Windows PowerShell service deskin tukena

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
syksy, 2021

Juuso Kilponen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Hämeenlinnan korkeakoulukeskus

Tekijä Juuso Kilponen Vuosi 2021

Työn nimi Windows PowerShell service deskin tukena

Ohjaajat Lasse Seppänen

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli perehtyä Microsoftin PowerShell-työkaluun ja selvittää, millainen rooli PowerShellilla on tulevaisuudessa palvelin- ja työasemaympäristöjen ylläpidossa. Työn käytännön osassa selvitettiin, miten PowerShellilla voi suorittaa service deskissä esiintyviä työtehtäviä, ja tehostaa näin työtehtävien suorittamista. Opinnäytetyössä tutustutaan PowerShellin komentoihin ja perusperiaatteisiin, joiden avulla lukija saa valmiudet PowerShellin käytön aloittamiseen.

Teoriaosuudessa perehdytään PowerShellin keskeisiin ominaisuuksiin ja toimintoihin, mitkä käyttäjän on hyvä tietää PowerShellista. Työssä esitellään myös PowerShellin versiohistoriaa ja pohditaan, millainen tulevaisuus PowerShellilla on. Teoriaosuudessa esitellään myös keskeisiä PowerShell-komentoja, joita hyödynnetään käytännön osuudessa. Opinnäytetyö on toiminnallinen. Työtä varten tehtiin teemahaastattelu kohdeyrityksessä.

Opinnäytetyön tuloksena syntyi opas, jossa perehdytään PowerShellin perustoimintoihin ja käytön aloittamiseen. Kerätyn aineiston perusteella PowerShellilla on hyvät edellytykset olla tärkeä palvelinten- ja työasemaympäristöjen hallintatyökalu tulevaisuudessa. Käytännön osassa todettiin, että PowerShellin avulla pystyy suorittamaan tehokkaasti käyttötuessa esiintyviä työtehtäviä, kun täytyy hallita tai muokata suurempia tietokokonaisuuksia. PowerShellin etäyhteystoimintojen sekä valmiiksi luotujen skriptien avulla pystyy myös tehostamaan työtehtävien suorittamista.

Avainsanat PowerShell, service desk, skripti, komento

Sivut 62 sivua ja liitteitä 1 sivu

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Abstract

Author Juuso Kilponen

Year 2021

Subject Enhancing service desk tasks with Windows PowerShell

Supervisors Lasse Seppänen

ABSTRACT

The purpose of the thesis was to get acquainted with Microsoft's PowerShell and investigate what kind of role PowerShell will have in the future in the administration of servers and workstations. In the practical part of the thesis, the purpose was to examine how service desk tasks can be performed with PowerShell and in this way increase the performance of working. The thesis gives basic understanding of PowerShell's principles and commands for the reader to start using PowerShell.

The theoretical part of the thesis introduces the main features and functions of PowerShell. The thesis includes summary of PowerShell's version history and considers what kind of future PowerShell will have. The theoretical part also contains summary of essential PowerShell-commands which are used in the practical part of the thesis. The thesis is practical. A thematic interview was conducted with the target company.

As a result of the research, PowerShell guide was made. In this guide, focus is on PowerShell's basic functions and how users can start working with PowerShell. The research demonstrates that PowerShell has good potential to be an important tool at managing server and workstation environments in the future. The practical part stated that with PowerShell service desk tasks can be enhanced when there is plenty of information that needs to be handled or edited. Tasks can also be enhanced with PowerShell remote commands and scripts.

Keywords PowerShell, service desk, script, command

Pages 62 pages and appendices 1 page

Sanasto

Aktiivihakemisto	Active Directory (AD), Windows-toimialueen käyttäjä- ja hakemistotietokanta
Attribuutti	Jonkin asian tai kohteen ominaisuus
Azure	Microsoftin pilvipalvelu
Cmdlet	PowerShelliin valmiiksi luotu komento, joka tekee ennalta määrätyn toimenpiteen
Muuttuja	Käytetään tiedon säilytykseen komentojen tai skriptien suorittamisen aikana
.NET-objekti	.NET-ympäristössä toimiva olio, joka sisältää erilaisia arvoja ja näihin liittyviä komentoja
Objekti	Kokonaisuus, joka sisältää joukon tietoa
Parametri	PowerShell-komentoihin lisättävä määrite, jolla komennosta saa tehtyä monipuolisemman ja tarkemman
Putkitus	PowerShell-komentojen liittäminen yhdeksi kokonaisuudeksi
SaaS	Palveluna hankittava ohjelmisto
Skripti	Valmiiksi luotu komento tai komentosarja
Windows PowerShell ISE	PowerShell-skriptien luomiseen ja testaamiseen kehitetty sovellus

Sisällys

Kuvat, ohjelmakoodit ja taulukot

Liitteet

1 Johdanto

Työasema- ja palvelinympäristöjen laajentuessa sekä monimutkaistuessa ympäristöjen hallintaa pyritään helpottamaan ja tehostamaan eri tavoilla. Microsoft on kehittänyt pitkään PowerShell-nimistä komentokehotetyökalua, jonka avulla työasema- ja palvelinympäristöjen ylläpitämistä on pyritty helpottamaan. Tässä opinnäytetyössä perehdyn PowerShell-työkaluun ja selvitän käytännön esimerkein, miten PowerShellia voi hyödyntää service desk -tehtävissä. Keskityn sekä teoria- että käytännön osassa Windows PowerShell 5.1 -versioon. Teoriaosassa kerron kuitenkin myös eri PowerShell-versioista ja pohdin, millainen tulevaisuus PowerShellilla on.

Tutustun opinnäytetyössä myös service desk -työhön. Koska service desk on usein loppukäyttäjien ensimmäinen kontaktikanava erilaisissa tietoteknisissä ongelmissa, tulee service desk -työssä vastaan monenlaisia työtehtäviä. Nykyaikana yksi työelämän trendeistä on työtehtävien tehostaminen. Tämä ilmenee myös service desk -työssä. Tästä syystä haluan selvittää, miten PowerShellilla pystyy tehostamaan ja suorittamaan service deskissä esiintyviä työtehtäviä.

