OULUN AMMATTIKORKEAKOULU

Toni Vaakanainen

PALVELIMIEN VALMISTELUT ITSENÄISEEN WEB-KEHITYKSEEN

PALVELIMIEN VALMISTELUT ITSENÄISEEN WEB-KEHITYKSEEN

Toni Vaakanainen
Opinnäytetyö
Kevät 2016
Tietojenkäsittelyn koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma, Web-sovelluskehitys

Tekijä: Toni Vaakanainen
Opinnäytetyön nimi: Palvelimien valmistelut itsenäiseen web-kehitykseen
Työn ohjaaja: Liisa Auer
Työn valmistumislukukausi- ja vuosi: Kevät 2016 Sivumäärä: 40

Opinnäytetyö käsittelee valmisteluja ja hyviä toimintatapoja web-kehitystä tehdessä. Opinnäytetyössä ei ole varsinaista toimeksiantajaa, mutta opinnäytetyön lopputuotetta on tarkoitus käyttää toimeksiannossa. Opinnäytetyön tavoitteena on valmistella kaksi tietokonetta web-palvelimiksi sekä esitellä Git-versionhallintatyökalun toimintaa ja käyttöä web-kehityksessä.

Opinnäytetyön lopussa kehityspalvelimet ovat käyttövalmiita web-kehityksen jatkamiseksi ja Git-versionhallinta on palvelimilla käytössä. Git-versionhallinta sisältää myös GitHubin palvelimella olevan WordPress-asennuksen pohjan, jonka voi siirtää mille tahansa Debian-pohjaiselle web-palvelimelle, joka täyttää WordPressin vaatimukset. Opinnäytetyön ohjeita seuraamalla pystyy luomaan web-palvelimen sekä käyttämään Git-versionhallinnan perustoiminnallisuuksia.

Jatkokehityksenä varsinaisen sivuston luomisen lisäksi ovat erilaiset automatisoinnit; esimerkiksi lähes kaiken tämän opinnäytetyön sisältämän toiminnallisuuden pystyy automatisoimaan opinnäytetyössä esiteltyjä komentoja käyttäen.

Asiasanat: LAMP, Git, WordPress

ABSTRACT

Oulu University of Applied Sciences
Business Information Systems

Author: Toni Vaakanainen
Title of thesis: Preparation of servers for independent web-development
Supervisor: Liisa Auer
Term and year when the thesis was submitted: Spring 2016 Number of pages: 40

This thesis presents preparations and good practices for independent web development. The thesis does not have a client, but the end product of the thesis can be used for assignments by actual clients. The aim of the thesis is to prepare two computers as web servers, and to present version control software Git. With Git the thesis should cover most basic usage and operations, mainly relating to web development.

At the end of the thesis web servers meant to be used in development are ready for use and Git version control is in use. Git version control includes clean installation of content management system WordPress. GitHub is used as the service provider for the Git server. Version controlled WordPress base can be moved on any Debian based web server, which fulfills the requirements of WordPress. By following the instructions of this thesis, creation of a web server should be possible, as well as use of Git.

This thesis leaves room for further development. The thesis could be continued by creating the actual website, for which the preparations are made for, and by automating some of the actions presented in the thesis.

Keywords: LAMP, Git, WordPress

SISÄLLYS

1 JOHDANTO

Tavoitteeni on kuvata hyviä toimintatapoja ja valmisteluja itsenäisen web-kehitystyön tekemiseen. Web-kehityksellä tarkoitan web-sisällön ja -palveluiden kehitystyötä sekä niiden ylläpitoon vaadittavien palvelimien ja palvelinohjelmistojen käyttöä. Mielestäni itsenäinen web-kehitys tarkoittaa sitä, että pystyy tekemään web-kehitystä ilman, että on riippuvainen muiden osaamisesta tai jostain tietystä palvelusta. Ulkopuolisia palveluita kuten web-hotellia tai GitHubia voi ja kannattaa käyttää tarpeen mukaan, mutta on hyvä olla tarpeellinen osaaminen, jotta voi vaihtaa palveluntarjoajaa tarvittaessa. Käsittelen siksi web-kehityspalvelimen ohjelmallisen rakentamisen sekä versionhallintaohjelmisto Gitin käyttöä. Oman Git-palvelimen rakentamista en kuitenkaan käsittele, koska käyttökelpoisia palvelimia on saatavilla ilmaiseksi.

Web-kehitystä voi tehdä monella tapaa. Yksinkertaisimmillaan tarvitaan vain tietokone, jolle on asennettu HTTP-palvelinohjelmisto. Kuvaamani työskentelytapa on kuitenkin pidemmälle viety. Kehityspalvelimien ohjelmallinen rakentaminen tyhjästä ja versionhallinnan käyttäminen saattaa vaikuttaa monessa tapauksessa raskaalta prosessilta. Kyky rakentaa oma palvelin ja käyttää versionhallintaa ovat kuitenkin mielestäni erittäin hyödyllisiä.

Monet eduista saattavat tulla ilmeisiksi vasta suuremmissa ja pidempikestoisissa projekteissa, joissa tavoitteena ei ole pelkästään sivuston tai palvelun julkaiseminen, vaan myös sen pitkäkestoinen ylläpito. Hyödyt näkyvät ennen kaikkea siinä, että sivuston tai palvelun käyttökatkokset on mahdollista pitää jatkokehityksen aikana mahdollisimman lyhyinä. Versionhallinnan käyttäminen mahdollistaa myös nopeat palautumiset mahdollisista virhetilanteista, kuten epäonnistuneesta tai virheellisestä päivityksestä.

Opinnäytetyössäni rakennetulle web-palvelimelle asennetaan myös WordPress-julkaisujärjestelmä. Syy WordPressin käyttöön on se, että esimerkkitoimeksiannossani kaikki henkilöt, jotka sivustoa tulevat päivittämään, eivät ole kehittäjätaustaisia. Julkaisujärjestelmän käyttö on tästä syystä tarpeen. Julkaisujärjestelmistä WordPress on minulle tutuin ja julkaisujärjestelmän valitseminen ei opinnäytetyössä ole keskeisessä osassa, joten päätin käyttää sitä. Toinen tunnettu vaihtoehto olisi ollut Joomla.

Käytän opinnäytetyössäni paljon englanninkielisiä termejä, koska kokemukseni mukaan kyseiset termit ja käsitteet ovat alalla varsin vakiintuneita, eikä juuri kukaan käytä suomenkielisiä vastineita. Suomenkielisten termien käyttäminen saattaisi aiheuttaa sekaannusta, koska käännökset ovat yleensä vapaita ja siten vaihtelevia.

2 PALVELIMIEN VALMISTELUT

Mielestäni pitkäkestoisessa, esimerkiksi vuosia jatkokehitettävän web-palvelun tai -sivuston, kehityksessä projektin alustus on projektin tärkein vaihe. Mikäli projektin valmistelut tekee huonosti, joutuu kyseisiin asioihin palaamaan aika-ajoin. Esimerkiksi sivuston tietokannan hajotessa versionhallinnan ja varmuuskopioinnin puutteet voivat jopa pysäyttää jatkokehityksen väliaikaisesti.

Valmistelut alkavat palvelinympäristöjen pystyttämisellä. Kuvaamassani DSP-mallissa (Development, Staging, Production) niitä on kolme tai useampi, riippuen kehittäjien määrästä. Vaihtoehtoisia kehitysmalleja ovat muun muassa DISP (Development, Integration, Staging, Production) ja DTSP (Development, Testing, Staging, Production). Opinnäytetyössä esitelty malli on tulkintani ja kokemukseni DSP-mallista, mutta siitä on myös monia muita tulkintoja. Esimerkiksi beanstalk Guides kuvaa oman versionsa DSP-mallista osoitteessa http://guides.beanstalkapp.com/deployments/best-practices.html.

Kuvaamassani mallissa ympäristöt ovat “production” (vaihtoehtoinen nimitys ”prod”), “staging” ja “development” (tai vaihtoehtoisesti ”devel” tai ”dev”). Lokaalit ympäristöt ovat tarvittavia tilanteessa, jolloin samaa tuotetta kehittää useampi henkilö yhtä aikaa. Näistä ympäristöistä käytetään yleensä termiä “Local” (eli lokaali).

Production-palvelin on se palvelin, joka sivustolla tai tuotteella on julkisesti käytössä, ja jota asiakkaat käyttävät. Production-ympäristöä en opinnäytetyössäni kuvaa, koska sen lopputulos ei ole vielä julkaistavissa oleva sivusto. Monesti production-palvelin on ostettu ulkopuoliselta palveluntarjoajalta ja syy tähän on käytännöllinen. Ulkopuoliset palveluntarjoajat ovat erikoistuneet palvelimien hallintaan ja niiden saavutettavuus on yleensä parempi kuin mihin kapasiteettia vuokraava yritys itse pystyisi. Pilvipalvelimiin erikoistuneet yritykset pystyvät myös vastaamaan kysyntään kustannustehokkaammin kuin yritys, joka ei ole kyseiseen toimintaan erikoistunut. Esimerkkinä kyseisistä palveluntarjoajista mainittakoon Amazon Web Services.

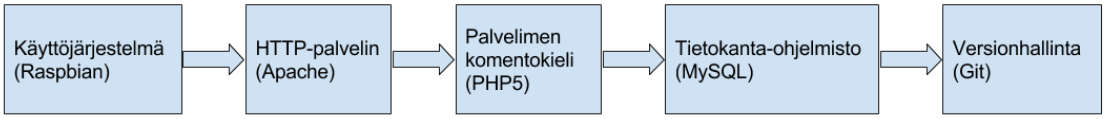
Staging-palvelin puolestaan on viimeisin kehityspalvelin, jonka versio on seuraavaksi menossa production-palvelimelle. Yleensä kyseisellä palvelimella tehdään viimeiset testit, joissa

tarkastetaan, että kaikki toimii kuten pitäisi. Production- ja staging-ympäristöjen pitäisi tästä syystä olla toiminnallisuuksiltaan mahdollisimman lähellä toisiaan.

Development-palvelin on yleensä se palvelin, johon kaikki kehittäjät vievät työnsä lokaalista ympäristöstään. Tämä palvelin voi olla kaikista epävakain, koska mahdolliset konfliktit ja ongelmat tulevat tässä ympäristössä ilmi. Palvelimen ja kehitettävän palvelun tilan pitäisi kuitenkin olla vakaa ennen kuin sen sisältö viedään staging-palvelimelle.

Local on kehittäjän oma ympäristö, omalla työkoneella. Local-ympäristö voi sisältää koko tuotteen tai ainoastaan esimerkiksi front-end-ympäristön, jolloin kehittäjä käyttää back-endinään joko devel-palvelinta tai jonkun muun kehittäjän lokaalia palvelinta.

Itsenäisen web-kehityksen valmisteluihin liittyvät asennukset on kuvattu alla. Vaiheet ovat käyttöjärjestelmän asennus, HTTP-palvelinohjelmiston asennus, palvelimen komentokielen asennus, tietokantaohjelmiston asennus sekä versionhallinnan käyttöönotto.



Kuva 1. Asennukset

Palvelinympäristöjen pystyttäminen alkaa käyttöjärjestelmän asentamisesta. *Devel*- ja *staging*-palvelinympäristöinäni toimii kaksi Raspberry Pi:tä, siitä syystä että kyseiset pientietokoneet ovat saatavillani. Devel Raspberry Pi on malliltaan Pi 1 model B+ ja staging-tietokone on Pi 2 model B. Mikäli kahta samanlaista tietokonetta ei ole käytettävissä, staging-ympäristön tietokoneen olisi hyvä olla tehokkaampi kuin devel-ympäristön.

