[Aashis Rimal] ([1304965])







Assignments

Mastering   
Version Control with Git

1st edition

Contents

[Introduction part 2](#_Toc463301940)

[Exercise 1 3](#_Toc463301941)

[Exercise 2 4](#_Toc463301942)

[Exercise 3 5](#_Toc463301943)

[Exercise 4 6](#_Toc463301944)

[Exercise 5 7](#_Toc463301945)

[Exercise 6 8](#_Toc463301946)

[Video 1 9](#_Toc463301947)

[Intermediate part 10](#_Toc463301948)

[Exercise 7 11](#_Toc463301949)

[Exercise 8 13](#_Toc463301950)

[Exercise 9 14](#_Toc463301951)

[Advanced part 15](#_Toc463301952)

[Exercise 10 16](#_Toc463301953)

[Exercise 11 18](#_Toc463301954)

[Exercise 12 19](#_Toc463301955)

[Exercise 13 20](#_Toc463301956)

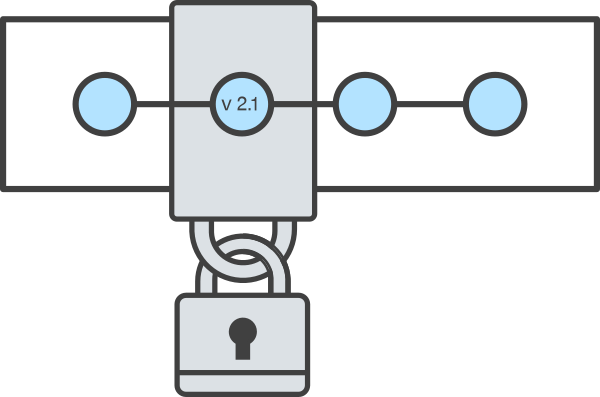
[Exercise 14 21](#_Toc463301957)

[Feedback 22](#_Toc463301958)

# Introduction part

## Exercise 1

**What is Version Control?** (Slide 11-20)

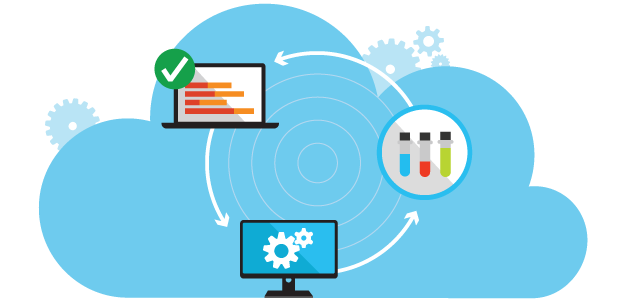


In simple words explain why you should use Version Control and what are the Benefits of using that for you and your team?

Version control is a software tool that keeps track of changes made to a document or source code in a special database every single time the source is changed. Version control makes a group or a team working together on a project easier to collaborate with each other. Each changes on the source is tracked and can be accessible whenever we want to view them. So, if a developer makes a mistake, he can go back to the previous correct version without disturbing his team members. Version control helps to see the complete changes made by the developer over the course of time which includes addition of text or files, their deletion and every edits made on those documents. A project can be branched to many parts and worked on independently and can again be merged back together so that the changes made do not conflict with each other. Another benefit of using version control is that the changes made can be saved with a message describing why the change was made. This makes it easy to understand the changes while looking back to the history.

## Exercise 2

**Continuous Integration** (Slide 21-25)



Briefly explain Continuous Integration development practice.

Developers write codes for their projects on a continuous basis. After writing some codes, they need to integrate those codes on a shared repository which is then verified automatically. This practice helps to detect errors quickly and find those errors early in the process.

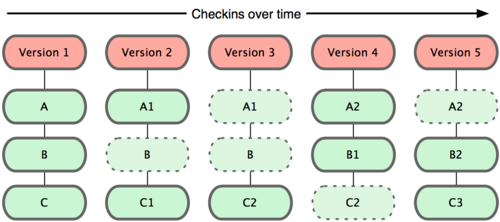
List some of the Continuous Integration software and tools and compare them briefly.

