Version management

Git

Git documentation

Git is a version control system that is used for software development and other version control tasks. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows. Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.

As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server. Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2.

Kasutatud mõisted

- Working directory - where you'll be doing all the work: creating, editing, deleting and organizing files.
- Staging area - where you'll list changes you make to the working directory.
- Repository - where Git permanently stores those changes as different *versions* of the project.
  - Commit - saving changes to the repository.
- HEAD - You can think of the HEAD as the “current branch”. When you switch branches with git checkout, the HEAD revision changes to point to the tip of the new branch.
- CHECKOUT - Updates files in the working tree to match the version in the index or the specified tree. If no paths are given, git checkout will also update HEAD to set the specified branch as the current branch.
- BRANCH - a branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master. As you initially make commits, you're given a master branch that points to the last commit you made. Every time you commit, it moves forward automatically.

Workflow

Basic workflow

- Käivita Git Bash
- Liigu soovitud teeki, milles alustada GIT projekti, ning käivita käsurealt '**git init**' (*The command sets up all the tools Git needs to begin tracking changes made to the project*). Sellega on vastav kataloog valmis Git versioonihaldust kasutama.
- Failide lisamiseks Git versioonihaldusesse, käivita '**git add [failinimi**' või '**git add** *'.
  - Käsuga 'git status' on näha teegis olevad failid. Juhul kui failid on lisatud Git-i, on need rohelised, juhul kui pole, on need punased.
- Käsuga '**git diff [failinimi]**' on näha failis tehtud muudatused.
- Kui tehtud muudatused on sobivad salvestamiseks, saab need käsuga '**git commit**' reposse saata.
  - süntaks ise on taoline '**git commit -m “[muudatuse nimi]“**'

- Käsuga '**git log**' on võimalik näha *current branch*-i commit-e.

Lihtsa töövoo peamised käsud

- **git init** creates a new Git repository
- **git status** inspects the contents of the working directory and staging area
- **git add** adds files from the working directory to the staging area
- **git diff** shows the difference between the working directory and the staging area
- **git commit** permanently stores file changes from the staging area in the repository
- **git log** shows a list of all previous commits

Backtrack

- Käsuga '**git show head**' kuvatakse viimase commitiga tehtud muudatused pluss 'git log' viimase muudatuse teave.
- juhul kui tekib vajadus taastada faili seis sellisena, nagu see oli peale viimast commiti ehk siis tühistada vahepealsed muudatused, siis seda saab teha käsuga '**git checkout HEAD [failinimi]**'
- juhul kui *staging area*-sse sai kogemata lisatud ebasobiv fail, näiteks mõne vajaliku reata, siis *staging area*-st tagasi saab võtta faili käsuga '**git reset HEAD [failinimi]**'.
- juhul kui vahepeal on õnnestunud repo-sse commit-ida ühe või mitu versiooni, mida pole vaja, siis tühistada saab seda käsuga '**git reset [SHA]**', kus SHA = '**git log**' saadud SHA väärtus.
- repo-st soovitud varasema versiooni kättesaamiseks: '**git checkout [failinimi]**'

Backtracki peamised käsud

- **git checkout HEAD filename** discards changes in the working directory.
- **git reset HEAD filename** unstages file changes in the staging area.
- **git reset SHA** can be used to reset to a previous commit in your commit history.

Git branching

- Uue branch-i loomiseks on käsk '**git branch [new_branch]**', erinevate branch-ide vahel liikumiseks on aga käsk '**git checkout [branch_name]**'.
- git käsurea olekuribal (käsurea kohal olev) on näha, mis branchil kasutaja hetkel on, viimane sulgudes olev sõne nt ”(master)”.
- branch-i muudatuste lisamiseks master branchi commit-i tehtud muudatused, liigu master branchi ning kasuta käsku nt “**git merge [branch_name]**'.
- erinevate branchide nägemiseks on võimalik kasutada käsklust '**git show-branch**'.
- sama/samu faile on võimalik erinevate branchide alt muuta ja commit-ida, erinevas branchis vastava failide versiooni laadimiseks kasutada käsku 'git checkout HEAD *'.
- juhul kui konkreetne branch on oma eesmärgi täitnud ning selle võib lisada masterisse ja branch ise kustutada '**git branch -d [branch_name]**'