Seuraavana vaiheena on WordPressin tarvitsemien palvelinohjelmistojen asennus. Kyseiset ohjelmistot ovat: Apache, PHP5 ja MySQL. HTTP-palvelimena käytän Apachea, koska minulla on siitä enemmän kokemusta kuin Nginxistä, joka on toinen varteenotettava vaihtoehto. Palvelimen komentokielenä on PHP5, jonka WordPress tarvitsee toimiakseen.

Kolmantena vaiheena on MySQL-tietokantapalvelimen asennus ja konfigurointi. Suurin osa WordPress-sivustoista käyttää MySQL-tietokantaa, MariaDB olisi toinen vaihtoehto. MariaDB:n

kehittäjät ovat suunnitelleet kyseisen tietokantaohjelmiston MySQL:n korvaajaksi, joten loppukäyttäjän näkökulmasta käytön ei pitäisi olla kovin erilaista. Valitsin kuitenkin käyttööni MySQL:n, koska siitä minulla on suurin osa kokemuksestani.

Viimeisenä osiona käsittelen mielestäni kaikista tärkeimmän osan eli versionhallinnan. Kyseisenä versionhallintaohjelmistona käytän Git:iä.

2.1 Palvelin

Palvelin on tietokone, joka toimittaa tietoa muille tietokoneille. Palvelin voi toimia ainoastaan lähiverkossa tai myös ulkoverkossa. Erilaisia palvelimia ovat esimerkiksi web-, sähköposti- tai tiedostopalvelimet. Yksi tietokone voi toimia useampana erilaisena palvelimena yhtä aikaa, tietokoneelle täytyy olla vain asennettuna sopiva palvelinohjelmisto. (TechTerms 2014, hakupäivä 17.10.2015.)

Mitä tahansa tietokonetta pystyy käyttämään palvelimena, mutta suurin osa isoista yrityksistä ja palveluntarjoajista suosii ”räkkitietokoneita”, jotka on suunniteltu palvelinkäyttöön. Kyseiset tietokoneet ovat yleensä (fyysisesti) mahdollisimman pieniä, sekä sisältävät (normaaleista tietokoneista poikkeavia) toiminnallisuuksia kuten (tietokoneen) toiminnan tilaa kuvaavia LED-valoja ja hot swap -tekniikalla toimivat kiintolevypaikat. Usein palvelintietokoneita hallitaan etäyhteydellä, jolloin tietokoneet eivät tarvitse omia hallintalaitteita. (TechTerms 2014, hakupäivä 17.10.2015.)

Opinnäytetyössäni käytän web-kehityspalvelimenani Raspberry Pi -tietokoneita. Raspberry Pi on erittäin pieni ja halpa tietokone, joka on varsin yleinen palvelintietokone harrastelukäytössä. Yleensä Raspberry Pi -tietokoneen käyttöjärjestelmänä toimii jokin versio Linuxista. Raspberry Pi:lle on myös täysin oma ja optimoitu Linux-versio Raspbian, joka pohjautuu yleiseen Debian jakeluversioon. Raspbian-pohjaisessa kehitysympäristössä erityisen hyvä puoli on se, että kehitystyön jälkeen lopputuotteen voi siirtää tehokkaammalle Debianiin perustuvalle palvelintietokoneelle.

W3Techs–sivusto ilmoittaa tilastoissaan Unix-pohjaisten web-palvelimien osuudeksi 67,2%, joista Linux–pohjaisiksi kerrotaan 53,1%, joista puolestaan Debian–pohjaisia voidaan sanoa olevan yli

60%, kun huomioidaan myös Ubuntun Debian–pohjaisuus. (W3Techs 2015a, hakupäivä 17.10.2015, W3Techs 2015b, hakupäivä 17.10.2015)

2.1.1 Raspbian

Linux on avoimeen lähdekoodin perustuva tietokonekäyttöjärjestelmä, joka koostuu Linus Torvaldsin alunperin kehittämästä kernelistä ja Richard Stallmanin aloittamasta GNU:sta. Linuxissa on paljon UNIX:ia muistuttavia piirteitä, joka on auttanut käyttöjärjestelmän yleistymisessä. (Linux Foundation 2009a, hakupäivä 17.10.2015, Linux Foundation 2009b, hakupäivä 17.10.2015.)

Raspbian on ilmainen käyttöjärjestelmä, joka perustuu Debianiin, ja on optimoitu Raspberry Pi:n laitteistolle (Welcome to Raspbian, hakupäivä 17.10.2015). Myös Debian on ilmainen käyttöjärjestelmä, joka puolestaan perustuu Linux tai FreeBSD kerneliin (Software in the Public Intreset, Inc 2015, hakupäivä 17.10.2015).

Raspbianin asennus alkaa lataamalla uusin versio Raspberry Pi Foundationin sivuilta https://www.raspberrypi.org/downloads/raspbian/, kirjoittamishetkellä kyseinen versio on Raspbian Jessie, joka perustuu Debian Jessieen. Asennusohje löytyy myös samalta sivustolta. Käytän asentamiseen Windows-tietokonetta, joten seuraan ohjetta https://www.raspberrypi.org/documentation/installation/installing-images/windows.md.

Vaiheet ovat: ladataan käyttöjärjestelmä zip-tiedostona, puretaan zip-tiedosto ja kirjoitetaan purettu kuva Win32DiskImager–ohjelmalla muistikortille. Kun käyttöjärjestelmä on kirjoitettu muistikortille, seuraava vaihe on laittaa muistikortti laitteeseen ja käynnistää ensimmäisen kerran. Ensimmäisellä käynnistyskerralla määritellään käyttöjärjestelmän asetukset.

Raspbianin asetusten määrittelyyn pääsee komentoriviltä komennolla `sudo raspi-config`. Määritin näppäimistön ja lokaalin suomeksi, mutta jätin käyttöjärjestelmän englanniksi. Tärkeimmät osiot muistaa määrittää asetuksissa ovat kuitenkin: expand file system, boot options, ssh ja hostname.

`Expand file system` -valinta saa Raspbianin tarkistamaan muistikortin koon ja ottamaan sen täysimääräisenä käyttöön. Varsinaisessa asennusvaiheessa käyttöjärjestelmä ottaa käyttöönsä ainoastaan tarvittavan minimitilan, joten tämän komennon ajaminen on tärkeää. `Boot optionsista` on puolestaan hyvä määrittää käyttöjärjestelmä käynnistymään oletuksena vain komentoriville, graafisella puolella ei Raspberry Pi:n tehoilla tee juuri mitään, ja sen käynnistäminen oletuksena vie vain laitteelta turhaan resursseja. `Advanced optionsin` alla on tärkeää muistaa sallia SSH yhteyksien avaaminen laitteeseen, jotta laitetta voi käyttää ilman näyttöä. Myös `hostname` kannattaa määrittää, eli antaa laitteelle oma uniikki nimi verkossa. Määritysten jälkeen käyttöjärjestelmä pyytää lupaa uudelleen käynnistymiseen, jotta muutokset tulevat voimaan.

Seuraavaksi Raspberry Pi:hin täytyy saada etäyhteys muodostettua, jotta laitteen voi irrottaa näytöstä ja näppäimistöstä. Laitteelle kannattaa määrittää staattinen IP-osoite reitittimen jakamien IP-osoitteiden ulkopuolelta tai määrittää reitittimen puolelta vakio-osoite, jottei laitteen IP-osoitetta tarvitse etsiä joka kerta uudelleen. Päädyin määrittämään tietokoneelle reittimeen vakio-osoitteen, joka tietokoneelle annetaan MAC-osoitteen perusteella. Tällöin laite toimii myös tarvittaessa eri verkoissa ilman että laitteen omiin internet-asetuksiin täytyy koskea. Yleensä olen määrittänyt asetuksen suoraan Raspbianiin, mutta Jessie-versiossa käyttöjärjestelmä käyttää dhcpcd-ohjelmistoa internet-asetusten määrittämiseen, joka ei ole minulle tuttu.

Huomioitavaa: jouduin kesken opinnäytetyöni vaihtamaan käyttämäni Raspbian-version vanhempaan Wheezy-versioon. Syynä tähän oli uusimmassa versiossa ollut virhe, joka aika-ajoittain rikkoi käyttöjärjestelmän APT-ohjelmiston (Advanced Packing Tool). En löytänyt ongelmaan muuta nopeaa ratkaisua kuin käyttöjärjestelmän uudelleen asennuksen, jonka vuoksi kaikki työ täytyy tehdä uudestaan. Koska en halunnut törmätä samaan ongelmaan uudestaan, vaihdoin vanhemman käyttöjärjestelmäversion, jossa kyseistä ongelmaa ei ole. Kaikki ohjeet, jota opinnäytetyössäni kuvaan Raspbianiin liittyen, pitävät kuitenkin paikkansa myös Wheezy-versiossa.

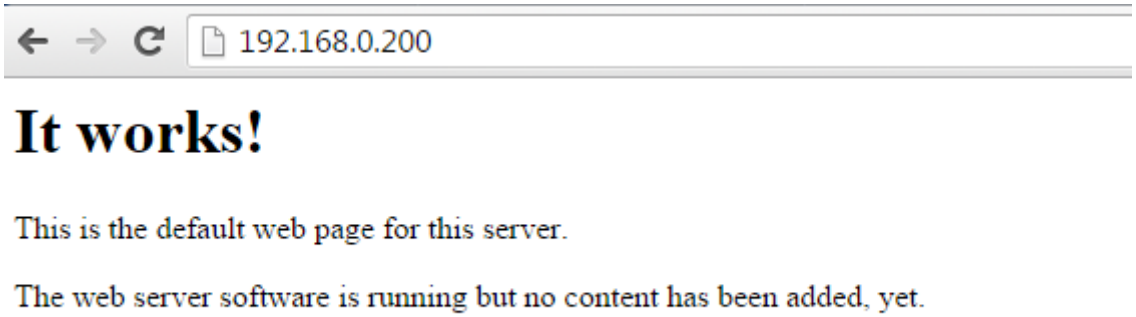
2.1.2 Apache

Mitä tahansa tietokonetta voi käyttää palvelimena, kunhan siinä on internet yhteys ja soveltuva ohjelmisto asennettuna (TechTerms 2011, hakupäivä 30.1.2016). HTTP-palvelimella tarkoitetaan yleensä palvelimen ohjelmistoa, joka vastaa HTTP-liikenteen käsittelystä.

Käytetyimmät vapaasti käytettävissä olevat HTTP-palvelinohjelmistot ovat Apache ja Nginx. W3Techsin mukaan Apache on selkeästi yleisimmin käytetty HTTP-palvelinohjelmisto 55,2% käyttöosuudella, seuraavana tulee Nginx 27% ja kolmantena Microsoft-IIS 12,3%. Kaikkien muiden kilpailevien ohjelmistojen käyttöosuus on alle 2,5% per ohjelmisto. (W3Techs 2016, hakupäivä 30.1.2016.)

Apachen asentaminen Debian-pohjaisessa järjestelmässä tehdään käyttämällä APT package handling utility -nimistä ohjelmaa ajamalla komento `sudo apt-get install apache2`. Luonnollisesti tätä ennen kannattaa muistaa ajaa komennot `sudo apt-get update` ja `sudo apt-get upgrade`, kuten aina ennen uuden ohjelmiston asentamista. Kyseiset komennot päivittävät Linuxin APT-ohjelmiston tiedot uusimmista päivityksistä ja ohjelmistoversioista. Jälkimmäinen komento päivittää kaikki ohjelmistot joille päivitys löytyy. Mikäli unohtaa ajaa update tai upgrade komennon, voi törmätä yhteensopivuusongelmiin tai ohjelmistosta voi vahingossa asentaa vanhentuneen version.

