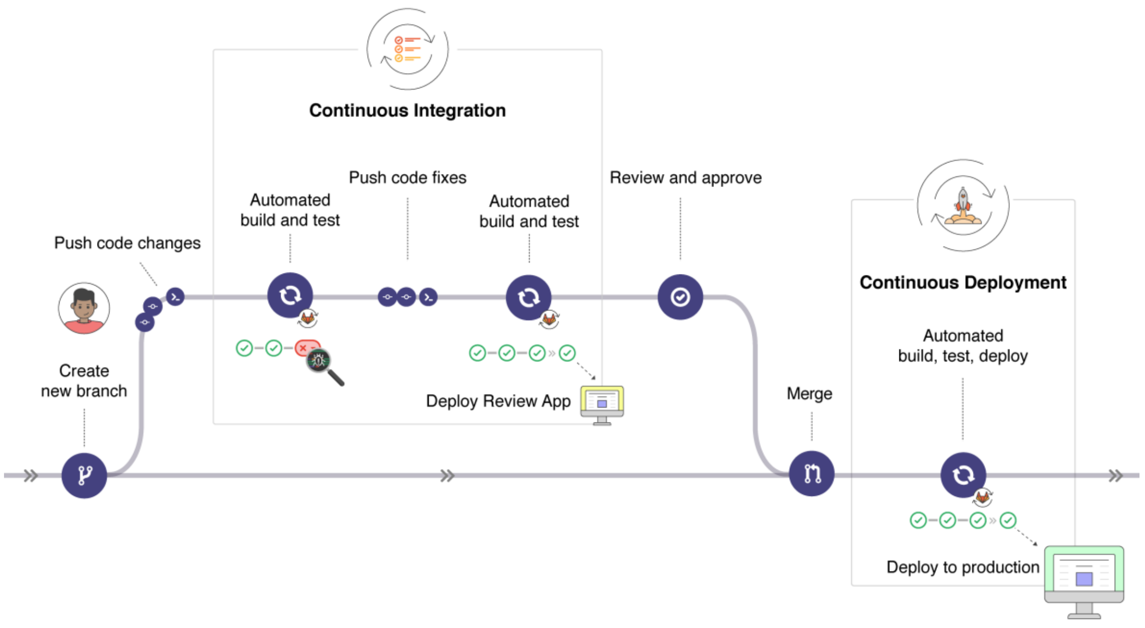


Figure 4. CI/CD Workflow (HowtoForge).

4 SECURING THE SERVER

4.1 Importance of security

One of the issues that can have the biggest impact is security. If the connection to the server or server itself is compromised, an unauthorized user can inject malicious code into the project without anyone noticing it or access user information. According to IBM's worldwide research: "*a majority of organizations (76%) predicted that remote work would make responding to a potential data breach a much more difficult ordeal." and it is "expected to increase data breach costs and incident response times*" during COVID-19. (IBM, 2020) Within Finnish companies associating cybersecurity with significant risks has increased from 57% in 2014 to 87% in 2019 and 2020. Increased remote work and the challenges that it brings with it have been pointed out as a reason in 2020. (OP Ryhmä, 2021)

4.2 Basics for securing the server

Securing the server is the most important part to get right the first time. If other aspects of the development environment are misconfigured, the worst thing that could happen is project costing more time and money. But if security fails then sensitive data could leak to malicious entities, the public image could suffer and a data breach could end up costing vastly more than some problem in the project. (IBM, 2020) Some important steps that can be taken to secure the server are:

- **Restrict access to the server back-end** to only trusted people who maintain the server. Project members do not need to access other software than their own tools.
- **Requiring SSH key authentication** for accessing everything else than developer tool front-ends. SSH is an encrypted protocol for communicating with the server. Using SSH keys for authentication is more secure than the conventional password.
- **Using firewall** software controls how services are exposed to the network.
- **Keep the system up-to-date** is very important as new security patches for vulnerabilities are released to keep the server secure.