Branching-u peamised käsud

- **git branch** lists all a Git project's branches.
- **git branch branch_name** creates a new branch.
- **git checkout branch_name** used to switch from one branch to another.
- **git merge branch_name** used to join file changes from one branch to another.
- **git branch -d branch_name** deletes the branch specified.

Git teamwork

In order to collaborate, you and team needs:

- A complete replica of the project on your own computers
- A way to keep track of and review each other's work
- Access to a definitive project version

- tekitamaks repo-st enale klooni, kasutada käsku '**git clone [repo asukoht] [klooni nimi]**'. klooni nimi on ühtlasi *remote*. kloonitud projektide andmeid on võimalik näha, vastava klooni asukohas, käsuga '**git remote -v**'.
- repo-st viimaste muudatuste hankimiseks kasutada klooni asukohas käsku “**git fetch**”. sellega tekitatakse eraldi branchi muudatus, klooni masterisse muudatuste saamiseks kasutada käsku “git merge origin/master”.
- klooni juures muudatuste tegemiseks on paslik teha uue branch, millesse sooritada soovitud muudatusega. peale muudatuste sooritamist ja commit-i suunata need repo-sse tagasi käsuga “**git push origin [klooni branchi nimi]**”.
- Repo-sse tekib vastav branch, mida on võimalik repo masteriga liita. peale käsku “**git merge [branch-i nimi]**”.

- A remote is a Git repository that lives outside your Git project folder. Remotes can live on the web, on a shared network or even in a separate folder on your local computer.
- The Git Collaborative Workflow are steps that enable smooth project development when multiple collaborators are working on the same Git project.

Teamworki peamised käsud

- **git clone** creates a local copy of a remote.
- **git remote -v** lists a Git project's remotes.
- **git fetch** fetches work from the remote into the local copy.
- **git merge origin/master** merges origin/master into your local branch.
- **git push origin <branch_name>** pushes a local branch to the origin remote.

GitHub

GitHub is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a Web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. GitHub offers both plans for private repositories and free accounts, which are usually used to host open-source software projects.

Deploying a site to GitHub

To specify the repo using Git, we'll have to add the remote and label it as the origin.