Asentamisen jälkeen kannattaa heti tarkistaa, että Apachen asennus onnistui navigoimalla selaimella palvelimen IP-osoitteseen. Apache luo asennuksen yhteydessä itselleen index-sivun, joka on oletussivu, jonka palvelin palauttaa. Ohessa ruudunkaappaus kyseisestä sivusta.

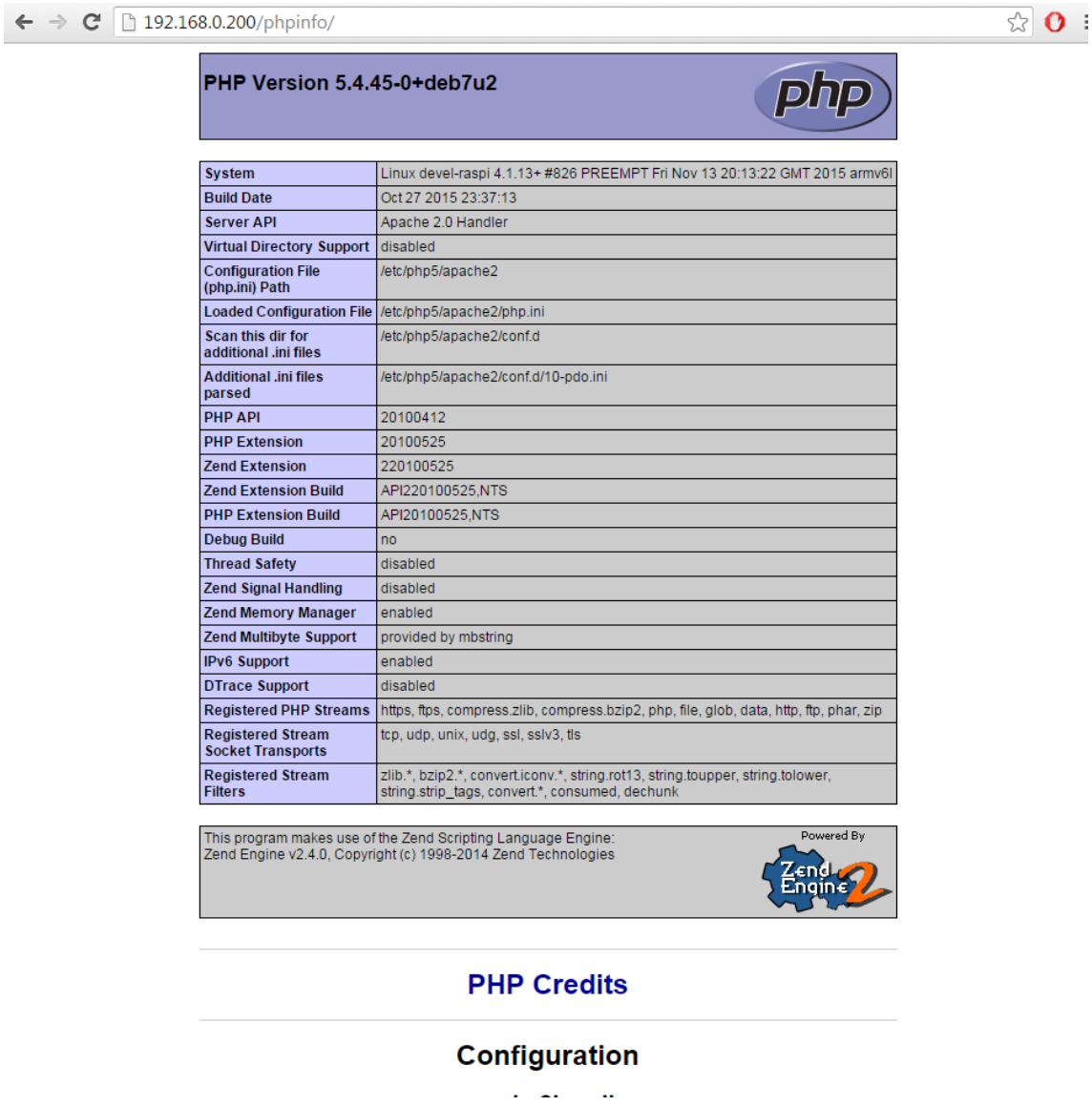


Kuva 2. Apache: It works!

2.1.3 PHP5

Web-palvelin tarvitsee yleensä myös oman komentokielensä, jotta palvelin pystyy palauttamaan muutakin kuin staattisia HTML-sivuja. Palvelinkomentokieliä ovat muun muassa PHP, Python, Perl, Java ja Node.js. Apachen kanssa yleisimmin käytetään PHP:tä. Aion käyttää kyseisellä palvelimella WordPressiä, joten uusimpaan WordPress-versioon tarvitsen PHP:n ja versionumeron täytyy olla vähintään 5.6.

PHP:n asentamiseksi Apache:n yhteyteen käytettävä komento on `sudo apt-get install php5 libapache2-mod-php5`. PHP:n toiminnallisuus kannattaa testata heti asennuksen jälkeen. Normaalisti Debian-pohjaisissa palvelimissa Apachen internetiin aukioleva kansio on joko "/var/www" tai uudemmissa versioissa "/var/www/html". Käyttämässäni versiossa, joka on Raspbian Wheezy, var/www. PHP:n toiminnallisuuden voi testata luomalla uuden kansion /var/www:n sisään, kansio on nimeltä phpinfo, ja kyseisen kansion sisään index.php-tiedoston, jonka koodisisältö on `<?php phpinfo(); ?>`. Navigoimalla selaimella palvelimen ip-osoitteeseen "{ipNumero}/phpinfo" palvelin kertoo kaikki PHP:n olennaiset tiedot. Ohessa kuva kyseiseltä sivulta.



Kuva 3. PHP-info

Index.php-tiedosto kannattaa ehdottomasti poistaa ennen kuin palvelin avataan saavutettavaksi internetistä, koska kyseinen tiedosto kertoo palvelimesta aivan liikaa tietoa ja paljastaa siten mahdolliset tietoturvahaavoittuvuudet.

2.1.4 MySQL-tietokantaohjemisto

MySQL:n kehittivät Michael Widenius ja David Axmark, MySQL AB:n perustajat. Ensimmäinen versio MySQL-tietokantaohjelmistosta julkaistiin 1995. Sun Microsystems osti MySQL AB:n vuonna 2008. (Rahabu Ransagar 2009, hakupäivä 21.3.2016.) Oracle puolestaan osti Sun Microsystemsin vuonna 2010. (Oracle (ei julkaisuvuotta), hakupäivä 21.3.2016.)

Nykyisin Oracle omistaa ja jatkokehittää MySQL-tietokantaohjelmistoa. Ohjelmisto on avoimeen lähdekoodiin perustuva ja vapaassa käytössä, sillä rajoituksella että ohjelmiston liiketoiminnallinen käyttö vaatii joissain tapauksissa lisenssin Oraclelta (Baron Schwartz 2009, hakupäivä 1.6.2016).

MySQL on nykyisin yksi yleisimmin käytetyistä tietokantaohjelmistoista (solid IT 2016, hakupäivä 2.6.2016). Osasyynä suosioon on todennäköisesti se, että muun muassa WordPress on suunniteltu käyttämään MySQL-tietokantaa. MySQL on myös osa perinteistä LAMP-palvelinta (Linux, Apache, MySQL, PHP), jollaisen opinnäytetyössäni rakennan.

MySQL ohjelmiston asentamiseksi käytetään komentoja:

1. `sudo apt-get install mysql-server php5-mysql`. Komento asentaa MySQL-palvelinohjelmiston sekä paketin, joka mahdollista PHP5-skriptien ottaa suoran yhteyden MySQL-tietokantaan.
2. `sudo mysql_secure_installation`. Komento korjaa useita haavoittuvuuksia perusasetuksista.

Mikäli root-käyttäjän salasanaa ei ole asennuksen yhteydessä määrittänyt, se kannattaa tehdä viimeistään tietoturvavahvennuksen aikana. Tietoturvavahvennuksen vaiheet ovat:

1. Root käyttäjän salasanan asetus/vaihto. Tähän osioon voi vastata turvallisesti ei, mikäli kunnollisen salasanan on laittanut jo asennuksen yhteydessä.
2. Poistetaanko anonyymit käyttäjät. Kyllä.
3. Estetäänkö root-käyttäjänä kirjautuminen lähiverkon ulkopuolelta. Tässä tapauksessa kyllä, koska aion aina ensin kirjautua etäyhteydellä palvelimelle ennen kuin muokkaan root-käyttäjänä tietokantaa. Vaikka olisin muualla, komennot tulevat tällöin samalta tietokoneelta. Poikkeuksia eri käyttötapauksissa tähän voi olla. Mikäli ei ole varma tarvitseeko kyseisen ominaisuuden, toiminnallisuus kannattaa poistaa käytöstä, koska se saattaa aiheuttaa aukon tietoturvassa.
4. Poistetaanko testitietokanta. Kuten ohjetekstikin kertoo, tietokanta on vain testaustarkoitukseen eli se kannattaa poistaa. Kyseisen osion suoritus saattaa myös epäonnistua, mikäli testitietokantaa ei jostain syystä ole olemassa, mutta sillä ei ole väliä.
5. Päivitetäänkö muutetut käyttäjäoikeudet heti. Kyllä.

2.1.5 SFTP ja Git

Joidenkin käyttöjärjestelmien tapauksessa tiedostopalvelinohjelmisto täytyy asentaa käyttöjärjestelmään erikseen, mutta esimerkiksi Debian-pohjaisessa Raspbianissa ohjelmisto löytyy oletusasennuksesta valmiina. Tällöin käyttäjän täytyy asentaa vain työskentelytietokoneelle (S)FTP-client -ohjelmisto. Itse käytän esimerkiksi FileZilla-ohjelmistoa.

Monissa Linux versioissa Git-versionhallintatyökalu on sisällytetty perusasennukseen ja näin on myös Raspbianin tapauksessa, joten Gitiä ei tarvinnut erikseen asentaa palvelimelle.

2.2 Git-versionhallinta

Git on hajautettu versionhallintajärjestelmä, jota pääasiassa käytetään ohjelmistokehityksessä. Gitin ero esimerkiksi Subversion-versionhallintajärjestelmään on se, että Git-versionhallinta toimii myös lokaalissa ympäristössä, eikä muutoksia tarvitse välttämättä viedä etäpalvelimelle tallentaakseen. (Daniel J. McGlinn (ei julkaisuvuotta), hakupäivä 16.5.2016.)

Gitin kehityksen aloitti Linus Torvalds, mutta nykyisin sitä ylläpitää Junio C Hamano. Torvalds antoi ohjelmistolle viralliseksi nimeksi: git - the stupid content tracker. (Git man page, hakupäivä 16.3.2016.)

Linus Torvalds kirjoitti Gitin perustan noin kymmenen vuotta sitten Linux projektin menetettyä mahdollisuuden käyttää BitKeeper-versionhallintajärjestelmää. Koska Linux-kernelin kehittäjät eivät pystyneet löytämään mieluista korvaavaa versionhallintajärjestelmää, Torvalds päätti luoda uuden itse. Gitin perusta luotiin alle viikossa. (Jennifer Cloer 2015, hakupäivä 16.3.2016.)

Git-ohjelmisto on lisensoitu, kuten Linux kernel, GNU GPL -lisenssillä. Lisenssi tarkoittaa, että ohjelmisto on ilmainen ja vapaasti hyödynnettävissä sekä muokattavissa.

Gitin käyttöönotto on matalakynnyksinen, koska Git-palvelinta ei käytännössä tarvitse itse pystyttää. Saatavilla on tunnettuja ja luotettuja toimijoita, joiden Git-palvelimet ovat käytettävissä ilmaiseksi tietyin rajoituksin, kuten GitHub ja Bitbucket.

