Version Control with Git

Git is a version control system and differs in many ways from classical data management possibilities (Windows Explorer) and Services (Dropbox, Google Drive, etc.) which do not provide a decentralized approach. Services like Dropbox and Google Drive sequentially update files in a central repository and get into trouble when a file was edited by multiple users at the same time without sync in-between edits. Also it requires a permanent internet connection to reduce possible sync problems.

Git on the other Hand is a decentralized system. While there is still a centralized repository (For instance on GitHub), every user has a local copy of the repository on his own machine. Git then allows to push and pull only certain files or folder to and from the repository. Even more important, it allows branching of projects, so you can actually work on multiple versions of the same project at the same time and later merge them together if needed.

What do we need?

- Git (https://git-scm.com/download/)
- Command Line / Terminal / Git Bash
- (Code) Editor
- Optionally a Visual Git Tool like SourceTree or GitKraken (https://git-scm.com/download/gui/windows)
- File Server or a Git Service like GitHub (https://github.com/) if you intend to work in a team (A repository reachable from the local network or internet)

First Steps

1. Setup Git
    a. git config --global user.name “First Last"
    b. git config --global user.email "email@example.com"
2. Create a project folder on your Desktop
3. Create an empty text document and name it readme.md
4. Add some content to it in an editor (“This is an example readyme file”)
5. Create an additional text document, this time call it example.txt and leave it empty
6. **Switch to Terminal / Console / Bash**
7. Falls nicht als Admin angemeldet (Mac & Linux) sudo
8. Travel to the project folder (cd (/d) “Path”)
9. Command: Git init
10. Show Hidden Files and Folders – Command: ls –la or dir /A:DH
11. Git add readme.md
12. Git status shows
    a. Staged and unstaged files
    b. Files ready to be commited
    c. Untracked files (Files not yet picked up by git at all)
13. Git add example.txt
14. Git status
15. Git rm –cached “exmpale.txt”
16. Git status
17. Git commit –m “Initial Commit – Added readme file”

18. Git status
19. Git add example.txt
20. Git status
21. Git commit – m "Added example file"
22. Git status (we are clean)

Explain Three-fold model (Show your painting skills)

- Working Area (Files you are actively working on)
- Staging Area (Files of one logical working unit meant to be submitted together)
- Repository (.git directory / sort of a snapshot database of your file system / Local)

Git works on a snapshot basis, so each commit is a snapshot. This enables you to rollback a project or files to a specific commit as well as see differences between commits.

https://sethrobertson.github.io/GitBestPractices/

23. Make a change to the example file ("This is an example file")
24. Git status (shows modified file)
25. Git checkout - - example.txt (Reverts the file to the last snapshot/commit)
26. Edit example.txt again
27. Git add
28. Git commit –m "Added text to the example file"
29. Git status
30. Git log
31. Git checkout PreviousCommitHashFromLog (We get an detached head)
32. Git log (only Commits previous to the checked out Commit are listed)
33. Git checkout LastCommitHashFromPreviousLog (We get back to the original state.. phu)
34. Git log
35. Again -> git checkout PreviousCommitHashFromLog
36. Git branch "TestBranch" (creates a new Branch from current checkout)

Shorthand would be git checkout PreviousCommitHashFromLog –b "TestBranch"

37. Edit example.txt file "This is an example in a second branch. And this sentence doesn't even exist in the Master Branch!"
38. Git status
39. Git add example.txt
40. Git commit –m "Added content to example file"
41. Git status
42. Git log
43. Git checkout master
44. Look at example.txt file
45. Git checkout TestBranch
46. Look at example.txt file again
47. We had a spelling error in our readme.md file.. This is very important needs to be fixed instantly, our work in the TestBranch can wait.. But other Team Members might be working on different things, so we're doing a hotfix and use an extra branch to do this (normally done on multiple files and longer timespan obviously)
48. Git checkout master
49. Git branch "mdHotfix"
50. Git checkout mdHotfix

51. Git log
52. Edit readme.md file
53. Git status
54. Git add readme.md
55. Git commit –m “Fixed spelling in readme file”
56. Git checkout master
57. Git merge mdHotfix (Because there were no commits in-between it was able to fast-forward – not a real merge, but rather like a new commit and the master moved to the same commit as mdHotfix)
58. Git branch –d mdHotfix (and one less branch)
59. Git checkout TestBranch
60. Edit readme.md, add an Additonal line “readme is an important file”
61. Git add readme.md
62. Git commit –m “Added more content to readme”
63. We are done with our work. Git checkout master
64. Git merge TestBranch
65. We got a conflict .. yeahaaa (our example.txt seriously doesn’t like auto merge..)
66. Manuall edit it to resolve the merge conflict
67. git add example.txt
68. git commit (–m “merged TestBranch to master”)
69. git branch –d TestBranch
70. git branch
71. We are all clean
72. Additional branch commands:
    a. Git branch –m “original” “newName”

Together with GitHub

Link (Team Repository - Assignment link)

Link (Class Repository (To pull only))

Product Owner / Team Lead als erstes. Neues Team erstellen mit dem Namen des Spiels.

Dann alle anderen (Das richtige Team wählen, kann nicht mehr geändert werden)!

73. Git remote add origin (Link – Class Repository)
74. More remote Commands:
    a. git remote (Overview)
    b. git remote add <alias> <url>
    c. git remote rm <alias>
75. Maybe you need to login
76. Yay.. if changes were made in github
    a. Git fetch <alias> (to update your repository to the latest remote)
    b. Git pull <alias> (to update your repository and working data to latest remote)
77. To mention ssh and clone