Olen tutustunut PowerShelliin työni kautta, mutta en ole päässyt perehtymään syvällisemmin PowerShellin tarjoamiin mahdollisuuksiin eri palveluiden ja järjestelmien hallinnassa. PowerShellin avulla pystyy esimerkiksi automatisoimaan toistuvia työtehtäviä skriptien avulla. Opinnäytetyö on minulle hyvä tilaisuus kartuttaa PowerShell-osaamistani ja päästä syvemmälle PowerShellin maailmaan.

Opinnäytetyön tutkimuskysymykset ovat:

- Millainen rooli PowerShellilla on tulevaisuudessa työasema- ja palvelinympäristöjen ylläpidossa?

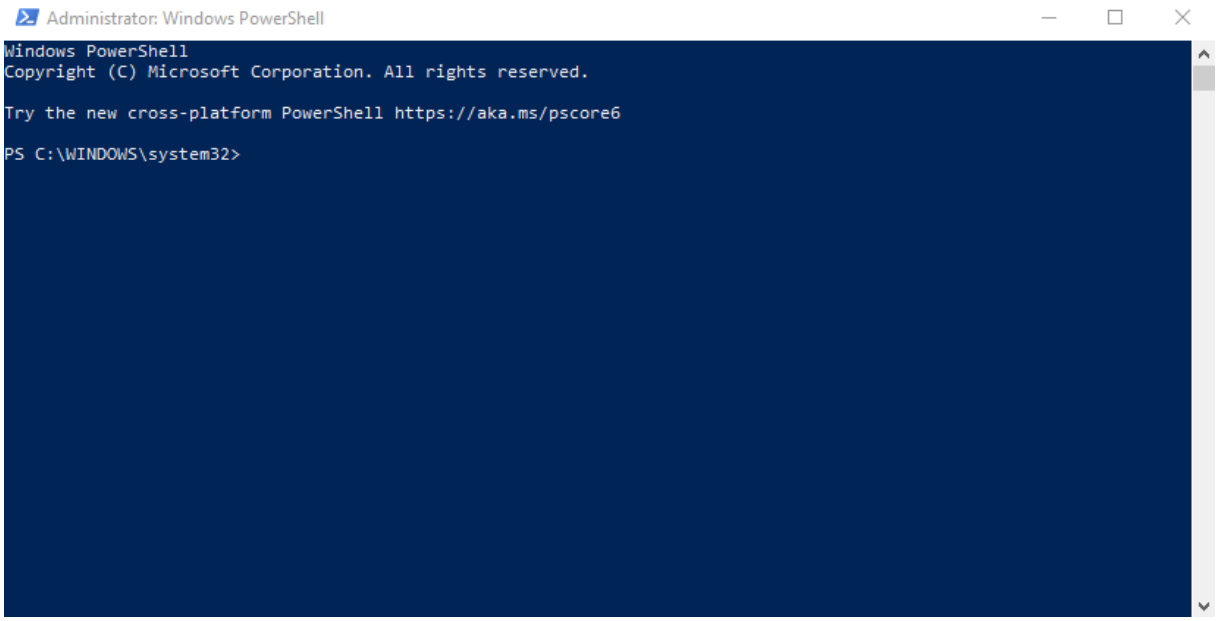
- Miten PowerShellilla voidaan tehostaa service deskin työtehtäviä?

2 PowerShell

PowerShell on Microsoftin kehittämä järjestelmäriippumaton komentokehote- ja automaatiotyökalu, joka on rakennettu .NET Common Language Runtime (CLR) -ympäristön päälle (Kuva 1). PowerShellin luojan Jeffrey Snoverin tarkoituksena oli luoda PowerShellista .NET-ympäristöä hyödyntävä uuden sukupolven alusta, jolla sovelluskehittäjät, testaajat, tehokäyttäjät, järjestelmänvalvojat sekä graafisen käyttöliittymän käyttäjät pystyisivät työskentelemään tehokkaasti ja käyttämään hyväksi .NET-objekteja. PowerShellista julkaistiin ensimmäinen testiversio vuonna 2005 työnimellä Monad. Noin vuosi testiversion julkaisemisen jälkeen Monad nimettiin uudelleen Windows PowerShelliksi, ja marraskuussa 2006 julkaistiin ensimmäinen versio Windows PowerShell 1.0. (Peters ym., 2018, ss. 8–9)

PowerShell hyödyntää .NET-objekteja syötteenä sekä tulostuksessa. PowerShellin tehokkuus perustuukin osaltaan .NET-objektien käsittelyyn, näiden sisältämään informaatiomäärään sekä kyseisen informaatiomäärän tehokkaaseen käsittelyyn. Toisin kuin perinteisissä komentokehotteissa, joissa käsitellään oletuksena merkkijonoja, PowerShellissa käsiteltävät objektit sisältävät paljon informaatiota, jota pystyy käsittelemään eri tavoilla PowerShell-komennoilla. PowerShell pääsee käsiksi myös tietokoneen levyjärjestelmään sekä muihin tietolähteisiin, kuten Windowsin rekisteriin ja sertifikaatteihin. Lisäksi PowerShelliin on saatavilla eri palveluihin sidoksissa olevia moduuleja. Moduulien mukana PowerShelliin saa käyttöön uusia toimintoja sekä komentoja, joilla pystyy hallitsemaan kyseisiä palveluja. (Microsoft, 2021d) PowerShellilla pystyy hallitsemaan paikallista tietokonetta, mutta myös etäyhteyden kautta muita tietokoneita tai palvelimia. Yhdellä komennolla voi kohdistaa saman toiminnon usealle eri tietokoneelle etäyhteyden välityksellä. Valmiiksi tehtyjä skriptejä voi myös kohdistaa joukolle eri tietokoneita. Nämä etäyhteysominaisuudet antavat PowerShellin käyttäjälle tehokkaan tavan hallita suurta määrää eri tietokoneita. (Microsoft, 2020e)