2.2.1 Gitin käsitteitä

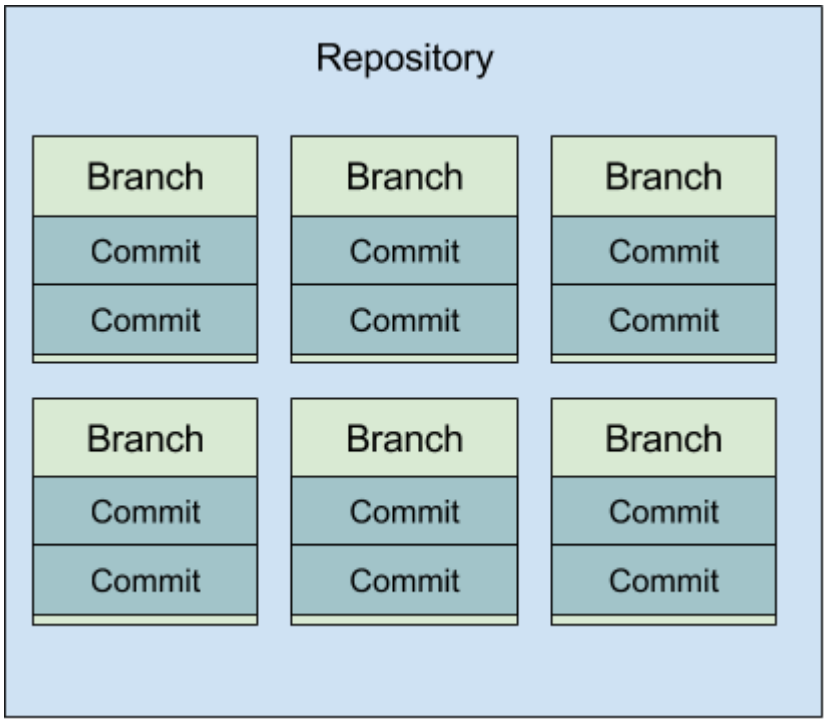
Gitin peruskäsitteitä ovat muun muassa repository, branch, commit, fork, remote ja local.

Repository Git-projektin päätaso. Repository pitää sisällään yhden tai useamman koodihaaran (branch). Usein lyhennetty “repo”.

Branch Branchit ovat Gitin ydin. Kaikki työ tapahtuu jollakin branchilla. Kaikki tiedostot, joita käyttäjä paikallisesti muokkaa, muokkautuvat ainoastaan kyseisellä branchilla. Muutokset täytyy tallentaa (commit), jotta ne voidaan lähettää (push) palvelimelle. Muutokset tapahtuvat palvelimella ainoastaan saman nimiselle branchille (mikäli sellaista ei ole, se luodaan samalla).

Commit Commit on yksi tila-tallennus banchin sisällä. Branchin tilaa pystyy hallitsemaan commitien perusteella. Branchin voi esimerkiksi palauttaa 5 commitia historiassa taaksepäin.

Mikäli palautus versiohistoriassa tehdään komennolla `soft reset`, tilan jälkeiset muutokset jäävät olemaan, mutta ovat tallentamattomina. Seuraavassa commitissa oletusarvoisesti kaikki muutokset tallennetaan uudestaan, mutta kaikki commitit jotka olivat ennen kyseistä tilaa katoavat. Mikäli palautus tehdään komennolla `hard reset` kaikki muutokset, jotka on tehty tilan jälkeen johon palataan, katoavat. Ohessa yksi näkemys repositorystä, brancheistä sekä commiteista.



Kuva 4. Repository

Fork	Kopio toisesta repositorystä. Kopio voi olla joko täydellinen tai vain osittainen. Kopiointi hetken jälkeen kummatkin repositoryt voivat edetä vapaasti eri suuntiin.
Remote (repository)	Etärepository.
Local (repository)	Paikallisella tietokoneella oleva repository.

2.2.2 GitHub

GitHub ylläpitää Git-palvelimia, jolloin kyseisen palvelun käyttäminen säästää käyttäjän vaivalta pystyttää ja ylläpitää Git-palvelin itse. Oman Git-palvelimen kanssa ongelmaksi saattaa muodostua palvelun ylläpito sekä saavutettavuus. GitHubin liiketoiminta perustuu siihen, että julkisten vapaan lähdekoodin repositorien luominen ja käyttäminen on ilmaista, mutta mikäli haluaa repositoryn olevan yksityinen, palvelu maksaa (Kakul Srivastava, 2016. Hakupäivä 1.6.2016).

Kehitystyössäni päätin käyttää GitHubia aikaisempien positiivisten kokemusten perusteella.

Päätin investoida myös pienimpään maksulliseen GitHub-palveluun, koska ilmaisessa versiossa GitHub sallii ainoastaan julkisten repositorien luonnin. Teen opinnäytetyössäni valmisteluja toimeksiantoon, joten julkinen GitHub repository ei tässä tapauksessa ole mahdollinen.

Kesken opinnäytetyöni GitHub uudisti hinnoittelumallinsa. Uudessa hinnoittelumallissa GitHub yksinkertaisti hinnoittelumalliaan ja poisti maksavilta asiakkailtaan rajoitukset repositoryjen määrästä. Uusi hinnoittelumalli perustuu käyttäjämääriin. (Kakul Srivastava, 2016. Hakupäivä 1.6.2016.)

2.2.3 Gitin peruskäyttö

GitHubiin yhteyden voi muodostaa joko HTTPS- tai SSH-yhteydellä. GitHub suosittelee yhteydenpitoa heidän palvelimiensa ja oman koneen välillä HTTPS-yhteydellä siksi, että HTTPS-yhteys toimii todennäköisemmin palomuurien ja proxyjen läpi (GitHub, (ei julkaisuvuotta). Hakupäivä 1.6.2016). SSH-yhteys on myös osaltaan haavoittuvaisempi kuin HTTPS-yhteys. Mikäli joku pääsee koneelle, jolla SSH-yhteyden muodostamiseen käytetty avain on tai saa avaimen muuten haltuunsa, henkilö saa rajoittamattomat etähallintaoikeudet tunnukselle.

Olen kuitenkin päätynyt käyttämään SSH-yhteyttä, koska se säästää vaivalta joutua kirjoittamaan oma käyttäjätunnus ja salasana joka kerta kun haluaa hakea muutoksia palvelimelta tai lähettää muutoksia palvelimelle. Kuvailen Gitin käyttöä SSH-yhteyttä hyödyntäen, mutta ero HTTPS-yhteyden käyttämiseen ei kuitenkaan ole merkittävä. SSH-yhteyden luomiseksi käyttäjän täytyy ensin luoda itselleen SSH-avain ja syöttää avaimen julkinen osa GitHubin palveluun.

Git tarjoaa erittäin monipuolisia versionhallintatoiminnallisuuksia, joiden kattavat tiedot löytyvät komennolla `man git`. Yleisimmät komennot ja toiminnallisuudet ovat: clone, pull, push, fetch, merge, checkout, branch, status, add, reset ja commit.

git clone {mikä}	Luo kansion hakemistoon, jossa komento ajetaan. Kansion nimi on sama kuin repositoryn, joka kopioidaan. Kyseinen hakemisto sisältää repositoryn master-branchin ja sen sisällön. Mikäli joku muu branch halutaan ladata aktiiviseksi, hakemistoon täytyy suorittaa

	erillinen komento. Clone on ensimmäinen komento, joka pitää ajaa paikallisella koneella vanhan repositoryn lokaalin kopion luomiseen.
git pull {mikä}	Hakee kaikki viimeisimmät muutokset etä-repositorystä ja tekee automaattisen mergen lokaaliin/lokaaleihin brancheihin. Itse pyrin välttämään git pull -komennon käyttämistä, koska git fetch ja git merge -komennot tekevät saman asian, mutta antavat mahdollisuuden suurempaan hallintaan, esimerkiksi olla tekemättä mergeä, kun huomaa muutoksen tapahtuneen.
git fetch {mikä}	Hakee viitteet etä-repositorystä, muttei tee kyseisillä viitteillä mitään, kertoo vain mihin brancheihin muutoksia on tullut.
git merge {mikä}	Yhdistää nimetyn branchin kyseisellä hetkellä aktiivisena olevaan branchiin.
git push {minne} {mikä}	Vie tallennetut muutokset etäpalvelimelle. Mikäli etäpalvelimella ei ole kyseisen nimistä branchia, se luodaan. Mikäli on, kyseiselle etä-branchille tehdään automaattinen merge. Mikäli puskettavaa branchia ei ole päivitetty olemaan ajantasalla etäbranchin kanssa, komento saattaa epäonnistua. Tilanteen pystyy yleensä korjaamaan hakemalla muutokset etäbranchiltä git pull tai git fetch ja git merge -komennoilla, ja korjaamalla mahdolliset konfliktit lokaalissa ympäristöstä ennen mergen loppuunviemistä.
git commit	Tallentaa lisätyt muutokset. Komennon jälkeen commit-viesti täytyy kirjoittaa, komento käynnistää koneen oletustekstieditorin, johon osa viestistä on esitäytetty. Kun tiedostoon kirjoittaa viestin, tallentaa ja sulkee, commit viedään loppuun. Mikäli viestin editointivaiheessa huomaa tehneensä virheen, commit-komennon pystyy keskeyttämään tyhjentämällä viestin. Tyhjä commit-viesti on siis merkki ohjelmistolle keskeyttää.

git add {mitä}	Kertoo Gitille, että tietyt tiedostot pitää tallentaa seuraavassa commitissa. Muutokset, joita ei ole lisätty tallettevaksi kyseisellä komennolla, jäävät talteen seuraavan commitin jälkeen, mutta kyseiset muutokset ovat olemassa ainoastaan lokaalissa ympäristössä.
git status	Kertoo käyttäjälle, mikä on lokaalin branchin tila verrattuna etäbranchiin, esimerkiksi onko lokaali-branch edellä vai jäljessä etäbranchia, vai onko kummallakin branchillä tallennettuja muutoksia, joka aiheuttaa tilanteen, jossa branchit ovat eri tilassa. Komento myös listaa tiedostot ja hakemistot, joissa on muutoksia. Se kertoo, mitkä näistä muutoksista on merkattu tallennettavaksi ja mitkä eivät. Se kertoo myös mitä tiedostoja tai hakemistoja ei ole ollenkaan olemassa versionhallinnassa.
git reset HEAD {mikä}	Poistaa tiedoston muutokset tallennettavien tietojen listalta. Muutokset ovat vielä olemassa ja ne eivät katoa lokaalista ympäristöstä. Muutoksia ei enää tämän komennon jälkeen tallenneta seuraavassa commitissa, ellei niitä lisätä uudestaan.