Some of the Continuous Integration Software are:

1. **Jenkins:** Jenkins is a continuous integration software developed in Java. Jenkins main focus is to build or test software products continuously and monitor externally run jobs. It can be integrated to Bugzilla, Jira, Bitbucket, Checkstyle, etc.
2. **Buildbot:** Buildbot is developed in python and is based on the Twisted framework. Buildbot installation has some masters and a collection of slaves. The masters monitor source code repositories for changes, coordinates the activities of the slaves and report the results to users and developers while the slaves run on a variety of operation systems.
3. **Travis CI:** Travis CI is an open source software which is free to host on own server. It also offers a SaaS version which allows free testing for open source projects. It can be integrated to GitHub and Heroku.
4. **Strider:** Strider is developed in Node.js and javascript and uses MongoDB as a backing store. Strider can be customized through plugins. It can be integrated to GitHub, BitBucket, Heroku, Git, etc.

## Exercise 3

**How does Git work?** (Slide 31-32)



Explain what the difference between how does Git work comparing to other Version Control Systems?

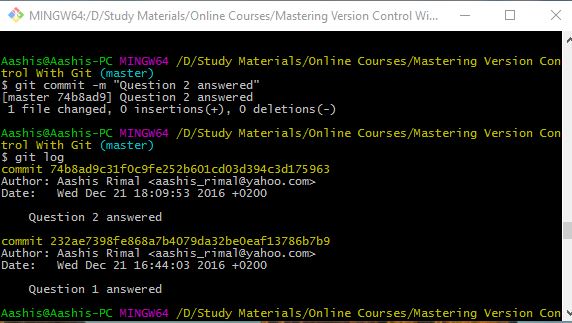
Git is a distributed version control system (DVCS) which emphasizes on speed, data integrity and support for distributed non-linear workflows. With DVCS, there is not one centralized code base to pull the code from and different branches hold different parts of the code. Git takes a picture of the files and stores a reference to that picture every time a user saves or commits the state of the project. Other version control systems, such as Subversion (SVN) and Concurrent Version System (CVS) use centralized version control and use only one master copy of the software.