Kuva 1 Windows PowerShell 5.1



2.1 PowerShell-versiot

Microsoft on kehittänyt PowerShell-työkalua Windows XP -käyttöjärjestelmäajoilta lähtien. Ensimmäinen PowerShell-versio julkaistiin vuonna 2006 (PowerShell 1.0) Windows XP SP2 -käyttöjärjestelmäversiolle sekä Windows Server 2003 SP1 -palvelinversiolle. Tämän jälkeen Microsoft on jatkanut PowerShellin kehittämistä ja julkaissut tasaisin väliajoin uusia PowerShell-versioita (Taulukko 1). Vuonna 2016 julkaistu PowerShell 5.1 oli viimeinen vain Windows-ympäristöön kehitetty PowerShell-versio. Tämän takia PowerShell versioista 1.0–5.1 käytetään myös nimitystä Windows PowerShell. Ensimmäinen monialustalle kehitetty PowerShell-versio 6.0 julkaistiin vuonna 2018. Kyseistä versiota kutsutaan myös nimellä PowerShell Core. Monialustamalliin siirtymisen myötä PowerShell 6.0 kehitettiin avoimen lähdekoodin .NET Core 2.0 -alustalle. Uusimmat PowerShell-versiot 7.0 ja 7.1 julkaistiin vuonna 2020. (Microsoft, 2020b)

Taulukko 1 PowerShell-versiot (Microsoft, 2020c)

Versio	**Julkaisupäivä**	**Huomautus**
PowerShell 7.1	11/2020	Luotu .NET Core 5.0:n päälle (nykyinen).
PowerShell 7.0	3/2020	Luotu .NET Core 3.1:n päälle (vakaa).
PowerShell 6.2	3/2019	
PowerShell 6.1	9/2018	Luotu .NET Core 2.1:n päälle.
PowerShell 6.0	1/2018	Ensimmäinen julkaisu, luotu .NET Core 2.0:n päälle. Asennettavissa Windows-, Linux- ja macOS-alustoille.
PowerShell 5.1	8/2016	Julkaistu Windows 10 Anniversary Update -päivityksen ja Windows Server 2016 -alustan mukana.
PowerShell 5.0	2/2016	Julkaistu Windows Management Framework (WMF) 5.0 -kirjaston mukana.

PowerShell 4.0	10/2013	Sisäänrakennettu Windows 8.1- ja Windows Server 2012 R2 -alustoihin. Asennettavissa Windows 7 SP1-, Windows Server 2008 R2 SP1- ja Windows Server 2012 -alustoille.
PowerShell 3.0	10/2012	Sisäänrakennettu Windows 8- ja Windows Server 2012 -alustoihin. Asennettavissa Windows 7 SP1-, Windows Server 2008 SP1- ja Windows Server 2008 R2 SP1 -alustoille.
PowerShell 2.0	7/2009	Sisäänrakennettu Windows 7- ja Windows Server 2008 R2 -alustoihin. Asennettavissa Windows XP SP3-, Windows Server 2003 SP2- ja Windows Vista SP1 -alustoille.
PowerShell 1.0	11/2006	Asennettavissa Windows XP SP2-, Windows Server 2003 SP1- ja Windows Vista -alustoille. Valinnainen komponentti Windows Server 2008 -alustalle.

2.1.1 Windows PowerShell

Windows PowerShellistä puhuttaessa tarkoitetaan PowerShell versioita 1.0–5.1. Kuten nimestäkin voi jo päätellä, Windows PowerShell -versiot on kehitetty vain Windows-käyttöjärjestelmille. Microsoft alkoi lisätä PowerShellia käyttöjärjestelmiinsä Windows 7 SP1- ja Windows Server 2008 R2 SP1 -versioista lähtien. Uusin vuonna 2016 julkaistu 5.1-versio tulee tällä hetkellä valmiiksi asennettuna kaikkiin Windows-käyttöjärjestelmiin. Microsoft on lopettanut Windows PowerShellin kehittämisen, mutta siihen julkaistaan edelleen tarpeellisia päivityksiä käyttöjärjestelmäpäivitysten mukana. (Microsoft, 2020b)

Windows Powershellin mukana on julkaistu PowerShell 2.0 -versiosta lähtien erillinen Windows PowerShell ISE (Integrated Scripting Environment) -sovellus. Windows PowerShell ISE -sovelluksessa käyttäjä voi kirjoittaa ja testata skriptejä sekä suorittaa PowerShell-komentoja. Ohjelman graafinen käyttöliittymä tarjoaa yhdessä ikkunassa PowerShell-komentokehotteen, skriptityökalun sekä kirjaston käytettävissä olevista komennoista. Näiden lisäksi sovellus tarjoaa joukon muita hyödyllisiä ominaisuuksia, joista on hyötyä PowerShellin käytössä ja skriptien kirjoittamisessa. Sovelluksen tarkoituksena onkin tarjota käyttäjille monipuolinen ja tehokas työkalu PowerShellin käyttöön sekä skriptien tekemiseen

ja testaukseen. Windows PowerShellin kehitystyön päättyessä myös Windows PowerShell ISE -sovelluksen kehitys on lopetettu. Sovellus toimitetaan kuitenkin valmiiksi asennettuna kaikkien Windows-käyttöjärjestelmien mukana, ja siihen julkaistaan edelleen tarpeellisia vakautta ja tietoturvaa parantavia päivityksiä Windows-päivitysten mukana. Microsoft on korvannut Windows PowerShell ISE -sovelluksen Visual Studio Code -sovelluksella, joka tukee uudempia PowerShell 6.0- ja 7.0-versioita. (Microsoft, 2018a)