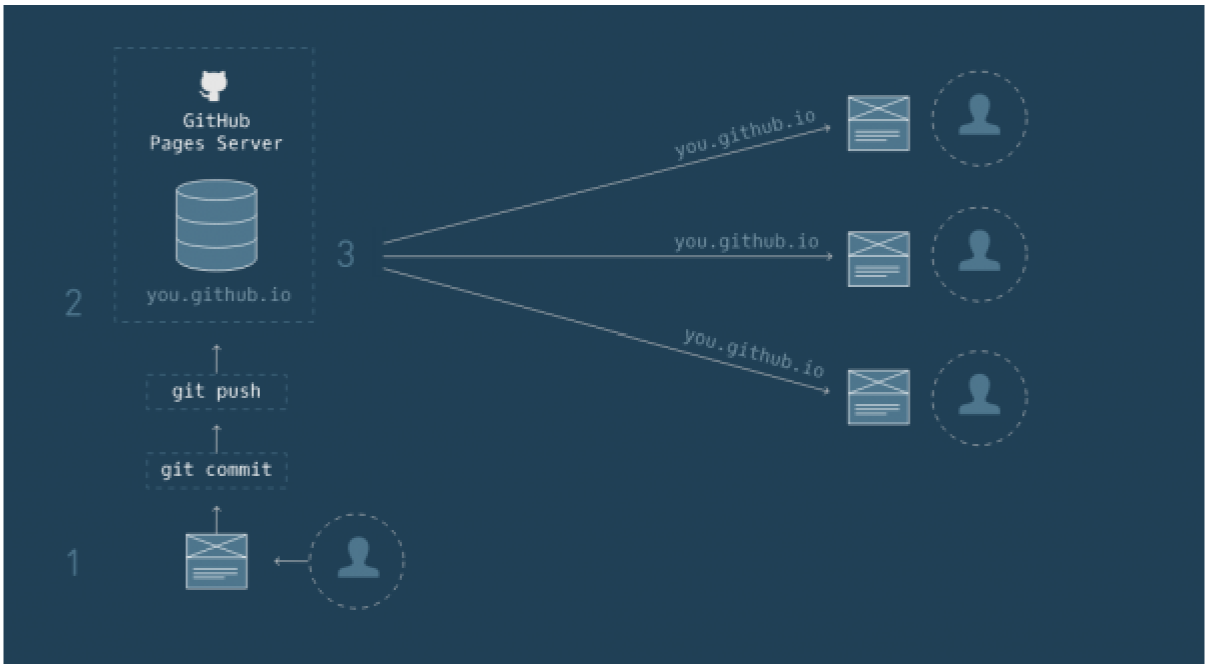
- The remote is the URL of the repo that will store your site's contents.
- The origin is an alias for the remote. You can think of an alias as an abbreviation or a substitute name. This means that instead of having to always type the lengthy remote URL over and over again, you can simply refer to it as origin later on.

- In the terminal, you can add the remote with the following command: '**git remote add origin**

https://github.com/your-user-name/your-user-name.github.io.git', antud hetkel '**git remote add origin https://github.com/kalle404/cautious-broccoli.git**'
    - Important: If you accidentally make a mistake when adding the remote URL, you can start over and remove the remote with the following command: '**git remote rm origin**'
- Confirm that the remote was succesfully added, by typing the following: '**git remote -v**'
- Add all of your site's contents using the following Git command: '**git add .**'
- Save your changes using Git's commit command and the following commit message: '**git commit -m "Save my work"**'
- In the terminal, push the contents of your site to your repo, use Git's push command and push the contents of your site up to your repo using the following command: '**git push -u origin master**'

Getting stuff from GitHub

- Commands can be read here.



Going live with a site

- Domain names are human-friendly names that identify servers on the Internet. A global system known as the Domain Name System (DNS) is used for storing which domain names correspond to which servers.
- For example, Codecademy's domain name is www.codecademy.com. When you type the domain name into your browser, your computer asks the DNS to identify which servers should receive the request in order to load our website.
- Often, the most time consuming part of buying a domain name is actually deciding what you'd like it to be. Be aware that not all domain names are available; many have already been claimed by others. We're going to use Amazon Web Services (AWS) to purchase your custom domain. AWS is an industry standard suite of web infrastructure services used frequently by developers. The specific service we're going to use to purchase your domain name is called Route 53.

Mida uurida vaja

- ühest arendusarvutist reposse commitimine
- ühest arvutist reposse lisaks commitimine
- ühes arvutis repo oleku taastamine
- teisest arvutist repost kloonimine
- teisest arvutist reposse lisaks commitimine
- teises arvutis repo oleku taastamine
- erinevate branchide tekitamine kahes arvutis
    - näiteks branchi tekitamine remotest käib käsuga: '**git push <REMOTENAME> <LOCALBRANCHNAME>:<REMOTEBRANCHNAME>**'
- branchide lisamine masterisse peale arenduse lõppu: liikuda master branch-i ja merge
- kui on tõsiselt vaja töökohas oleva seisuga repo üle kirjutada: '**git push -f origin master**'
- kuvada, millised branchid on olemas ning millised versioonid deploy-tud: '**git branch -v**'

kalle404 githubi repo info

- profiil
- docuwiki repo
- jaani repo