## Exercise 4

**Installing Git** (Slide 45-47)

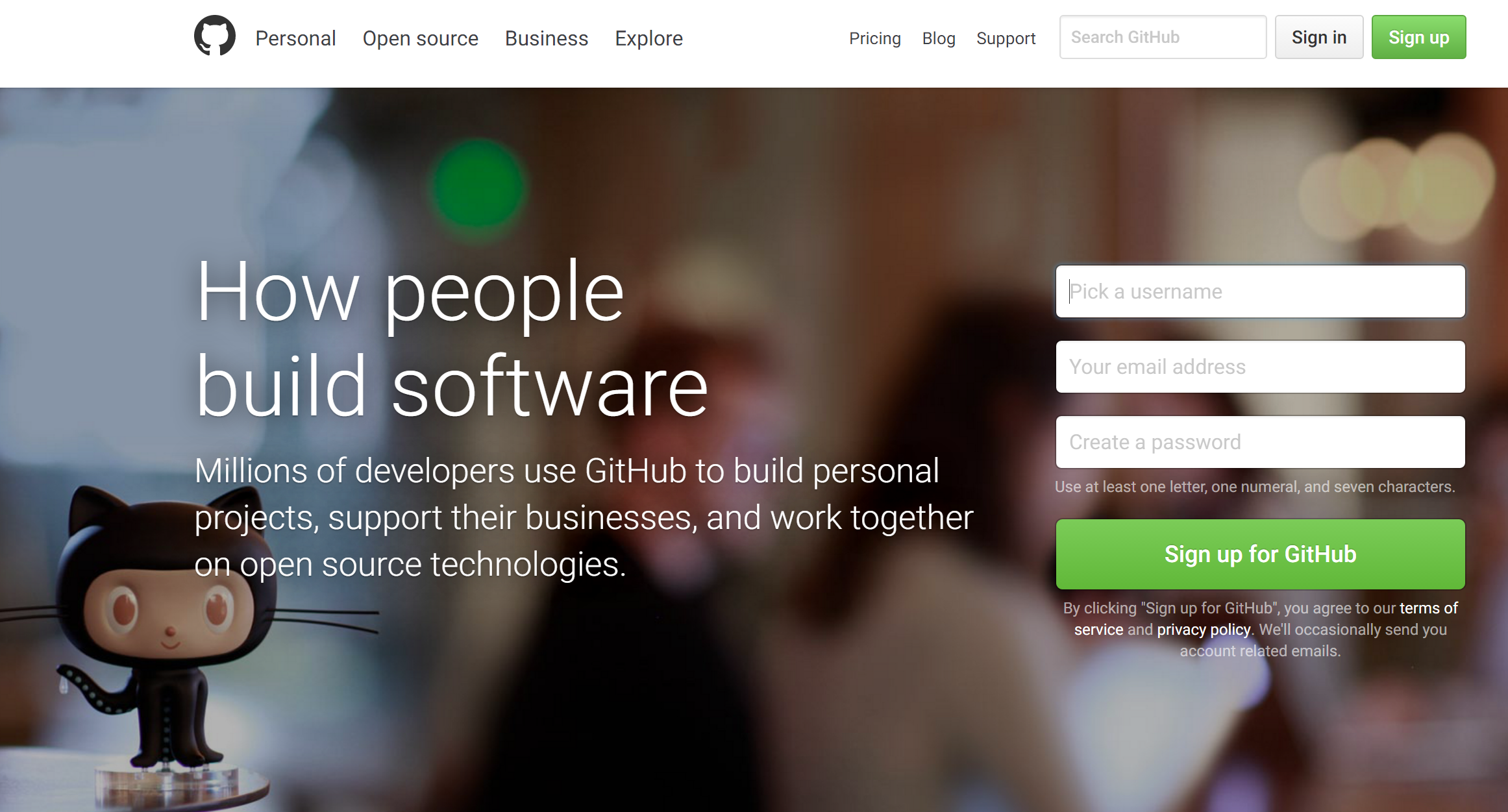


Install Git on your computer and provide screenshots from the process.

I have already installed git in my computer. Here is a snapshot of git bash.

## Exercise 5

**GitHub** (Slide 50-51)



Create your account in GitHub and write your username.

aashisr

## Exercise 6

**Related Articles**



Search and find 3 beneficial online articles, videos, or etc. related to the material of this part of the course, and briefly describe the material you learned from them.

Link 1: https://www.smashingmagazine.com/2008/09/the-top-7-open-source-version-control-systems/

Subject: Version Control Systems

What you learned:

I learned how git works in comparison to other version control systems. Git is a distributed version control system in which different branches hold different parts of the code unlike on other version control systems where only one master copy is used.

Link 2: https://opensource.com/business/15/7/six-continuous-integration-tools

Subject: Continuation integration tools

What you learned:

Here is a comparison of six open source continuous integration tools. Jenkins, buildbot, travis CI, strider, Go, Integrity, etc are the widely used version control systems. They were built on different programming languages such as Integrity on Ruby, Buildbot on Python, Jenkins on Java and Strider on Node.JS and javascript with MongoDb as a backing store.

Link 3: https://www.youtube.com/watch?v=Jif2U2oPVI4

Subject: What is version control ?