2.1.2 PowerShell Core 6.0

Microsoft muutti linjaansa Windows PowerShell 5.1 -version jälkeen julkaisemalla seuraavan PowerShell-version avoimen lähdekoodin versiona. Samalla Microsoft siirtyi monialustamalliin tarjoamalla PowerShellin Windows-alustojen lisäksi myös muille alustoille. Muutoksen myötä myös käyttäjäyhteisö pääsi mukaan PowerShellin kehitystyöhön. Uusi PowerShell Core 6.0 -versio, joka pohjautuu .NET Core -ympäristön päälle julkaistiin tammikuussa 2018. (Peters ym., 2018, ss. 11–12) Monialustamalliin siirtymisen myötä PowerShell Core 6.0 tuli tarjolle Windows-, Linux- ja macOS-alustoille. Windows-alustoilla on mahdollista käyttää sekä Windows PowerShell -versiota että PowerShell Core -versiota saman aikaisesti. Tämä mahdollistaa esimerkiksi Windows PowerShellin ja Core-version vertailun samalla tietokoneella. Vaikka PowerShell Core 6.0 -versiossa siirryttiin uuden NET. Core -ympäristön päälle, pyrki Microsoft säilyttämään taaksepäin yhteensopivuuden Windows PowerShellissa käytettyjen komentojen ja moduulien kanssa. Osa vanhoista moduuleista oli kuitenkin vielä kääntämättä uuteen ympäristöön PowerShell Core 6.0 -version julkaisun hetkenä. Tilannetta parannettiin kuitenkin kääntämällä moduuleja yhteensopiviksi Microsoftin ja kehittäjäyhteisön kesken. (Microsoft, 2018b)

PowerShellin muuntaminen monialustamalliin voi kuulostaa aluksi erikoiselta ratkaisulta, sillä PowerShellin ensimmäisestä versiosta alkaen PowerShell on pidetty tiukasti Windows-alustalle suunnattuna komentokehotetyökaluna. 5.1-versioon ennättänyt Windows PowerShell oli hiottu hyvin toimivaksi ja kaiken kattavaksi Windows-alustojen ylläpitotyökaluksi. Uuden Core-version myötä monet Windows PowerShell -version toiminnot poistuivat käytöstä yhteensopivuusongelmien vuoksi. Monialustamalliin siirtymistä tukee kuitenkin ajatus siitä, että esimerkiksi Microsoftin omassa Azure-pilvipalvelussa käytetään runsaasti Linux-pohjaisia alustoja. Tarjoamalla PowerShellin

käyttöön myös Linux- ja macOS-alustoille, Microsoft tarjoaa universaalin hallinta- ja automaatiotyökalun kaikille pilvipalveluiden käyttäjille. Vaikka PowerShellin kehitys jatkuu Core-version aloittamalla tiellä, ei Windows PowerShellia ole kuitenkaan kuopattu. Windows PowerShell toimitetaan valmiiksi asennettuna edelleen jokaisen Windows 10- ja Windows Server -käyttöjärjestelmän mukana. Windows PowerShell tuskin saa enää uusia ominaisuus- tai versiopäivityksiä, mutta se ei silti ole huono asia, sillä Windows PowerShell 5.1 on jo itsessään hyvin monipuolinen ja loppuun asti hiottu kokonaisuus. (Jones, 2018)

PowerShell Core 6.0 -version jälkeen Microsoft julkaisi vielä parannellut versiot 6.1 sekä 6.2. Lukuisten suorituskykyparannusten sekä bugikorjausten lisäksi tärkeimpiä parannuksia olivat siirtyminen .NET Core 2.1 -ympäristöön sekä Windows Compatibility Pack for .NET Core -paketin lisääminen osaksi PowerShell Core 6.1 -versiota. Tällä yhteensopivuuspaketilla PowerShell Core -version komentojen määrä nousi yli 1900 kappaleeseen. (Microsoft, 2018c)

2.1.3 PowerShell Core 7.0

Maaliskuussa 2020 julkaistiin uusi versio PowerShell Coresta. PowerShell 7.0 -version mukana PowerShell sai jälleen uusia ominaisuuksia sekä taaksepäin yhteensopivuutta parannettiin Windows PowerShellista tuttujen moduulien kanssa. Myös .Net Core -ympäristö päivittyi versioon 3.1. (Microsoft, 2020f)

PowerShell 7.0 versioon on julkaistu uusin päivitys marraskuussa 2020. PowerShell 7.1 on tällä hetkellä uusin saatavilla oleva PowerShell-versio. PowerShell 7.1 käyttää hyväksi uutta .NET Core 5.0 -ympäristöä. 7.1-versiossa pääpaino oli keskittyä yhteisön esiin nostamien ongelmien korjaamiseen. (Aiello, 2020)

2.2 PowerShellin tulevaisuus

PowerShellin tulevaisuus on vahvasti sidoksissa monialustamalliin. Peters ym. (2018, s. 27) uskovat, että tämä tulee jatkumaan myös tulevaisuudessa. Avoimeen lähdekoodiin siirtymisen myötä myös PowerShellin kehitysyhteisö on otettu vahvasti mukaan PowerShellin kehitystyöhön. Vanhaan Windows PowerShell -versioon kehitettyjä moduuleja muokataan jatkuvasti yhteensopivaksi uuden PowerShell Core -version kanssa. Pilvipalvelut,

kuten Azure, tulevat olemaan myös yksi PowerShellin ydinalueista. PowerShell on tuettuna esimerkiksi Azureen integroidussa Azure Cloud Shellissa. (Peters ym., 2018, s. 27) Jeffrey Snover (Snover, 2018) mainitsee pitämässään luennossa, että PowerShellin uusi missio monialustamalliin siirtymisen myötä on tarjota tehokas työkalu digitaaliseen muutokseen. PowerShell halutaan tarjota jokaiselle alustalle minkä tahansa palvelimen, palvelun tai pilvipalvelun hallintaan. Perinteisiä paikallisia palvelinympäristöjä ei ole myöskään jätetty suunnitelmien ulkopuolelle.