2.2.4 Git branching

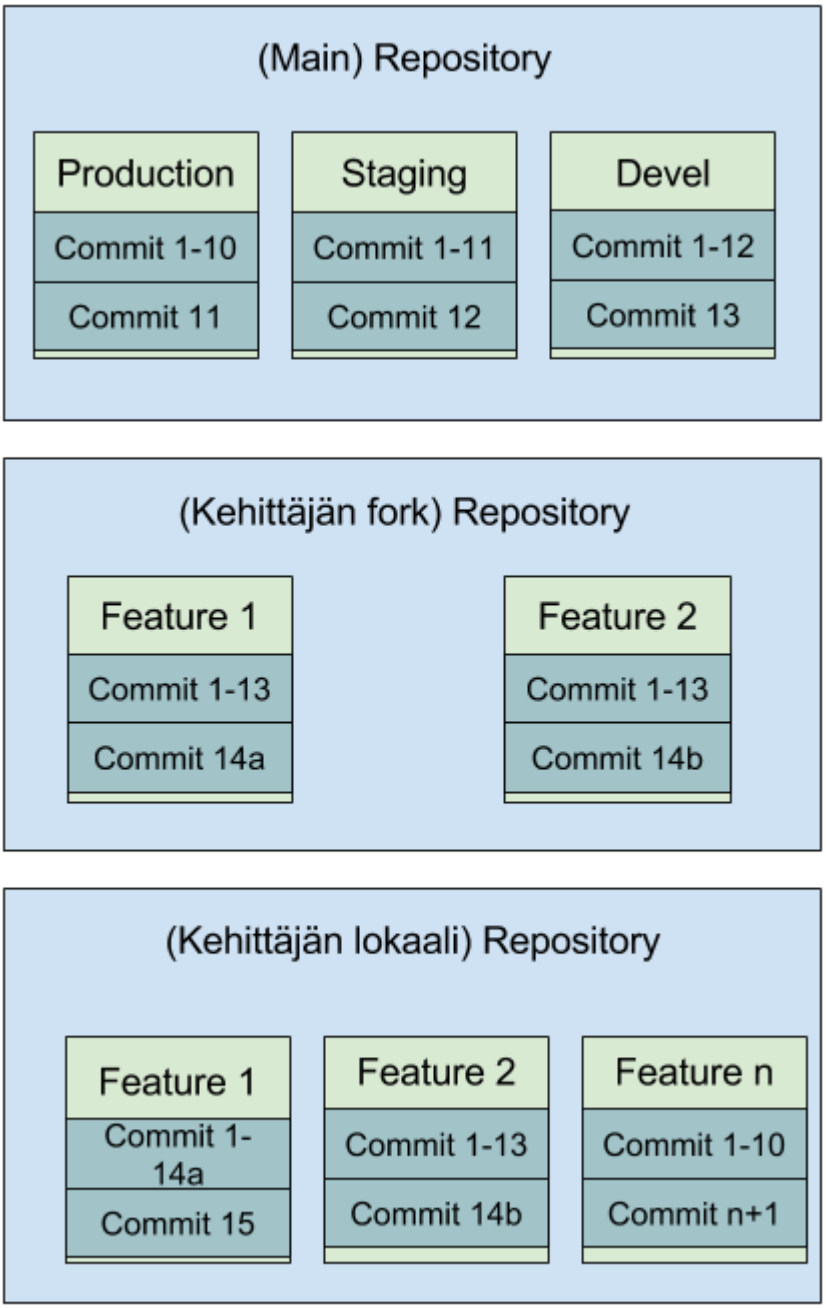
Git-versiohallitulla projektilla on oma *repository*, josta jokaisella kehittäjällä voi olla oma *fork* eli eräänlainen kopio, joka seuraa projektin päärepositoryn tilaa. Tämä kehittäjän oma *fork* seuraa kehittäjän oman lokaalin kehitysympäristön tilaa, ja on se paikka johon kehittäjä vie omat muutoksensa. Vaihtoehtoisesti kehittäjän lokaalikehitysympäristö voi myös seurata projektin päärespositoryä.

Projektin päärepositoryllä voi olla useampi haara eli *branch*, jotka kuvaavat projektin eri tiloja. Gitin brancheja voi hallita ja nimetä monella tapaa. DSP-mallissa mielestäni loogista on, että haarat ovat ”production”, ”staging” ja ”development”. Haaroja voi olla myös useampia riippuen kehittäjän mieltymyksistä.

Kuvaamassani mallissa production-haara on julkisesti käytössä oleva versio, esimerkiksi web-palvelussa tämä on julkinen versio sivustosta. Staging-haara on productionin esiversio ja on käytössä production-ympäristöä vastaavassa testitilassa. Tämä haara on se, jota pääasiassa testataan ja johon mahdolliset bugi-korjaukset tehdään.

Useimmin muuttuva haara puolestaan on devel, johon kehittäjät tekevät *pull requestinsa*, eli johon uusimmat koodimuutokset kehittäjiltä tulevat. Kyseisen haaran sisältämä koodi voi olla usein rikki. Devel-haaraa ei ole järkevää testata, koska tätä haaraa ja sen palvelinta kehittäjät käyttävät testiympäristönään oman lokaalin ympäristönsä lisäksi. Esimerkiksi silloin kun kaksi kehittäjää työskentelee saman toiminnallisuuden parissa.

Ohessa esimerkkikuva eri repositoryjen mahdollisista tiloista.



Kuva 5. Git forkit ja branchit

2.3 Varmuuskopionti

Erillistä varmuuskopiointia ei tarvitse oikein tehdyn versionhallinnan lisäksi. Versionhallinta on mielestäni tehty oikein, kun versionhallinnan eri kokonaisuudet eivät ole vain yhdellä tietokoneella. On hyvä huomioida, että mikäli ne ovat yhdellä ulkopuolisella palvelulla, tämän palveluntarjoajan täytyy olla luotettava.

Koodin ollessa versionhallinnassa useammalla tietokoneella, on todennäköistä, että suurin osa koodista saadaan palautettua, vaikka jokin tietokoneista hajoaisi. Eri tietokoneilla olevat versiot eivät kuitenkaan välttämättä ole täysin samassa tilassa, joten pientä hävikkiä voi ilmetä. Mahdolliseen ongelmaan kannattaa varautua viemällä kriittiset muutokset mahdollisimman usein versionhallintaan, josta muutokset leviävät mahdollisimman nopeasti useammalle tietokoneelle. Mielestäni Git-versionhallinnointi on tehty hyvin, kun etärepository on jossain luotettavassa palvelussa, esimerkiksi GitHubissa tai BitBucketissa.

Vaihtoehtoisesti oman Git-palvelimen kanssa olisi aiheellista, että on olemassa etärepository erillisellä tietokoneella kuin missä kehittäjien omat repositoryt ovat, eli käytännössä kaksi erillistä Git-palvelinta. Tällaisessa tapauksessa jomman kumman palvelimen pettäessä todennäköisesti kaikki koodi saadaan palautettua.

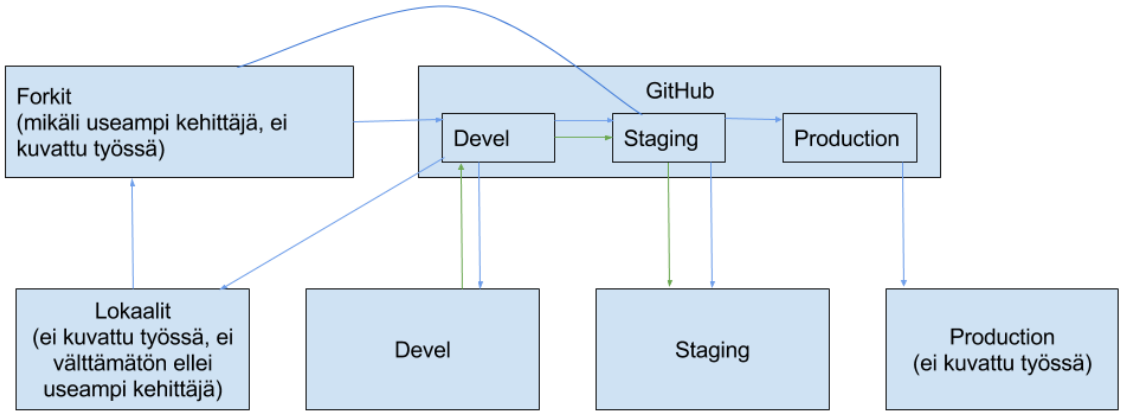
Mikäli Git-palvelin, jossa on muut kuin kehittäjien omat repositoryt, hajoaa, todennäköistä on, että kehittäjien palvelimella löytyy sama koodi. Kyseinen koodi kun on tämän kehittäjien Git-palvelimen kautta kulkenut. Vaihtoehtoisesti kehitys-Git-palvelimen hajotessa on todennäköistä, että kehittäjillä itsellään on ainakin tärkeimmistä brancheistä kopiot omilla koneillaan, joilta he voivat palauttaa kyseiset branchit uudelle korvaavalle palvelimelle.

Todennäköisesti heikoin kohta tässä lenkissä on kehittäjien omat tietokoneet. Mikäli kehittäjän tietokone hajoaa, viimeisimmät lokaalit muutokset eivät todennäköisesti ole missään muualla. Hyvä nyrkkisääntö onkin tehdä vähintään yksi tallennus omaan etärepositoryyn kerran päivässä, mikäli on muutoksia tehnyt.

MySQL-tietokannan sisältö ei oletusarvoisesti ole Gitin versionhallinan piirissä, koska tietokanta on asennettu Git-ohjelmiston tarkkaileman kansion ulkopuolelle. Tallennukset pitää tehdä siis manuaalisesti tekemällä MySQL dump aika-ajoin Git-kansion sisälle. Dump-tiedosto kannattaa tallentaa edellisen päälle, jottei niitä ala kertyä liikaa. Git-käsittelee tätä SQL tiedostoa kuten mitä tahansa muutakin tiedostoa. Uuden dumpin ylikirjoittaessa vanhan tiedoston ohjelmisto tunnistaa, että tiedosto on muuttunut, ja seuraavassa tallennuksessa versiointi toimii samalla tavalla kuin muissakin tiedostoissa. Vanhemman dump-tiedoston pystyy palauttamaan repositorystä kuten muutkin tiedostot.

3 TYÖSKENTELY ASENNUSTEN JÄLKEEN

Staging- ja devel-palvelimille on asennettu Raspbian Wheezy, Apache, PHP5 ja MySQL. Vuorossa on WordPressin asennus devel-palvelimelle, asennuksen vieminen GitHubiin sekä asennuksen kopioiminen staging-palvelimelle GitHubista. Ohessa kuvaus prosessista.



Kuva 6. Git-prosessi

Kuvassa on esitetty mahdolliset ympäristöt, jotka kuvaamassani työskentelymallissa voi olla. Lokaalit ympäristöt ja forkit eivät ole välttämättömiä, ellei projektissa ole useampi kehittäjä, jotka työskentelevät samanaikaisesti. Tästä syystä olen jättänyt ne pois opinnäytetyöstä. Olen myös rajannut production-ympäristön opinnäytetyön ulkopuolelle.

Kuvassa vihreät nuolet merkkaavat kuvaamaani prosessia. Siniset nuolet kuvaavat vaihtoehtoista prosessia, joka on viety loppuun sakka. Kuvassa alemmalla rivillä on kaikki varsinaiset työskentely-ympäristöt. GitHub-laatikon sisällä olevat laatikot ovat branchejä. Staging-branchin ja forkit yhdistävä kaari kuvaa mahdollisia bugi-korjauksia suoraan kehittäjän forkista, tämä ei kuitenkaan välttämättä ole hyvä käytäntö. Vaihtoehtoisesti muutokset voi *cherry pickata* devel-branchista, mutta tätä toiminnallisuutta en ole kuvannut tai käsitellyt opinnäytetyössä.

Seuraavien lukujen esimerkeissä esiintyy Lähin Pizza. Kyseessä on yritys, jonka kehitystyön valmisteluja opinnäytetyöni ohessa teen.

3.1 WordPressin asentaminen devel-palvelimelle

Luodaan palvelimelle kansio sivustolle, hakemiston ”/var/www” alle komennolla `sudo mkdir lahinpizza`. Sudoa täytyy käyttää, koska ”var/www” kuuluu root-käyttäjälle.

Linux-ympäristöissä tiedoston omistaa aina jokin käyttäjä ja ryhmä, tästä syystä käyttäjää ja ryhmää joutuu välillä vaihtelemaan tai vaihtoehtoisesti muokkamaan tiedostojen käyttöoikeuksia yleisesti. Vaihdetaan luodun kansion omistajaksi oma käyttäjä sekä Apahcen ryhmä, eli tässä tapauksessa ”pi” ja ”www-data”, komennolla `sudo chown pi:www-data lahinpizza`. Luodaan index-sivu, jotta voidaan testata että kaikki palvelimella toimii oikein, komennoilla: `cd lahinpizza; touch index.html; sudo chown pi:www-data index.html; vim index.html`. Kirjoitetaan HTML-tiedostoon mikä tahansa viesti ja navigoidaan selaimella tarkistamaan, että viesti näkyy oikeassa osoitteessa ”{ip-osoite}/lahinpizza/index”.