- **Backup frequently** because data can be lost when physical components fail, a malicious actor gets into the system and execute a ransomware attack or some other accident can lead to a data loss.
- **Setup a dedicated VPN** so the developers can securely access their development environment without exposing it to the public network. Installing and configuring VPN software to the development server and requiring developers to install and use VPN client software creates a secure tunnel between the development server and the developer’s computer. (Ellingwood etc., 2015)

A VPN makes connecting to the development environment secure and easy, but it might not always be enough depending on how sensitive the secret project being worked on is. Additional ways to secure the system might include restricting where the developer can connect to the development environment or what software can be installed on the client computer. This adds more steps that need to be taken before the developer can access the work making it less easily accessible but making the system overall more secure. It is important to find the right balance between how secure and how accessible the desired system is.

4.3 Implementation for this Thesis

According to the server documentation and further inspection, the server used in the thesis is already secured by only the system administrator having access to the server back-end through an SSH connection to a custom port, only SSH custom port and VPN services are allowed through the firewall, updates and backups are done during the weekly inspection of the server, and the VPN used is WireGuard.

Remote connection to the server’s back-end is restricted to only one computer meant for server maintenance tasks. If something happens to the computer and the SSH key is lost, the only way to have administrative access to the server is by creating a new SSH key for the new computer locally. Since this way of connecting is rarely used aside from weekly check-ups, it does not need to have easier access. The current level of security has been deemed satisfactory.

The developers are required to use only computers dedicated to working to limit possible attacks through non-work-related activities. WireGuard has client software for all the major operating systems and they are easy to install and use. Only having VPN as an only additional manual task for developers is a minor inconvenience and dedicating a computer to work keeps it more organized and thus might increase productivity.

5 UNDERSTANDING THE CODE

After some time – when the code's author has forgotten what was written – it can be hard to understand even their own code if it is not simple, well documented or easy to read. When collaborating on a project with multiple other people it can be even harder to understand what a colleague meant on a piece of code. This leads to unnecessary work of trying to find out what that piece of code does by either taking time deciphering it or by getting the original author to explain everything.

All of this can be significantly mitigated or completely eliminated by setting rules and guidelines on how to write readable code and documenting everything in a clear language.

5.1 Documentation

Clear and understandable documentation of code, group policies and used tools is important for any project. It helps the code's author to remember what a piece of code meant, quickly replicate the installation process or debug malfunctioning software, or onboard new members into the project.

Software developers working in a group should have common guidelines on how the documentation is written and it should be peer-reviewed by another group member to make sure others understand it as well. This helps to keep documentation quality understandable. There is nothing worse than unintelligible documentation for complex code.

The general documentation should be collected into one place that can be referred to or copied from when needed. Good places to write the project's documentation and keep it organized are services such as internal wiki, project documentation software or a user-managed website.

When writing code, it is part of the best practices of documenting and structuring the code's file structure to have a README file at the project's root folder. README file should be written either in plain text or markup languages such as Markdown or reStructuredText and should contain a basic summary for the code, how to install it and some other general information. This information should be added to the main documentation also to make it easier to find in case of not having access to the project files. Another important way of documenting code is by using comments in the code itself if this is supported in the programming language. A code comment should be short and descriptive and written somewhere where it is easy to find the piece of code that the comment explains.

5.2 Style Guides

Although "*every programming language has its own set of stylistic conventions that are meant to promote readability*" (Miami University, 2021) – when programming – a developer can have their own style of writing code and an idea of how readable code looks. To help keep the code readable and consistent regardless of who writes it, a style guide is needed. "*A style guide contains a set of standards for writing and designing content.*" (Write the Docs, 2021) These standards can describe anything from the smallest detail to how large pieces of code need to be formatted.

5.3 Enforcing the Use of Style Guides

When writing code, the style guide's standards for that code can be enforced manually by peer-reviewing the code and automatically by adding integration approval steps for syntax checkers and code style formatters to the continuous integration phase of the project's workflow. (Atlassian, 2021) For the best effect, both manual and automatic options should be used during continuous integration. Automated syntax checkers and code style formatters help the developer to find more possible errors in the code before they find their way to the peer-review phase and need another person to get involved to solve those problems. This helps the developer to learn to write more readable and consistent code and errors that can not be found during automated tasks can be solved together during peer-review.