PowerShell ei ole ainoa komentokehotetyökalu, jolla pystyy hallitsemaan pilviresursseja. Microsoftin Azure-pilvipalveluun on saatavilla esimerkiksi Azure CLI -komentokehotetyökalu, jolla pystyy hallitsemaan Azuressa olevia resursseja samaan tapaan kuin PowerShellilla. Käyttäjillä on siis mahdollisuus valita haluamansa komentokehotetyökalu eri vaihtoehtojen välillä. Se, mitä työkalua käyttäjät päätyvät käyttämään, on varmasti osittain kiinni siitä, mitä työkalua he ovat aiemmin tottuneet käyttämään. Azure CLI:n ja PowerShellin tapauksessa molemmilla työkaluilla pystyy hallitsemaan Azurea samalla tavalla. Vain komentojen syntaksi on erilaista. Joitakin eroja eri kielistä löytyy, mutta molemmat ovat varteenotettavia vaihtoehtoja Azuren hallintaan. Erilaisten työkalujen tarjoaminen Azuren käyttöön voidaan nähdä kilpailusta huolimatta hyvänä asiana; moninainen hallinnointiympäristö tarjoaa mahdollisimman laajalle joukolle mieleiset työkalut ympäristöjen ylläpitoon. (Miller, 2019)

Graafisia käyttöliittymiä hyödyntävien palvelimien lisäksi palvelinympäristöissä otetaan käyttöön myös erilaisia server core- ja nano server -palvelimia. Kyseiset palvelintyypit eivät sisällä ollenkaan perinteistä graafista käyttöliittymää vaan näiden hallinta tapahtuu kokonaan keskitettyjä hallintatyökaluja hyödyntämällä. Palvelimiin ei myöskään pysty ottamaan perinteistä etätyöpöytäyhteyttä. PowerShell on yksi keskeisistä hallintatyökaluista tämän tyyppisten palvelinten hallinnassa. Tämä lisää entisestään PowerShellin tärkeyttä palvelinten ylläpitotehtävissä. (Krause, 2016, s. 305)

Microsoft on kehittänyt monia hallintatyökaluja, jotka toimivat graafisessa käyttöliittymässä PowerShellin päällä. Käyttäjän tehdessä erilaisia toimenpiteitä graafisessa käyttöliittymässä komennot ohjataan PowerShell-komentoina eteenpäin, ja tehdyt muutokset tulostetaan näkyviin graafiseen käyttöliittymään. (Krause, 2016, s. 304) Vaikka Microsoft on luonut palvelujen hallintaan graafisen käyttöliittymän, osa toiminnoista on mahdollista tehdä vain

PowerShellin kautta. Näin ollen järjestelmän ylläpitäjän on käytettävä PowerShellia, mikäli tarvittava toiminto ei ole saatavilla graafisessa hallintapaneelissa. PowerShellia hyödyntämällä palvelujen ylläpito ja määrittäminen pysyvät myös helpommin yhdenmukaisina. Kun käytössä ovat samat komentosarjat, joita voi kopioida tai ajaa skriptillä, vältytään mahdollisilta inhimillisiltä virheiltä. PowerShellin vahvuutena on myös dokumentointi, joka tapahtuu automaattisesti. PowerShellista saa haettua kaikki tehdyt toimenpiteet yhdellä komennolla, ja näin ollen tehtyjen muutoksien ja komentojen tarkastelu jälkeenpäin on helppoa. (Minasi, 2014)

Kun PowerShell siirrettiin avoimeen lähdekoodiin, PowerShellin kehitysyhteisö pääsi mukaan PowerShellin kehitystyöhön. PowerShellin kehitystä on viety eteenpäin yhteisön kanssa GitHub-palvelussa. Yhteisö vastasi noin 50 prosentin osuudesta pull-pyynnöistä vuonna 2018. Tämän tilaston myötä voidaankin ajatella, että PowerShellin kehitystyö ja tulevaisuuden kehityssuunta ovat siirtyneet osittain kehittäjäyhteisön käsiin. (Snover, 2018) Tarkasteltaessa viimeisen vuoden aikana tehtyjä pull-pyyntöjen määrää, voidaan huomata yhteisön tehneen yli 50 prosenttia pyynnöistä. Myös GitHubiin nostetut ongelmapyynnöt ovat pääasiassa yhteisön luomia. (Microsoft, 2021c)

Digitaalisen muutoksen myötä pilvipalveluiden käyttö on kasvanut merkittävästi viime vuosina. Esimerkiksi monet ohjelmistot toimitetaan nykyisin SaaS-palveluna (Software as a Service). Tästä hyvänä esimerkkinä Microsoftin Office 365 -palvelu. (Stiennon, 2019) Tällä hetkellä kolme suurinta pilvipalveluiden toimittajaa ovat Amazon Web Services, Microsoft Azure ja Google Cloud Platform. Näiden kolmen suurimman toimijan osuus pilvipalvelujen kokonaismäärästä kasvaa jatkuvasti, kun yritykset siirtävät liiketoimintojaan pilveen. (Dignan, 2021) PowerShellilla pystyy hallitsemaan kaikkien kolmen suurimman edellä mainitun toimittajan pilvipalveluita. Pilvipalvelun hallintaa varten tarvitsee asentaa vain tarvittava moduuli PowerShelliin. Tämän jälkeen pilvessä olevien resurssien hallinta onnistuu normaaliin tapaan käyttäjän tietokoneessa olevan PowerShellin kautta. (AWS, n.d.; Google, n.d.; Microsoft, 2021b)

Digitaalisen muutoksen yksi mahdollistava tekijä on automaation lisääminen. Kasvavaan automaatiotarpeeseen tarvitaan kuitenkin tehokkaita työkaluja. PowerShell on yksi vaihtoehdoista, sillä se tarjoaa tehokkaan alustan ja työkalut automaation lisäämiseen. Tästä

syystä PowerShell on tärkeämpi osa järjestelmien ylläpitoa kuin koskaan aiemmin. (Snover, 2018) Automaatiosta saatavia hyötyjä ovat esimerkiksi inhimillisten virheiden väheneminen, prosessien tehostuminen ja resurssien säästäminen. Kun prosessi suoritetaan kokonaan automaationa, ei tähän tarvitse resursoida työntekijää, ja näin ollen yrityksen kustannukset laskevat ja tuottavuus paranee. (McHugh, n.d.)