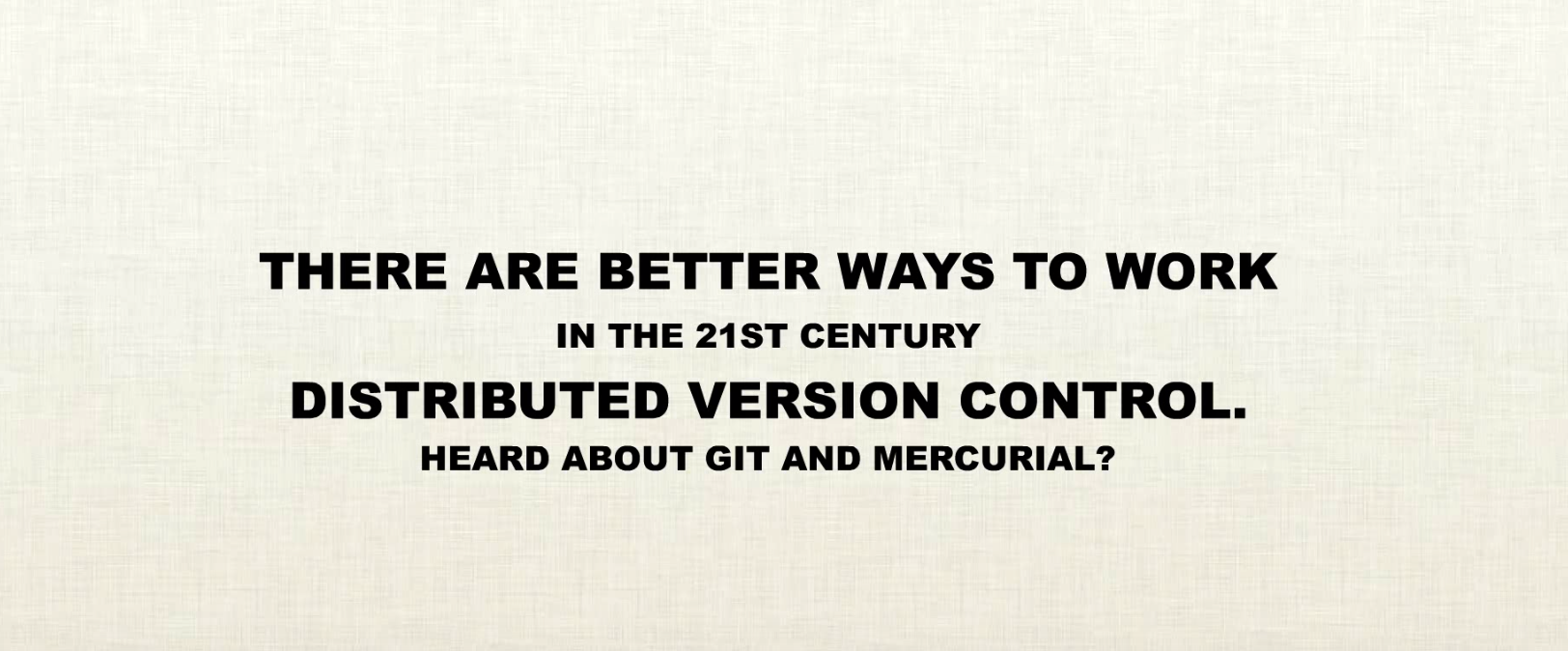
What you learned:

* Git is a fast and modern implementation of version control.
* Git provides a history of content changes.
* Git facilitates collaborative changes to files.

## Video 1

**Title: Centralized vs Distributed Version Control in 90 seconds**

**Duration: 91 seconds**



Link: <https://www.youtube.com/watch?v=_yQlKEq-Ueg>

**How was it? Anything new you learned?**

This is an interesting and informative video with meaningful animations and a pleasant musical background. This video shows the differences between 20th century’s centralized version control system and the new, modern and efficient distributed version control system. In centralized version control system, a failure or error in the code by one of the developers could hamper all other developers. Remote commits are slow and it is hard to merge files in the repository. However, using distributed version controls like git and mercurial can make publishing local commit fast even in the absence of network. Developers work on their part and publish commits regularly to the central repository which can be reviewed and managed by project supervisor. Merging is easy and fast.

# Intermediate part

## Exercise 7

**Git commands** (Several slides)



Fill the blanks with the proper git commands:

git commit – git revert – git config – git add – git init – git log – git reset – git clone – git stash – git status – git checkout – git clean

The \_\_\_\_git clean\_\_\_\_\_\_\_\_ command removes untracked files from your working directory. This is really more of a convenience command, since it’s trivial to see which files are untracked with git status and remove them manually.

The \_\_\_\_\_git checkout\_\_\_\_\_\_\_ command serves three distinct functions: checking out files, checking out commits, and checking out branches.

\_\_\_\_\_git stash\_\_\_\_\_\_\_ temporarily shelves changes you‘ve made to your working copy so you can work on something else, and then come back and re-apply them later on.

The \_\_\_\_git clone\_\_\_\_\_\_\_\_ command copies an existing Git repository. This is sort of like svn checkout, except the “working copy” is a full-fledged Git repository—it has its own history, manages its own files, and is a completely isolated environment from the original repository.

The \_\_\_git commit\_\_\_\_\_\_\_\_\_ command commits the staged snapshot to the project history.

The \_\_\_\_\_git status\_\_\_\_\_\_\_ command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven’t, and which files aren’t being tracked by Git.

The \_\_\_git config\_\_\_\_\_\_\_\_\_ command lets you configure your Git installation (or an individual repository) from the command line. This command can define everything from user info to preferences to the behavior of a repository.

The \_\_\_\_git add\_\_\_\_\_\_\_\_ command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit.