Tehdään symbolinen linkki oman käyttäjän kotihakemistoon, jotta käyttäjän kotihakemistosta on oikopolku työskentelykansioon, tämä helpottaa myös SFTP:n käyttöä. Minun tapauksessani oikea hakemistopolku on ”home/pi”, komento `ln -s /var/www/lahinpizza lahinpizza` luo symboolisen linkin. Ladataan uusin suomenkielinen WordPress osoitteesta https://fi.wordpress.org/, kirjoitushetkellä uusin versio on 4.4.2. Siirretään zip-paketti palvelimille SFTP-ohjelmistolla.

Ennen purkamista, pitää muistaa poistaa index.html tiedosto, jottei purkaessa tule outoja konflikteja mahdollisten samannimisten tiedostojen vuoksi. Poisto-komento on `rm index.html`. Purkukomento puolestaan on `unzip wordpress-4.4.2-fi.zip`. Nyt hakemistossa ”/var/www/lahinpizza” pitäisi olla kansio ”wordpress-4.4.2-fi”. Vaihdan kansion nimeksi ”wordpress”.

Poistetaan zip, sitä ei enää tarvita, etenkin kun sitä ei haluta sisällyttää versionhallintaan, komennolla `rm wordpress-4.4.2-fi.zip`. Aion laittaa koko lahinpizza-kansion sisällön Gitiin talteen.

Periaatteessa WordPressin päätiedostoista suurin osa on turha laittaa Gitiin viemään tilaa, koska ne voidaan aina ladata uudelleen WordPressin omilta sivuilta. Tämän sivuston kanssa tavoitteeni

on kuitenkin tehdä Git-hakemistosta sellainen, että pystyn ottamaan sivuston käyttöön Gitistä mahdollisimman nopeasti ja pienellä vaivalla. Jatkokehitettävästä skriptistä tulisi aika monimutkainen, jos sen pitäisi ensin pystyä lataamaan WordPress, purkamaan se, lataamaan GitHubista brach sekä tekemään myös muut muutokset, joita automaattideploy vaatii.

3.2 Gitin alustaminen devel-palvelimella

GitHub suosittelee käyttämään HTTPS-yhteyttä SSH-yhteyden tilalta, koska silloin jokaisen komennon kanssa komennon kirjoittajan täytyy tietää oikea käyttäjätunnus ja salasana. Oma kokemukseni kuitenkin on että SSH-yhteys on käyttäjäystävällisempi, koska käyttäjätunnusta ja salasanaa ei nimenomaan tarvitse jatkuvasti kirjoittaa.

Luodaan gitille omat SSH avaimet (ohjeet löytyvät osoitteesta https://help.github.com/articles/generating-a-new-ssh-key/). Gitin suositteleman 4096-bitin SSH-avaimen luominen voi kestää kauan, varsinkin Raspberry Pi:llä.

Hakemistossa /var/www/lahinpizza ajetaan komento `git init`. Lisätään remote (tarkemmat ohjeet löytyvät osoitteesta https://help.github.com/articles/adding-a-remote/). Komennot ovat `git remote add {remoten omakeksimä nimi} {remoten git osoite}`, esimerkiksi `git remote add origin git@github.com:TVaakanainen/lahinpizza.git`.

Komento `git status` kertoo että olen nyt branchilla master ja minulla on tarkkailematon hakemisto wordpress. En tällä hetkellä halua tehdä master branchille mitään, vaan haluan tehdä itselleni devel-branchin, joka käytännössä kuvaa kyseistä palvelinta, jolla tällä hetkellä työskentelen. Komento `git checkout –b devel` luo minulle uuden branchin ja vaihtaa sen aktiiviseksi.

Haluan tehdä ensimmäisen tallennuksen, jossa laitan vain puretun WordPressin git-versionhallintaan. Komento `git add wordpress/` hoitaa kyseisen operaation. Komento `git status` listaa kaikki muutokset/lisäykset, jotka kirjataan versionhalllintaan seuraavassa commitissa. Lista on varsin pitkä.

Komento `git commit`, tekee ensimmäisen kirjoituksen -- paitsi ettei tee, koska en ole alustanut gitin globaaleja muuttujia user.email ja user.name. Ajan kehotuksesta komennot `git config –global user.email ”{sähköpostiosoite}” ja git config –global user.name ”{Nimi}”`. Alustuksen jälkeen Git antaa commit-komennon suorittaa tehtävänsä.

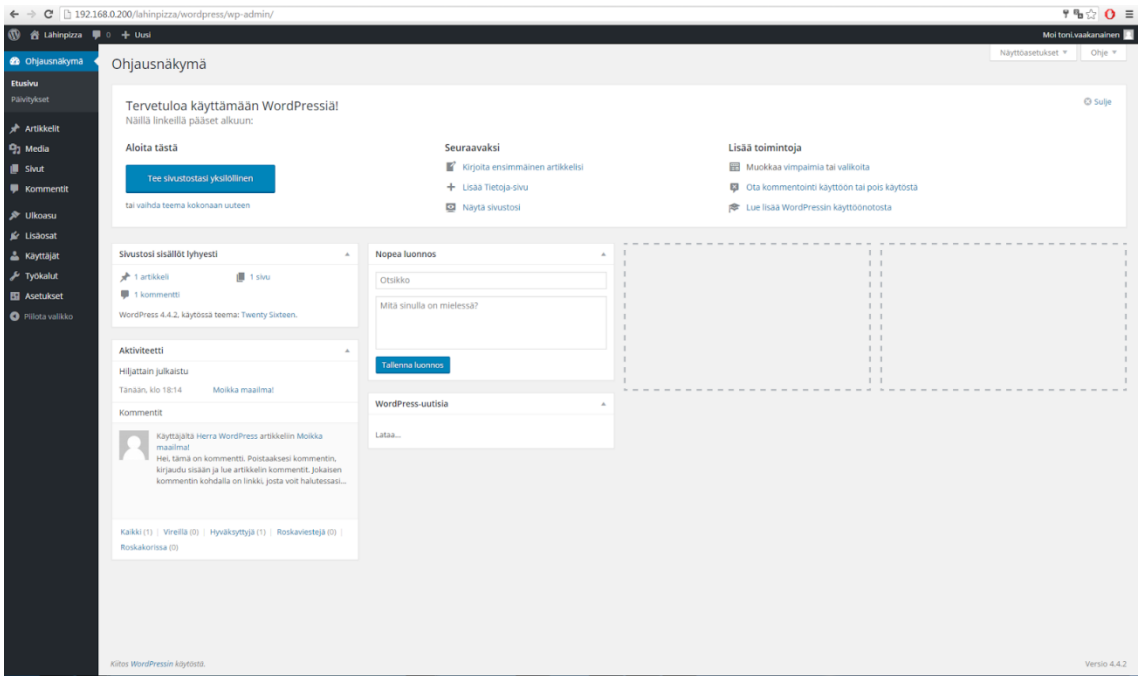
Jokaiselle commitille kannattaa kirjoittaa lyhyt kuvaava viesti. Viestin tallennuksen jälkeen Git tulostaa kaikki tapahtuneet muutokset ja lista on jälleen varsin pitkä. Ajamalla jälleen komennon `git status`, Git kertoo minulle ettei kyseisellä branchilla ole tallentamattomia muutoksia.

3.3 WordPress-tietokanta devel-palvelimelle

Seuraavaksi täytyy luoda WordPressille tietokanta ja käyttäjä kyseiseen kantaan. Turvallisinta on, että jokaisen tietokannan käyttäjä on eri. Kirjaudun MySQL root-käyttäjänä sisään ja ajan seuraavat komennot: `create database lahinpizza; grant all privileges on lahinpizza.* to "lahinpizza"@"localhost" identified by ”{salasana}”; flush privileges; exit;`. Testaan käyttäjän kirjautumalla sillä sisään. Tietokanta näyttää käyttävän peruskirjain-kirjastonaan latin1, nähdäkseni tarvittavat suomalaiset kirjaimet ovat tuettu, joten en vaihda kirjaimistoa.

Seuraava vaihe on ajaa WordPressin oma asennusskripti. Asennusskriptin osoite on: http://192.168.0.200/lahinpizza/wordpress/wp-admin/setup-config.php. Ennen kuin asennusohjelman voi suorittaa, WordPress-kansion ja kaiken sen sisällön omistusoikeuksia täytyy muuttaa. Komento `sudo chown -R www-data:www-data wordpress/` vaihtaa omistusoikeudet kaikkii tiedostoihin www-data-käyttäjälle, joka on Apachen oletuskäyttäjä. Komennon jälkeen voin suorittaa asennusohjelman.

Asennuksen jälkeen sivusto toimii, ja pystyn kirjautumaan WordPressiin admin osioon. Ohessa kuvakaappaus toimivasta WordPress-sivuston admin-osiosta.



Kuva 7. WordPress-sivuston admin-näkymä

3.4 WordPress-tietokannan vienti GitHubiin

Asennuksen jälkeen on hyvä hetki tehdä seuraava Git commit. Vaihtuneet tiedosto ovat wordpress/.htaccess ja wordpress/wp-config.php. Kannattaa myös tarkistaa manuaalisesti, että wp-config.php-tiedoston sisältö on toivottu. Tiedostoa lukiessa selvisi, että asennusskripti vaihtoi tietokannan merkistön, sekä määritteli salt-merkistöt. Salt-merkistöjen puuttuminen voi olla erittäin suuri tietoturvahaavoittuvuus, ja ne kannattaa käydä tarkistamassa.

Samalla on hyvä hetki ottaa myös ensimmäinen tietokantavarmuuskopio, komennolla `mysqldump -u lahinpizza -p{salasana} lahinpizza > lahinpizza.sql`. Ennen komentoa kannattaa tarkistaa että sen suorittaa oikeassa kansiossa, minun tapauksessani haluan dump-tiedoston olevan hakemistossa /var/www/lahinpizza. Lisätään tietokannan varmuuskopio gitiin ja tehdä commit, komennoilla `git add lahinpizza.sql` ja `git commit`.

Toimintojen jälkeen on hyvä tehdä ensimmäinen pusku remoteen, komennolla `git push origin devel`. Komennon jälkeen huomasin, että branchieni suunnat ovat mahdollisesti väärinpäin. Lokaali devel-branchini on mahdollisesti hierarkiassa korkeammalla kuin GitHubissa

oleva. Uudelleennimeän kyseisen lokaalin haaran ja tuon uuden kopion remotesta, jotta Git-hierarkia on varmuudella oikein päin. Komennot ja komentojen vastaukset alla.