2.3 PowerShellin suorituskäytäntö ja vaikutustaso

PowerShelliin on sisäänrakennettu toimintoja, joilla on pyritty estämään käyttäjän epähuomiossa suorittamat mahdollisesti haittaa aiheuttavat komennot tai skriptit. Suorituskäytännön (execution policy) tarkoituksena on estää käyttäjää suorittamasta mahdollisesti haitallisia skriptejä. Suorituskäytäntö ei rajoita PowerShell-komentokehotteessa suoritettavia komentoja, vaan estää valmiiksi luotujen skriptitiedostojen suorittamisen. Oletuksena execution policy on määritetty Windows 10 -käyttöjärjestelmissä tasoon restricted, joka estää skriptien suorittamisen. Windows palvelimilla asetukseksi on määritetty remote signed, joka vaatii ladattujen skriptien olevan allekirjoitettu luotetun julkaisijan toimesta. Execution policyn pääsee muuttamaan avaamalla PowerShellin järjestelmänvalvojan oikeuksilla ja suorittamalla Set-ExecutionPolicy-komennon (Komento 1). (Robbins, 2018, ss. 10–12)

Komento 1 Komento, jolla voi muuttaa suorituskäytännön

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

PowerShellissa on käytössä myös toiminto, joka määrittää jokaisen suoritetun komennon vaikutustason (impact level). Koska PowerShell on luotu tehokkaaksi järjestelmätyökaluksi, voi sen huolimattomalla käytöllä saada aikaan suurta vahinkoa kohdejärjestelmässä. Vaikutustaso-toiminnolla PowerShell arvioi kunkin suoritettavan komennon ja vertaa sitä etukäteen määritettyyn kynnysarvoon. Jos suoritettava komento vastaa kynnysarvoa tai ylittää sen, käyttäjältä pyydetään komennon suorittamisen jälkeen vielä erillinen varmistus komennon suorittamisesta. Tällä ominaisuudella pyritään välttämään tilanteita, joissa käyttäjä suorittaa epähuomiossa komennon, jonka vaikutus on oletettua suurempi.

PowerShellin impact level -kynnysarvon voi tarkistaa suorittamalla PowerShellissa komennon $confirmpreference. (Hassell, 2017, s. 35)

2.4 PowerShell-komennot ja niiden rakenne

PowerShellissa käytössä olevista komennoista käytetään nimitystä cmdlet, joka lausutaan ”command-let” (Hassell, 2017, s. 27). Cmdlet-komennot suorittavat halutun toimenpiteen ja palauttavat normaalisti .NET-objektin käyttäjälle tai seuraavalle komennolle. Toisin kuin monissa muissa komentokehotteissa, PowerShellissa käsitellään tekstin sijaan .NET-objekteja, jotka sisältävät paljon informaatiota. (Microsoft, 2020a)

PowerShell-ikkunassa voi suorittaa myös ulkoisia komentoja. PowerShell hyväksyy DOS-komentokehotteen komentoja, kuten ping, tracert ja net use. Tuntemattomampien komentojen kohdalla PowerShell saattaa kuitenkin antaa virheilmoituksia ja epäonnistua komennon suorittamisessa, sillä PowerShell ei tunnista kyseessä olevan DOS-komento. Ongelman voi kiertää käyttämällä kahta väliviivaa (--) komennon jälkeen. Tämä merkintä kertoo PowerShellille kyseessä olevan DOS-komento, ja näin ollen PowerShell osaa siirtää komennon vanhaan komentokehoteikkunaan. (Hassell, 2017, s. 27)

PowerShell-komentokehotteeseen on lisätty muutamia toimintoja helpottamaan komentojen kirjoittamista ja käyttöä. PowerShellissa voi käyttää komennon automaattista täydennystä näppäimistön tabulaattorinäppäimellä. Jos käyttäjä haluaa esimerkiksi kirjoittaa komennon Get-Command, riittää, että komentokehotteeseen kirjoittaa Get-Co ja painaa tabulaattoria. Komentokehote osaa ehdottaa käyttäjälle tämän perusteella Get-Command-komennon. Automaattinen täydennys hakee aina kirjoitettua sanaa lähinnä olevan komennon aakkosten perusteella. Painamalla uudelleen tabulaattoria PowerShell hakee seuraavaksi lähinnä olevan komennon, mikäli komentoja löytyy. Samalla, kun PowerShell täydentää komennon, muuttuu komentojen isot ja pienet kirjaimet syntaksin mukaiseksi. (Microsoft, 2017)

PowerShell komentokehotteessa voi selata myös aiemmin suoritettuja komentoja näppäimistön nuolinäppäimillä. Ylöspäin-nuolinäppäimellä pääsee selaamaan vanhoja komentoja, ja alaspäin nuolinäppäimellä pääsee palaamaan takaisinpäin komentohistoriassa.