The \_\_\_\_git log \_\_\_\_\_\_\_\_ command displays committed snapshots. It lets you list the project history, filter it, and search for specific changes.

The \_\_\_git init\_\_\_\_\_\_\_\_\_ command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new empty repository.

The \_\_\_\_git revert\_\_\_\_\_\_\_\_ command undoes a committed snapshot. But, instead of removing the commit from the project history, it figures out how to undo the changes introduced by the commit and appends a new commit with the resulting content.

When you undo with \_\_\_\_\_git reset\_\_\_\_\_\_\_ (and the commits are no longer referenced by any ref or the reflog), there is no way to retrieve the original copy—it is a permanent undo.

## Exercise 8

**git init vs. git init –bare** (Slide 16-19)



Explain briefly the difference between “git init” and “git init –bare” commands.

[Your answer]

Git init is a git command which creates a new empty repository or converts an existing file to a Git repository from the current directory. It adds .git folder inside it which stores the revision history. Git init creates a working directory where the files can be edited, modified and added. However, git init --bare command creates a new repository without a working copy. Bare repository can be used as a central hub or server to share the changes with other people. For example, github.com repository is created as a bare repository.

## Exercise 9

**Related Articles**



Search and find 3 beneficial online articles, videos, or etc. related to the material of this part of the course, and briefly describe the material you learned from them.

Link 1: https://www.siteground.com/tutorials/git/commands.htm

Subject: GIT Tutorial Commands

What you learned:

I learned the major commands used in git along with their description and examples. Some of the major commands are:

-git config: Sets configuration values for username, email

-git init: initializes the git repository

-git add: adds changed files from the working directory to the repository

-git commit: git commit commits to the file changed with a message and sets it to point to that commit

Link 2: http://stackoverflow.com/questions/7861184/what-is-the-difference-between-git-init-and-git-init-bare

Subject: Difference between git init and git init --bare

What you learned:

Git init is used to create a new repository in a working directory locally in a computer while git init –bare command creates a repository without a working directory.

Link 3: https://www.youtube.com/watch?v=cEGIFZDyszA&list=PL6gx4Cwl9DGAKWClAD\_iKpNC0bGHxGhcx

Subject: Git tutorials

What you learned:

- Configure username and email

- Creating repository

- Git basic commands

- Git workflow

- Comparing staging area with repository

- git ignore and github Desktop

# Advanced part

## Exercise 10

**Git commands** (Several slides)



Fill the blanks with the proper git commands:

Git branch

git commit --amend

git pull

git rebase

git remote

git push

git fetch

git checkout

git merge

The \_\_\_\_\_git checkout\_\_\_\_\_\_\_ command lets you navigate between the branches created by git branch.

The most common use case for \_\_\_\_git push\_\_\_\_\_\_\_\_ is to publish your local changes to a central repository. After you’ve accumulated several local commits and are ready to share them with the rest of the team, you (optionally) clean them up with an interactive rebase, then push them to the central repository.

The \_\_\_\_git fetch\_\_\_\_\_\_\_\_ command imports commits from a remote repository into your local repo. The resulting commits are stored as remote branches instead of the normal local branches that we’ve been working with.

The \_\_\_\_git pull\_\_\_\_\_\_\_\_ command, fetch the specified remote’s copy of the current branch and immediately merge it into the local copy.

The \_\_\_\_\_\_git remote\_\_\_\_\_\_ command lets you create, view, and delete connections to other repositories.

The \_\_\_\_git rebase\_\_\_\_\_\_\_\_ rebase the current branch onto <base>, which can be any kind of commit reference.

The \_\_\_\_git merge\_\_\_\_\_\_\_\_ command lets you take the independent lines of development created by git branch and integrate them into a single branch.

The \_\_\_\_\_git branch\_\_\_\_\_\_\_ command lets you create, list, rename, and delete branches. It doesn’t let you switch between branches or put a forked history back together again.

The \_\_\_\_\_git commit --amend\_\_\_\_\_\_\_ command is a convenient way to fix up the most recent commit. It lets you combine staged changes with the previous commit instead of committing it as an entirely new snapshot. It can also be used to simply edit the previous commit message without changing its snapshot.

## Exercise 11

**git commit --amend** (Slide 11)



Explain briefly why you should not amend public comments.

