1. **Video 51: Introduction**

* Commits
* Merging and combining different branches
* Resolving Conflicts

1. **Video 52: Understanding the git stash**

* **git stash:** saves un-staged commits
* **git stash apply**: gives you access to your latest/last un-staged commit
* **git stash apply list**: gives you access to all your un-staged commits in a list format, the top
* **git stash apply 0/1/2:** takes you to a specific stash (un staged commit)
* **git stash push -m “commit message”**: adds a message to your stash/un-staged commit
* **git stash pop [stash number]:** adds a stashed commit to your repository, from here you can proceed to commit the change (git add . , git commit -m “commit message”)
* **git stash drop [stash number]:** deletes a specific stash
* **git stash clear:** deletes the entire stash list

1. **video 53: Bringing Lost Data Back with git reflog**

* **git reset - -hard HEAD~1:** deletes your last commit
* **git reflog**: shows all the changes applied within the last 30 days
* **git reset - -hard [log number]:** brings back the specific deleted commit
* **git branch -D [branch name]:** deletes a branch
* Bringing back the branch and the commit in it:
  + **git reflog:** accesses the commit added in this branch
  + **git checkout [commit number]:** checks-out the created commit
  + **git checkout -**b [branch name] OR **git switch -c [branch name]**: recreate the branch and switch to it

1. **Video 54: Combining branches (Master and feature branch)**

* Introduction

1. **Video 55: Understanding Merge**

* Merge Types
  + Fast Forward
  + Non Fast Forward Merger
    - **Recursive**
    - Ours
    - Octopus
    - Subtree

1. **Video 56: Applying the Fast Forward Merge**

* Add files in the master branch
* Create feature branch and add files in it
* Merge Master branch and Feature branch, to do so:
  + Switch to the master branch and use command:
    - **git switch Master**
    - **git merge Feature**
* Fast forward merge has occurred
* **git merge - -squash feature**: taking all commits in a branch, combining them and staging them together
* **git commit -m “merged feature and master”:** merges feature branch into the master branch

1. **Video 57: The Recursive Merge (Non-Fast-Forward)**

* git merge - -no-ff [branch name]: merges a branch into the main branch without using the fast forward method

1. **Video 58: Rebasing Theory**

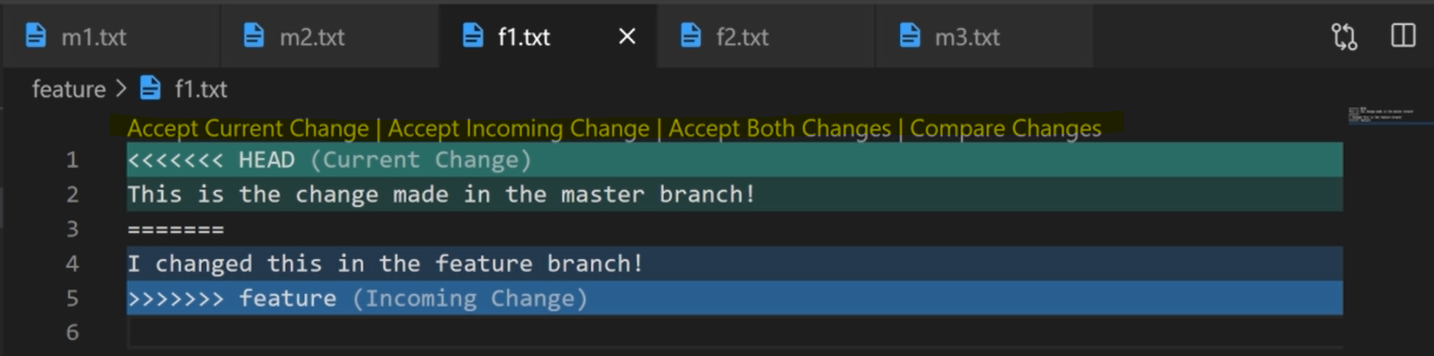
* Taking the last commit in the master branch, putting it back into the feature branch and therefore making it a base for new commits in the feature branch. Then merge the rebased master back to the master branch.

1. **Video 59: Applying git rebase**

* **git rebase master:** takes last commit in master branch and rebases it in the feature branch
* rebasing creates new commits (new id’s), however the content will be the same. NB! Rebasing changes history of project in this way.
* In the end, rebasing creates a fast forward merge.

1. **Video 60: Handling merge conflicts**

* Conflicts occur for example if two people work in the same file under different branches.
* **git abort**: aborts the merge
* **git log - -merge**: shows you the commits you want to merge
* **git diff**: shows you the conflict problems that you are encountering
* choose from given options to solve the conflict: Accept current change, Accept incoming change, Accept both changes, Compare changes. Or go back to VSC and make changes manually:



1. **Video 61: Merge vs Rebase vs Cherry Pick**

* Cherry picking is adding a specific commit to a specific branch, therefore creating copies of commits (duplicates)

1. **Video 62: Understanding git Cherry Pick:**

* **git cherry-pick [commit id]**: selects a specific commit and commits it in a specific branch
* Note: Commit id needs to be copies prior to using this command

1. **Video 63: Working with tags (git tags)**

* Tags highlight important stages in a project and help in finding commits quickly.
* **Light weight tag (Temporary tag):** A pointer towards a commit in a branch (Head).
* **Annotated tag**: Full object in git, contains information like email of the person who added the tag.
* **git tag**: shows you a list of all tags in your project.
* **Light weight tag:**
  + To add a tag to a commit, first use **git log** to see the commit ID and copy it.
  + Once copied us:
    - **git tag [tag number] [commit id]**
  + **git show [tag number]**Shows you the detail of the commit (file name and commit id)
  + **git checkout [tag number]:** detaches the commit and shows you info related to this commit
  + **git tag -d [tag number]:** removes the tag on a commit
* **Annotated tag:**
  + **git tag -a [tag number] -m “tag message”:** creates a tag with detailed information on the tag and the person that created the tag

1. **Video 64: Wrap Up (Summary)**