Myös tekstin kopioiminen ja liittäminen onnistuu PowerShell-ikkunassa. (Hassell, 2017, ss. 8–10)

PowerShell-komennot koostuvat aina ennalta määritetyn säännön mukaan, jotta komentojen kirjoittaminen ja ymmärtäminen olisi mahdollisimman helppoa. Komentokehotteessa käytettyä kieltä ja kirjoitusasua kutsutaan nimellä syntaksi. PowerShellin syntaksi muodostuu seuraavalla tavalla: Verbi-substantiivi -parametri arvo -parametri arvo -kytkinparametri. PowerShell-komennot alkavat aina verbillä, esimerkiksi Get, Stop tai Set. Verbin perään tulee väliviiva ja substantiivi, esimerkiksi Event, Table tai Process. Näistä verbi-substantiivi-osan komennoista käytetään nimeä cmdlet. Cmdlet-komentoja voi käyttää sellaisenaan, esimerkiksi Get-Process. Jotta komennosta saa tehtyä tarkemman tai komentoon saadaan lisämääreitä, voi komentoon liittää erilaisia parametreja. Parametrien avulla komennoista saa huomattavasti monipuolisemman tai komennon hakemaa sisältöä pystyy tarkentamaan. Parametreja voi liittää komentoon yhden tai useamman. Parametrin eteen lisätään aina väliviiva, ja parametrin jälkeen tulee välilyönti sekä arvo (Komento 2). (Hassell, 2017, ss. 21–22)

Komento 2 PowerShell-komento, jossa on käytetty syntaksia Verbi-substantiivi -parametri arvo -parametri arvo

```
Get-EventLog -LogName Application -Newest 10
```

Komennon loppuun voi liittää vapaaehtoisen kytkinparametrin eli switch-parametrin (Komento 3). Kytkinparametrit voivat olla joko päällä tai pois päältä. Mikäli parametrin jättää pois komennosta, PowerShell tulkitsee parametrin olevan pois päältä. Jos taas parametrin lisää komentoon, PowerShell tulkitsee parametrin olevan päällä. Kytkinparametreille ei määritetä arvoa. (Hassell, 2017, s. 22)

Komento 3 Komennossa on mukana kytkinparametri -verbose

```
Get-EventLog -LogName Application -Newest 10 -Verbose
```

Osa PowerShell-komennoissa käytetyistä parametreista on niin sanottuja sijaintiparametreja (positional parameter). Kyseisiä parametreja käytetään hyvin usein ja samalla tavalla eri

komennoissa. Koska sijaintiparametrit ovat aina samassa kohdassa komennossa, voi ne jättää kirjoittamatta. Komentoihin tulee tällöin vain sijaintiparametrin jälkeinen arvo. Vaatimuksena on kuitenkin, että arvo on kirjoitettu juuri oikeaan kohtaa komennossa. (Hassell, 2017, s. 24)

Komento 4 Komennosta on jätetty pois sijaintiparametri

```
Get-EventLog Application -Newest 10
```

Komennossa 4 Application on arvo sijaintiparametrille -LogName. Kyseinen sijaintiparametri sijoittuu aina verbi-substantiivi-osan jälkeen, ja näin ollen PowerShell osaa tulkita parametrin arvon oikein, vaikka itse parametria ei ole kirjoitettu komentoon. (Hassell, 2017, s. 24)

2.5 PowerShellin help-toiminto

PowerShellia käytettäessä tulee enemmin tai myöhemmin vastaan tilanne, jossa käyttäjä ei muista, miten komento kuuluu kirjoittaa tai tarvittava cmdlet ei ole tiedossa. Tätä tapahtuu usein varsinkin PowerShell-käytön alkuvaiheessa. Varsinkin monimutkaisemmissa komennoissa oikean syntaksin muistaminen voi olla hankalaa tai komennon rakennetta ei muista ulkoa. PowerShellissa on sisäänrakennettu ohjetoiminto, josta löytää ohjeen käytännössä jokaiseen cmdlet-komentoon, mukaan lukien eri PowerShell-moduuleiden mukana tulevat cmdletit. Ohjetoiminnon tulisi olla ensimmäinen kanava tiedon etsimisessä, sillä ohjeet ovat ajantasaisia, ja ne säästävät käyttäjän aikaa. PowerShellin ohjetiedostot voi päivittää komentokehotteessa suorittamalla järjestelmänvalvojan oikeuksin komennon Update-Help. (Hassell, 2017, ss. 28–29)

Ohjetoiminnon cmdlet on Get-Help. Get-Help-cmdletin voi liittää halutun komennon eteen, josta haluaa etsiä ohjeita. Jos käyttäjä tarvitsee lisätietoja esimerkiksi Get-Process-komennosta, voi PowerShelliin syöttää komennon 5 mukaisen komennon. (Microsoft, n.d.-a)

Komento 5 Haetaan PowerShellin help-toiminnolla tietoa komennosta Get-Process

```
Get-Help Get-Process
```

Oletuksena help-toiminto palauttaa yhteenvedon tai supistetun version komennosta löytyvästä ohjeistuksesta. Mikäli käyttäjä haluaa vielä enemmän ohjeita näkyviin, komentoon voi lisätä erilaisia parametreja. Esimerkiksi parametrilla -Full saa näkyviin koko ohjeartikkelin kyseisestä komennosta. Parametrilla -Online (Komento 6) saa avattua verkkoselaimeen komentoon liittyvän Microsoftin ohjesivun. (Microsoft, n.d.-a)

Komento 6 Avataan Microsoftin ohjesivu nettiselaimeen komennosta Get-Process

```
Get-Help Get-Process -Online
```

Mikäli käyttäjä ei tiedä etsimäänsä komentoa, help-toiminnolla voi etsiä myös tietyn sanan perusteella ohjeartikkeleita. Hakeminen onnistuu lisäämällä Get-Help-komennon jälkeen hakusana, jossa on tähtimerkit sanan ympärillä (Komento 7). (Hassell, 2017, s. 31)

Komento 7 Haetaan ohjeita help-toiminnolla sanasta item

```
Get-Help *item*
```

Kun käyttäjä ajaa komennossa 7 esitetyn komennon, PowerShell tulostaa komentokehotteeseen joukon cmdlet-komentoja, joissa on mukana sana ”item”. Käyttäjä voi etsiä haluamansa cmdletin ja hakea tästä jälleen lisätietoja komennon 6 mukaisesti. (Hassell, 2017, s. 31)