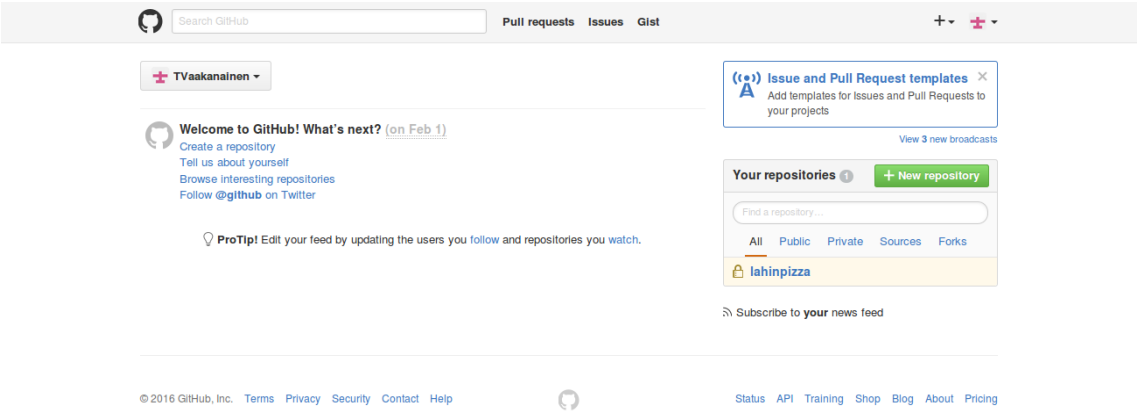
```
pi@devel-raspi /var/www/lahinpizza $ git status
# On branch devel
nothing to commit (working directory clean)
pi@devel-raspi /var/www/lahinpizza $ git branch -m devel-creation
pi@devel-raspi /var/www/lahinpizza $ git status
# On branch devel-creation
nothing to commit (working directory clean)
pi@devel-raspi /var/www/lahinpizza $ git fetch origin
pi@devel-raspi /var/www/lahinpizza $ git checkout -b devel origin/devel
Branch devel set up to track remote branch devel from origin.
Switched to a new branch 'devel'
```

3.5 Staging-palvelimen tuonti ajantasalle

Staging-palvelimelle täytyy luoda SSH-avaimet (katso luku 3.2), jonka jälkeen etärepositorystä täytyy tehdä kopio palvelimelle. Yksinkertaisin tapa luoda lokaali kopio Git-repositorystä on ajaa komento “git clone {repositoryn osoite}”. Komento luo kopion palvelimella olevasta Git-hakemistosta kansioon jossa komento ajetaan, eli minulle oikea paikka ajaa komento on /var/www. Komento on minun tapauksessani `git clone git@github.com:TVaakanainen/lahinpizza.git`. Komennon jälkeen var/www-hakemistoon on luotu uusi kansio lahinpizza, jonka sisällä master-brach on oletuksena aktiivinen.

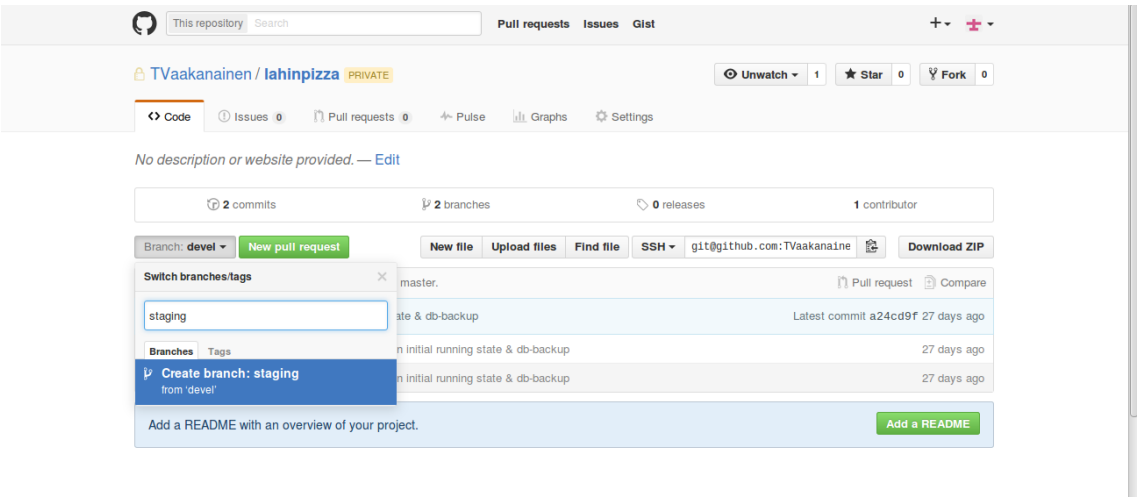
Staging-palvelin tuodaan ajantasalle hakemalla oikea branch GitHubista. Git näyttää komennolla `git branch` ainoastaan lokaalit branchit, ja komennon ajaessa näkee, että lokaalissa on ainoastaan master-branch, joka tulee oletuksena Git kopion mukana. Lokaaliin halutaan kuitenkin staging-branch, komento `git branch -a` näyttää myös remoten branchit. Ajamalla komennon näkee, ettei kyseistä branchia ole vielä olemassa, vaan remotessa “origin” on branchit “master” ja “devel”.

Yksi tapa luoda staging-branch on luoda se GitHubin puolella graafisesta käyttöliittymästä, jolloin kyseisen branchin pitäisi seurata GitHubissa olevaa devel-branchia. Ohessa ruudunkaappauksia GitHubin web-käyttöliittymän käytöstä.



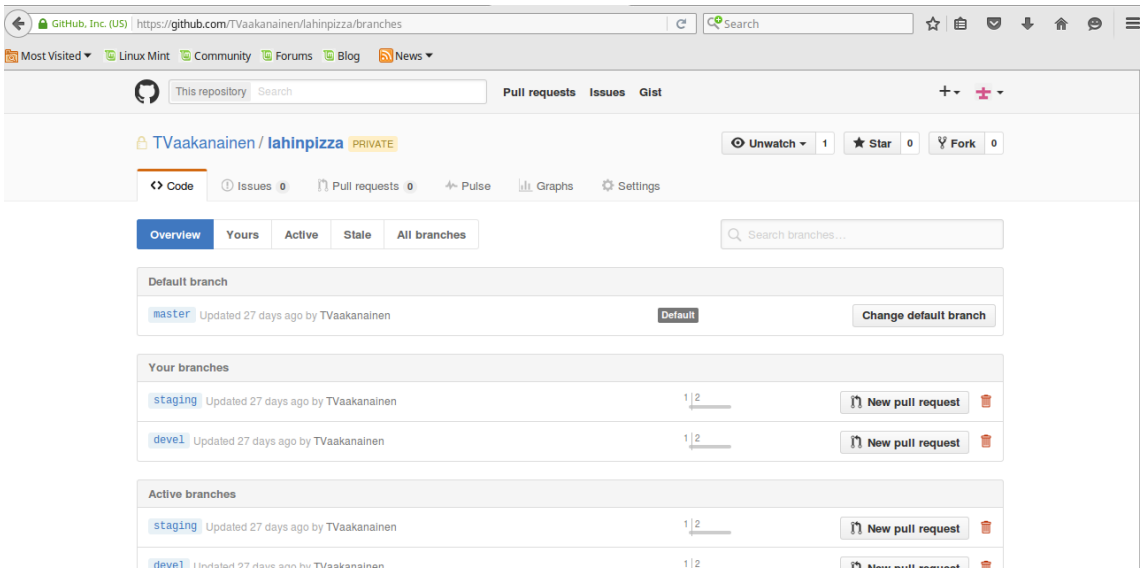
Kuva 8. GitHub oletusnäkymä kirjautumisen jälkeen

Ennen seuraavaa kuvaa olen painanut ruudunkaappauksessa oikealla näkyvää lahinpizza linkkiä, sekä vaihtanut tarkkailemani branchin masterista develiin.



Kuva 9. GitHub branchin luonti

Ohessa kuva GitHubin hallintanäkymästä, jossa näkyvät kaikki repositoryn branchit.

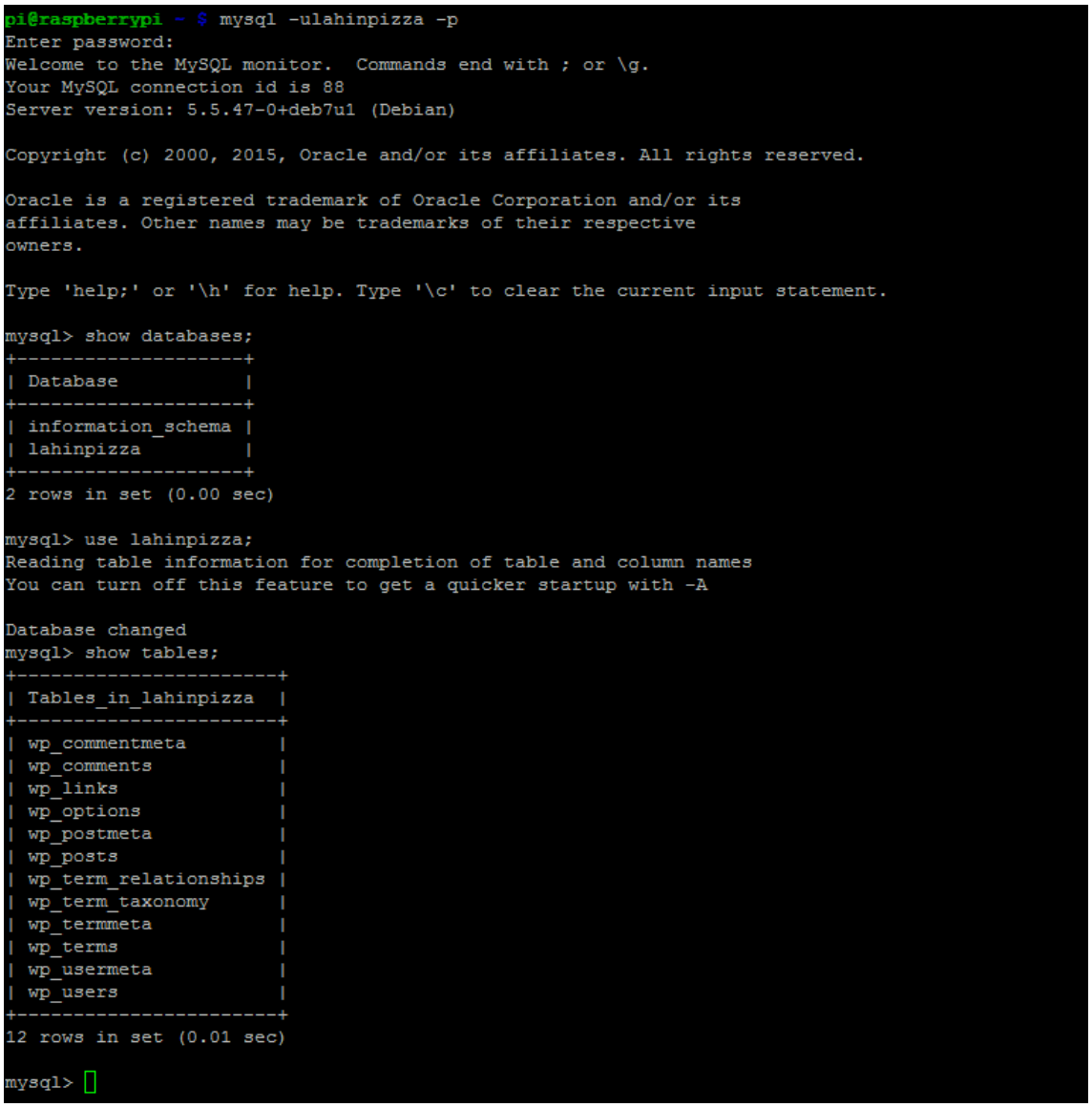


Kuva 10. GitHub branches-näkymä

Kun ajaa komennon `git branch -a` staging-palvelimella mikään ei näytä muuttuneen. Staging-palvelimen Gitiin ei ole päivitetty tietoja GitHubin repositorystä. Komento `git fetch origin` korjaa tilanteen. Nyt komento `git branch -a` näyttää myös staging-branchin. Ohessa seuraavat komennot ja komentojen vastaukset.