5.4 Documentation and Style Guide for the Project

All the documentation for the development environment will be written into a self-hosted internal wiki page. It holds all the information on group policies, used software and code documentation. Since the programming language chosen for this project is Python all the code packages have a README file in the root folder and Python’s own comments are used to document parts of the code.

After comparing style guides for Python, such as Python’s own style guide and Google’s style guide for Python it was decided that Google’s style guide would be used. To enforce the style guide syntax checkers such as Mypy and Flake8 and code style formatter isort are used. Mypy is a static type checker, Flake8 is a good general syntax checker and isort formats Python imports.

6 COMMUNICATION

Communication is an important part of collaborating with others. When doing remote work the importance of software used for communication grows and team members need to be able to get a hold of each other during working hours.

6.1 Why invest in communication

When investing in other parts of the project workflow and required tools, it is easier to see the return of investment. If that part is missing, work does not get done. But when investing in communication tools, the benefit is not as clear. If group members need to get hold of each other, they will find a way by using any tool even if it is not the most efficient method or they just do not communicate. The harm this causes can be hard to see since everything works, but just slightly less efficiently. According to research, when put into numbers disengaged employees have 37% higher absenteeism, 18% lower productivity and 15% lower profitability amounting to a loss of roughly third of the employee's annual salary. (Borysenko, 2019) This means that a loss caused by a Finnish employee working in an IT job can cover around 200 employee's licenses for good business communication software like Slack. (Palkkadata, 2021)(Slack, 2021)

6.2 Right tools for the right task

It is important to choose the right tool for the task no matter whether it is programming or communicating with colleagues. By using proper video conferencing software groups are able to hold meetings as they do in the company's office, but it does not mean it will have the same effect. Organizing a quick meeting to discuss a small matter often can take more time remotely than it would take in the office where everyone is in the same space and just have to move to another room briefly. Sending larger and more official documentation by email is better, but if the matter is urgent and a more nuanced discussion with a colleague is required then the business communication software is a better option. (Hummerston, 2020)

As opposed to a video conferencing tool's and email's straightforward requirements, a business communication software can become an important tool that boosts efficiency a lot if the right one is picked. A good business communication software has a suite of different functions that not only cover communicative functions but also collaborative and engagement oriented functions too. It should have a user-management functionality to restrict access to only authorized personnel and it should be managed by a trusted person in charge of system management. Persistent chat rooms help keep the messages organized and on topic. Other features such as integrations to third-party services are a huge plus like integration for Slack and GitLab can be configured to send notifications whenever an important even workflow is done or when group member's attention is required. (GitLab, 2021)

6.3 Software of choice

For this thesis project, Slack was chosen because of prior experiences and already existing and easy integrations for new tools like GitLab. Slack filled all the set requirements and migrating from it to another service would have been too much unnecessary work with a possible steep learning curve.

7 THE DEVELOPMENT ENVIRONMENT

This chapter briefly goes through the order in which everything was set up for the remote development environment. It will only address setting up the GitLab and the CI pipeline as although important as a related concept, server security and communication software are not in the scope of the practical setup part of this thesis.

7.1 Setting up GitLab

GitLab supports multiple different installation methods such as normal installation to the operating system of choice or deploying a Docker container with everything in it. For this thesis, the normal method was chosen for its simplicity. Installation was easily done by following GitLab's official documentation.

After installation, the root password was changed and all the users were added to the GitLab. The process of adding users consists of:

- **Adding user groups** for easier user management.
- **Setting up permissions** for each user depending on how much access that user needs.
- **Adding an SSH key** for users, as that will be the method chosen to access the Git repository.