As commit message is also the part of the commit, changing public comments changes the hash of the commit and also the git history. Git history can only be changed when the files are private. If we push our changes for other people to see and pull, it can create problems to their repositories. So, public comments should not be amended.

## Exercise 12

**git merge** (Slide 58-62)

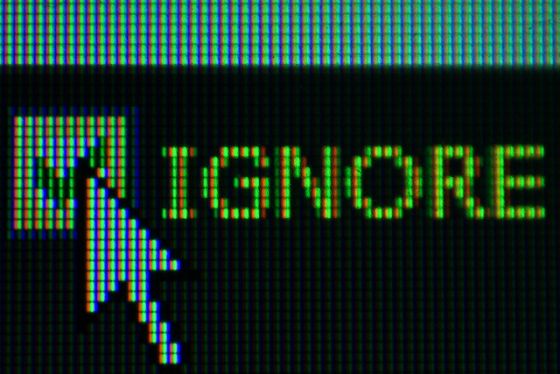


Explain briefly the difference between Fast-forward Merge and 3-way Merge.

[Your answer]

## Exercise 13

**Ignoring Files** (Slide 65)



Set the content of .gitignore file that git ignores files with the extentions of EXE and BAT.

[Your answer]

## Exercise 14

**Related Articles**



Search and find 3 beneficial online articles, videos, or etc. related to the material of this part of the course, and briefly describe the material you learned from them.

Link 1: https://www.youtube.com/watch?v=KDt01U859Ik

Subject: Adding a remote Repository

What you learned:

[Your answer]

Link 2: https://www.youtube.com/watch?v=FdZecVxzJbk

Subject: Fixing common mistakes and undoing bad comments

What you learned:

- How to fix common mistakes in the repository

- How to undo bad commits

- why not to amend the public comments

- Adding gitignore files

- Copying the commits to other branches with “git cherry-pick”

Link 3:

Subject:

What you learned:

[Your answer]

# Feedback

**Quiz**

1. In your opinion, how hard was the quiz?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Hard | Hard | Moderate | Easy | Very Easy |
|  |  |  |  |  |

2. Your feedback/suggestions/comments about the quiz.

[Answer here]

**Project**

3. In your opinion, how hard was the project?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Hard | Hard | Moderate | Easy | Very Easy |
|  |  |  |  |  |

4. Your feedback/suggestions/comments about the project.

[Answer here]

**Assignments**

5. In your opinion, how hard was the assignments?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Hard | Hard | Moderate | Easy | Very Easy |
|  |  |  |  |  |

6. Your feedback/suggestions/comments about the assignments.

[Answer here]

**Teacher**

7. How much satisfied you are about the teacher’s professional skills?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Satisfied | Satisfied | So-so | Unsatisfied | Very unsatisfied |
|  |  |  |  |  |

8. How much satisfied you are about the teacher’s skills to deliver the material in the lectures?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Satisfied | Satisfied | So-so | Unsatisfied | Very unsatisfied |
|  |  |  |  |  |

9. Your feedback/suggestions/comments for the teacher to improve himself.

[Answer here]

**Course**

10. How much satisfied you are about the course material?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Satisfied | Satisfied | So-so | Unsatisfied | Very unsatisfied |
|  |  |  |  |  |

11. How much satisfied you are about the course teaching method and structure?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Very Satisfied | Satisfied | So-so | Unsatisfied | Very unsatisfied |
|  |  |  |  |  |

12. Your feedback/suggestions/comments about the course.

[Answer here]

13. Your satisfaction percentages from everything related to this course (0-100).

[Answer here]

14. Were there any other material related to the course that you were looking forward to learn but they were not in the course material?

[Answer here]

15. Did you encounter any mistakes in the course material?

There is no question no. 22 in Quiz.

16. Do you have any other feedback/suggestions/comments?

[Answer here]

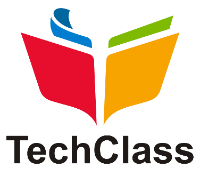
**You need to return this assignment before its deadline for your final assessment.**

**Good luck!** ☺ **Farhad Eftekhari**

Copyright © 2016 by Farhad Eftekhari

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law. For permission requests, write to the publisher, addressed “Attention: Permissions Coordinator,” at the address below.

Helsinki Metropolia UAS  
Vanha maantie 6  
02650 Espoo, Finland  
www.techclass.co

  ****