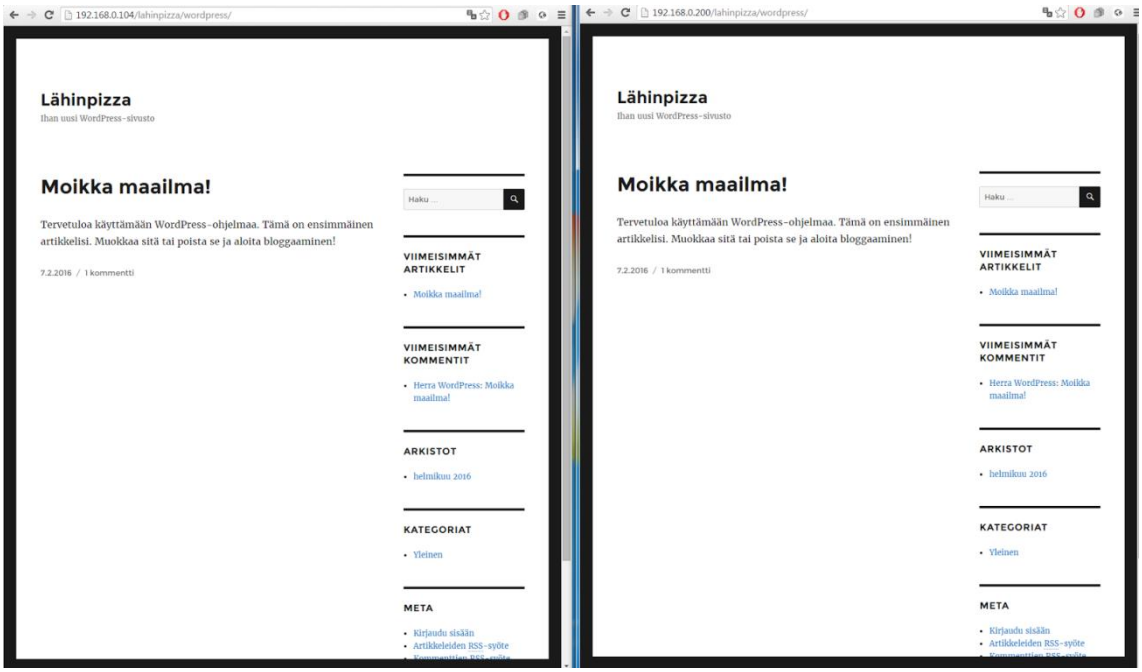
```
git checkout -b staging origin/staging
pi@raspberrypi ~/lahinpizza $ git status
# On branch staging
nothing to commit (working directory clean)
```

Seuraavaksi WordPressille täytyy luoda jälleen tietokanta, katso devel-ohjeet. Tämän jälkeen tuodaan tietokantakopio lahinpizza.sql, joka on ladattu gitistä. Hakemistossa /home/pi/lahinpizza ajetaan komento `mysql -ulahinpizza -p lahinpizza < lahinpizza.sql`. Tarkastetaan että tietokanta ainakin näyttää importoituneen oikein. `mysql -ulahinpizza -p; use lahin pizza; show tables;`. Ohessa kuva tietokannan tilasta.



Kuva 11. MySQL show tables

Nyt sekä staging että devel ovat samassa tilassa. Ohessa ruudunkaappaus devel- ja staging-palvelimien WordPress-sivuista.



Kuva 12. Devel- ja staging-palvelimet samassa tilassa

Kuvaillun projektin tila opinnäytetyön lopussa on se, että kehityspalvelimet devel ja staging sisältävät kaikki tarvittavat ohjelmistot web-kehitystyön jatkamiseksi. Molemmissa ympäristöissä on puhdas WordPress-asennus, joka on versiohallinnoitu Gitillä.

Kehitystyötä jatketaan nykyisestä tilanteesta siten, että kehitystä viedään eteenpäin Devel-palvelimella, jolta muutokset viedään GitHubiin. GitHubiin täytyy muistaa viedä tiedostomuutosten mukana myös tietokantamuutokset.

GitHubissa devel-haara sisältää aina devel-palvelimen koodin. Mikäli projektissa on useampi kehittäjä, devel-haaraan viedään yleensä vain kehittäjän mielestä valmiita kokonaisuuksia.

Staging-palvelin tuodaan aina aika ajoin samaan tilaan kuin devel-palvelin, normaalissa prosessissa silloin kun seuraavaksi julkaistavan päivityksen kehitys on Devel-haarassa saatu loppuun. Staging-palvelimella on aina GitHubissa olevan staging-haaran koodi. Kun staging-palvelinta päivitetään staging-haaraan täytyy yhdistää devel-haaran koodi. Tämän yhdistämistoiminnon voi tehdä GitHubin visuaalisessa käyttöliittymässä. Useamman kehittäjän projektissa GitHubin kautta tehty *pull request* antaa muille kehittäjille mahdollisuuden tarkastaa muutokset sekä kommentoida niitä ennen yhdistämistä.

Kehitystyön edetessä pitää luoda myös production-ympäristö. Production-ympäristössä Apachen konfigurointi pitää viedä loppuun luomalla virtuaalipalvelin. Virtuaalipalvelin mahdollistaa asiakkaalle näkyvien url-polkujen muokkaamisen sekä nimipalvelimen (DNS) ohjaamien kutsujen ohjaamisen oikeaan paikkaan production-palvelimella.

4 POHDINTA

Tarkoituksenani oli kuvata minkälaisia valmisteluja on mielestäni hyvä tehdä ennen web-kehitysprojektin aloittamista. Kuvaamani valmistelut eivät kuitenkaan ole täysin kattavia ja jatkokehitettävää jää.

Apachen konfigurointi ei ole opinnäytetyössäni loppuun asti viety. Viimeistään production-palvelimella Apacheen täytyy luoda virtuaaliserveri. Virtuaaliserverillä sivuston osoitteen saa yksinkertaisemmaksi, mikä helpottaa etenkin tehokäyttäjien sivuston käyttöä. Ammattimaisesti tehty osoiterakenne antaa sivustosta ammattimaisemman kuvan ihmisille, jotka asiaan kiinnittävät huomiota.

Toinen parannus olisi palvella sivustoa HTTPS-yhteyden kautta, WordPress-sivustot käyttävät oletuksena HTTP-yhteyttä. Tämän muutoksen tekeminen voi olla hieman työläämpi, koska se täytyy tehdä juuri oikein tai muuten sivusto voi vaikuttaa epäluotettavalta käyttäjille. Mikäli sivusto käyttää HTTPS-yhteyttä, mutta jotain palvellaan HTTP-yhteyden kautta, selaimet varoittavat käyttäjää. Kunnollisen HTTPS-yhteyden käyttöönotto myös maksaa, koska SSL-sertifikaatti täytyy ostaa joltain luotetulta palveluntarjoalta. SSL-sertifikaatin voi luoda myös itse, mutta myös silloin selain varoittaa käyttäjää siitä, ettei sertifikaatti ole tarkistettu.

Mikäli sertifikaattiin ei halua investoida rahaa, mutta haluaa tehdä sivustosta hieman tietoturvallisemman, yksi vaihtoehto on tehdä oma sertifikaatti, jota käyttää ainoastaan WordPressin admin-puolella. Silloin kaikki admin-puolen liikenne kulkee salattuna, ja varoituksen näkee ainoastaan adminit, jotka tietävät mistä varoitus johtuu.

Kolmas parannus olisi automatisoida kaikki interaktio Gitin ja palvelimien välillä. Tämä vaihe vaatii skriptin kirjoittamisen, joka tekee kaikki tarvittavat toiminnot, kuten hakee muutokset GitHubista palvelimelle, suorittaa merge-komennon sekä päivittää tietokannan mergen jälkeen. Kyseinen skripti tekisi palvelimien päivittämisestä erittäin yksinkertaista. Palvelimella täytyisi suorittaa ainoastaan yksi komento, joka hoitaa kaiken muun.

Neljäs vaihe olisi kirjoittaa automaatiotestejä. Yksinkertaisen sivuston kanssa riittäisi todennäköisesti se, että palvelimet ottaisivat vuorollaan yhteyden toisiinsa ja tarkistaisivat että sivut latautuvat. Mahdollisesta käyttökatkoksesta palvelin lähettäisi ylläpitäjälle sähköpostia.

Edelliseen toiminnallisuuteen löytyy paljon hyviä toimijoita, jotka palvelua myyvät, esimerkiksi Pingdom. Kyseiset palvelut maksavat, mutta mikäli oma aika tai osaaminen ei vastaavan toiminnallisuuden tekemiseen riitä, palvelut ovat rahansa arvoisia. Käyttökatkokset voivat johtaa loppukäyttäjien menettämiseen. Käyttökatkosten tunnistamisen lisäksi kyseiset palvelut antavat myös paljon metriikkaa, jonka kerääminen itse voi olla erittäin haastavaa. Esimerkkinä näistä metriikoista vastausaikoja, jopa eri puolilta maailmaa. Kyseinen metriikka voi olla erittäin arvokasta mikäli kohderyhmä ei ole ainoastaan kotimaista.

LÄHTEET

Baron Schwartz, 2009. When are you required to have a commercial MySQL license?. Hakupäivä 1.6.2016, http://www.xaprb.com/blog/2009/02/17/when-are-you-required-to-have-a-commercial-mysql-license/

Daniel J. McGlinn, (ei julkaisuvuotta). What is Git. Hakupäivä 16.5.2016, http://mcglinn.web.unc.edu/blog/what-is-git/

Git man page, 16.3.2016

GitHub, (ei julkaisuvuotta). Which remote URL should I use?. Hakupäivä 1.6.2016, https://help.github.com/articles/which-remote-url-should-i-use/

Kakul Srivastava, 2016. Introducing unlimited private repositories. Hakupäivä 1.6.2016, https://github.com/blog/2164-introducing-unlimited-private-repositories

Linux Foundation, 2009a. What Is Linux: An Overview of the Linux Operating System. Hakupäivä 17.10.2015, https://www.linux.com/learn/new-user-guides/376-linux-is-everywhere-an-overview-of-the-linux-operating-system

Linux Foundation, 2009b. What Is Linux: An Overview of the Linux Operating System – The Birth of Linux. Hakupäivä 17.10.2015, https://www.linux.com/learn/new-user-guides/376?start=2

Jennifer Cloer, 2015. 10 Years of Git: An Interview with Git Creator Linus Torvalds. Hakupäivä 16.3.2016, http://www.linux.com/news/featured-blogs/185-jennifer-cloer/821541-10-years-of-git-an-interview-with-git-creator-linus-torvalds

Oracle, (ei julkaisuvuotta). Oracle and Sun Microsystems. Hakupäivä 21.3.2016. https://www.oracle.com/sun/index.html

Rahabu Ransagar, 2009. MySQL Primer. Hakupäivä 21.3.2016.
https://tecnoesis.wordpress.com/tag/mysql-origin/

Software in the Public Intreset, Inc, 2015. About Debian. Hakupäivä 17.10.2015.
https://www.debian.org/intro/about

Solid IT, 2016. DB-Engines Ranking. Hakupäivä 2.6.2016. http://db-engines.com/en/ranking

TechTerms, 2011. Web Server. Hakupäivä 30.1.2016. http://techterms.com/definition/web_server

TechTerms, 2014. Server. Hakupäivä 17.10.2015, http://techterms.com/definition/server

Welcome to Raspbian. Hakupäivä 17.10.2015. https://www.raspbian.org

W3Techs, 2015a. Usage statistics and market share for Linux for websites. Hakupäivä
17.10.2015, http://w3techs.com/technologies/details/os-linux/all/all

W3Techs, 2015b. Usage statistics and market share for Linux for websites. Hakupäivä
17.10.2015, http://w3techs.com/technologies/details/os-unix/all/all

W3Techs, 2016. Usage of web servers for websites. Hakupäivä 30.1.2016,
http://w3techs.com/technologies/overview/web_server/all