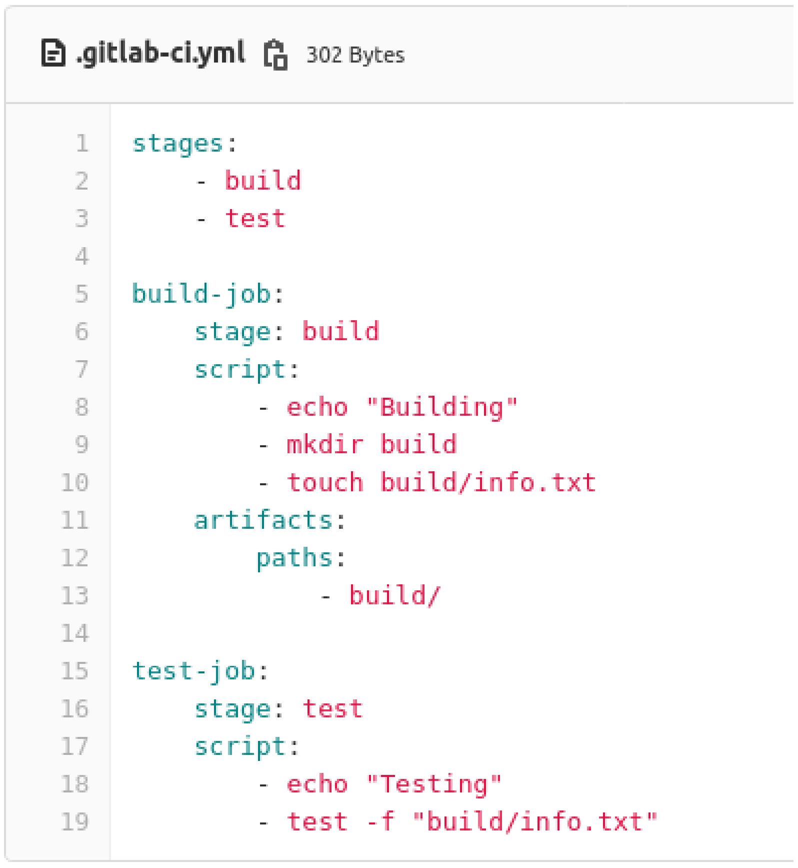
Lastly, a test project will be pushed to the repository to test that everything works so far.

7.2 Configuring CI Pipeline

GitLab has a built-in CI pipeline functionality. To enable CI pipeline functionality for a project in GitLab a GitLab runner and a configuration file for that CI pipeline is needed.

A GitLab runner is "*an application that works with GitLab CI/CD to run jobs in a pipeline*". (GitLab, 2021) The installation instructions for a runner can be found by navigating to settings in the GitLab user interface and selecting the CI/CD option.

After installing a runner, a configuration file called .gitlab-ci.yml can be created inside the root folder of the project. This configuration file contains the instructions on how to run the pipeline for that specific project. Figure 5 below is an example of that file used in the thesis to test that CI functionality works.



.gitlab-ci.yml 302 Bytes

```yaml
stages:
    - build
    - test

build-job:
    stage: build
    script:
        - echo "Building"
        - mkdir build
        - touch build/info.txt
    artifacts:
        paths:
            - build/

test-job:
    stage: test
    script:
        - echo "Testing"
        - test -f "build/info.txt"
```

Figure 5. GitLab CI Configuration File.

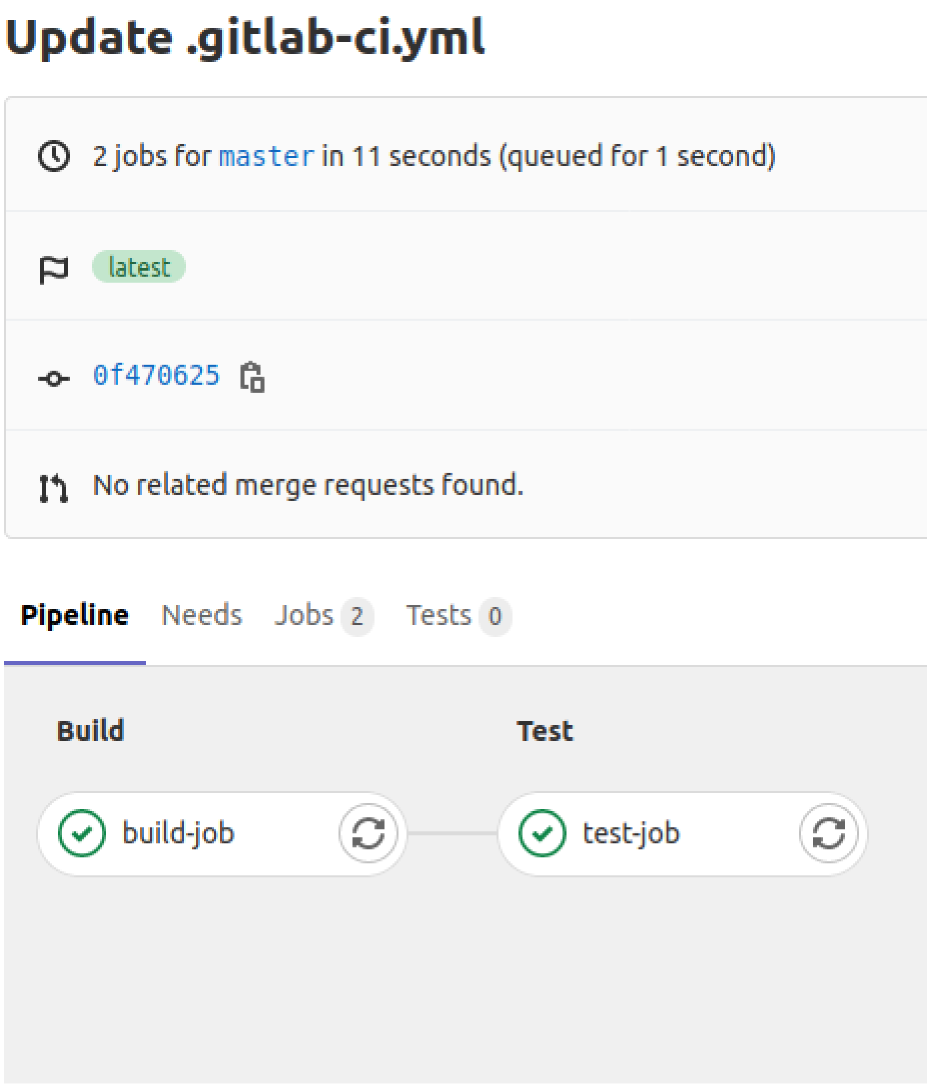
The stages category in the .gitlab-ci.yml file defines what stages need to be run. In this case, only the build and test stage is used.

After stages, the configuration defines jobs. Here there are two jobs: build-job and test-job. Both jobs contain a stage definition to tell at what stage of continuous integration it is run and a scripts section where terminal commands are run in order. The artefacts section in the build-job tells what artefacts will be passed on. Artefacts are files and directories produced that jobs create. When using different programming languages or developing different types of projects this configuration file is the only thing affected from this thesis.

The build stage is used to prepare the project so it can be executed. For example, programming languages that need to be compiled before executing the program will be compiled here. For testing that the continuous integration part works, the build stage only needs to run a couple of terminal commands to demonstrate it. The script only tells that it has started the build-job, creates a directory called build and a text file called info.txt inside it.

The test stage is used when running the tests such as unit tests in the code or syntax checkers. Here the script shows that it started the test-job and tests if the info.txt exists inside the build folder that was passed on from the build-job.

After committing and pushing the changes to the repository, the pipeline will start to run jobs automatically according to the instructions given in the configuration file. Under CI/CD option in GitLab’s navigation menu, there is a pipelines option that shows all the pipelines and their statuses. When selecting the pipeline for the commit, it shows the pipeline’s information. Figure 6 below shows that the build-job in the build stage and the test-job in the test stage both have passed successfully.



Update .gitlab-ci.yml

2 jobs for master in 11 seconds (queued for 1 second)

latest

0f470625

No related merge requests found.

Figure 6. Pipeline Information.

8 CONCLUSION

This thesis aimed to investigate basic concepts related to remote development environments and to help understand them. It also aimed to building a functional remote development environment that is simple and fast to build.

The study undertaken in this thesis should prove to be sufficient for a basic understanding of the subject and lay the foundation for further research. The breakdown for the practical set up shows how easy it is to create a new remote development environment. These systems are very well optimized, making them lightweight, and easy to use and install. With working trends showing change towards either fully remote or hybrid work conditions, there should be no reason not to create a remote development environment. Technology related to the subject is very mature and the author does not see it experiencing any major changes anymore.

The only issues regarding the change to remote work conditions are socioecological ones such as changing pay and lack of socializing. The technology itself is not a problem for new adopters.

As this thesis' purpose was to introduce the subject in a clear and concise manner, there are many ways to improve the remote development environment. The author of this thesis highly recommends to use this thesis as a reference when creating a new remote development environment.

REFERENCES

Atlassian, What is Continuous Integration?, Retrieved April 19, 2021. Available at: https://www.atlassian.com/continuous-delivery/continuous-integration

Backlog, June 23, 2020, Git vs. SVN: Which version control system is right for you?, Retrieved April 28, 2021. Available at: https://backlog.com/blog/git-vs-svn-version-control-system/

Borysenko, K., May 2, 2019, How Much Are Your Disengaged Employees Costing You?, Retrieved April 20, 2021. Available at: https://www.forbes.com/sites/karlynborysenko/2019/05/02/how-much-are-your-disengaged-employees-costing-you

Chacon, S. & Straub B., 2014. Pro Git, 2014, Apress

Courtney, E., February 19, 2021. 23 Companies Switching to Long-Term Remote Work, Retrieved April 20, 2021. Available at: https://www.flexjobs.com/blog/post/companies-switching-remote-work-long-term

Ellingwood, J.; Tagliaferri, L.; Camisso, J. & dbrian, March 5, 2015. Recommended Security Measures to Protect Your Servers, Retrieved April 18, 2021. Available at: https://www.digitalocean.com/community/tutorials/recommended-security-measures-to-protect-your-servers

Git, Branching and Merging, Retrieved on April 19, 2021. Available at: https://git-scm.com/about/branching-and-merging

GitLab, February 2021, Out of the Office, 2021, GitLab

GitLab, Slack Notifications Service, Retrieved April 20, 2021. Available at: https://docs.gitlab.com/ee/user/project/integrations/slack.html

Herhold, K., April 16, 2020. Working From Home During the Coronavirus Pandemic: The State of Remote Work, Retrieved April 20, 2021. Available at: https://clutch.co/real-estate/resources/state-of-remote-work-during-coronavirus-pandemic

HowtoForge, How to set up Gitlab for Continuous Integration and Deployment on CentOS, Retrieved April 18, 2021. Available at: https://www.howtoforge.com/how-to-set-up-gitlab-server-for-ci-cd-operation-on-centos

Hummerston, J., August 12, 2020, The Importance of Business Communication Software in 2020, Retrieved April 20, 2021. Available at: https://hackernoon.com/the-importance-of-business-communication-software-in-2020-ss7r3u33

IBM, August 2020, Cost of a Data Breach Report 2020, 2020, IBM

Ikeanyi, C., April 16, 2018, Working with Git, Retrieved April 18, 2021. Available at: https://medium.com/lean-in-women-in-tech-india/working-with-git-9dafa8f986b3

Miami University, Programming Style Guidelines, Retrieved April 19, 2021. Available at: https://miamioh.edu/cec/academics/departments/cse/academics/programming-style/index.html

OP Ryhmä, January 2021, OP:n Suuryritystutkimus 2021, 2021, OP Ryhmä

Palkkadata, Palkat kategoriassa: IT, Retrieved April 20, 2021. Available at: https://www.palkkadata.fi/salaryinfo/it

Slack, Pricing, Retrieved April 20, 2021. Available at: https://slack.com/intl/en-fi/pricing

Somasundaram, R., 2013. Git: version control for everyone beginner's guide: the non-coder's guide to everyday version control for increased efficiency and productivity, 2013, Birmingham: Packt Pub.

Write the Docs, A beginner’s guide to writing documentation, Retrieved April 20, 2021. Available at: https://www.writethedocs.org/guide/writing/beginners-guide-to-docs

Write the Docs, Style Guides, Retrieved April 20, 2021. Available at: https://www.writethedocs.org/guide/writing/style-guides