2.6 Putkitus

PowerShellia käytettäessä tulee usein tarve linkittää useita komentoja yhteen isommaksi kokonaisuudeksi. Esimerkiksi järjestelmästä haettu tieto voidaan haluta lähettää jatkokäsiteltäväksi toiselle komennolle. Tällainen komentojen toisiinsa liittäminen on yksi PowerShellin vahvuuksista ja mahdollistaa monimutkaisten tehtävien suorittamisen nopeasti ja vaivattomasti. Eri komentojen yhdistämistä toisiinsa yhdeksi kokonaisuudeksi kutsutaan putkitukseksi (piping). Putkituksessa käytettävät komennot erotetaan |-merkillä. Putkittaminen tehdään kuten komennossa Komento 8 on esitetty. Samalla periaatteella voi putkittaa usean erilaisen komennon yhdeksi kokonaisuudeksi, jossa tieto liikkuu aina vasemmanpuoleiselta komennolta oikealle (Komento 9). (Hassell, 2017, ss. 37–39)

Komento 8 Haetaan käynnissä olevat Chrome-prosessit ja välitetään saadut tiedot seuraavalle komennolle, joka listaa tiedot listamuotoon

```
Get-Process -Name chrome | Format-List
```

Komento 9 Haetaan Chrome-prosessit, valitaan ominaisuus id ja tulostetaan tiedot listamuotoon

```
Get-Process -Name chrome | Select-Object id | Format-List
```

Putkittamalla komentoja edellä kuvatulla tavalla, voi komennoista saatua tietoa tulostaa myös PowerShellin ulkopuolelle. PowerShellista pystyy tulostamaan tietoa esimerkiksi tiedostoon komennolla Out-File (Komento 10). (Hassell, 2017, s. 43)

Komento 10 Haetaan Firefox-prosessit ja tulostetaan ne firefox.txt -tiedostoon

```
Get-Process -Name firefox | Out-File firefox.txt
```

2.7 Tiedon tuonti ja vienti PowerShellissa

PowerShellilla voi kerätä paljon erilaista tietoa kohdejärjestelmästä, mutta tietoa voi tuoda PowerShelliin myös ulkoisista lähteistä monella eri tapaa. Esimerkiksi valmiit skriptit voivat pyytää käyttäjää antamaan tietoa PowerShelliin Read-Host-cmdletin avulla. Read-Host-cmdletilla käyttäjää pyydetään kirjoittamaan tieto PowerShell-ikkunaan, minkä jälkeen tieto tallennetaan usein muuttujaan myöhempää käyttöä varten (Komento 11). Tällä tavalla voidaan ottaa vastaan käyttäjältä myös esimerkiksi salasanoja käyttämällä -AsSecureString-parametria (Komento 12). Kyseinen parametri piilottaa syötettävän tiedon *-merkkien taakse ja tallentaa syötetyn tiedon SecureString-objektiin. (Microsoft, n.d.-d)

Komento 11 Pyydetään käyttäjää syöttämään tietokoneen nimi ja tallennetaan se muuttujaan $tietokone

```
$tietokone = Read-Host “Anna tietokoneen nimi”
```

Komento 12 Pyydetään käyttäjää syöttämään salasana ja tallennetaan se muuttujaan $salasana

```
$salasana = Read-Host "Anna salasana" -AsSecureString
```

Monessa eri tilanteessa on tarve tuoda tietoa ulkoisesta lähteestä PowerShelliin. Esimerkiksi valmiiksi haettu data voidaan haluta siirtää PowerShelliin käsittelyä varten. Kun tieto on saatu käsiteltyä, se pitää saada tallennettua jälleen tiedostoon. Välillä tietoa tarvitsee siirtää myös järjestelmästä toiseen, esimerkiksi aktiivihakemistosta Office 365 -palveluun. PowerShellilla käyttäjä voi hakea tietoa erilaisista tiedostoista käsittelyä varten. Yksi eniten käytetyistä tavoista on hakea tietoa csv-tiedostoista. Niissä tieto on usein lajiteltu valmiiksi sopiviin riveihin ja sarakkeisiin, joista PowerShell osaa hakea tiedon ja muodostaa näistä sopivia kokonaisuuksia. Tietojen hakeminen onnistuu cmdletilla Import-Csv (Komento 13). Tietojen hakemisen jälkeen tietoja voi käsitellä PowerShellissa eri komennoilla. (Hassell, 2017, ss. 44–47)

Komento 13 Polkuun C:\temp\yhteystiedot.csv tallennetusta csv-tiedostosta haetaan tiedot PowerShelliin

```
Import-Csv C:\temp\yhteystiedot.csv
```

Vastaavasti tietojen siirtäminen PowerShellista csv-tiedostoon onnistuu Export-Csv cmdletilla, kuten komennossa Komento 14 on esitetty. Kyseisen cmdletin kanssa käytetään -Path-parametria, jolla määritetään, mihin sijaintiin csv-tiedosto tallennetaan. (Hassell, 2017, ss. 47–48)

Komento 14 Haetaan Firefox-prosessit ja tallennetaan tiedot C:\temp\firefox.csv-tiedostoon

```
Get-Process -Name firefox | Export-Csv C:\temp\firefox.csv
```

Kuten komentojen rakenteesta voi huomata, Import-Csv ja Export-Csv komennot käyttävät hyväksi PowerShellin putkitusta. Samalla logiikalla tietoa pystyy tuomaan ja viemään myös muista lähteistä. Hyvänä nyrkkisääntönä voi pitää sitä, että tiedon tuontiin tarkoitetulle

Import-cmdletille on aina vastaparina tiedon vientiin tarkoitettu Export-cmdlet. (Hassell, 2017, ss. 47–49)

Erilaisista tekstitiedostoista pystyy tuomaan tietoa PowerShelliin helposti Get-Content-komennolla (Komento 15). Komento soveltuu tekstimuodossa olevan tiedon hakemiseen tiedostosta. (Hassell, 2017, ss. 27–28)

Komento 15 Luetaan tiedostossa Numerot.txt oleva tieto PowerShell-istuntoon

```
Get-Content C:\temp\Numerot.txt
```

PowerShell pystyy myös muuntamaan tietoa tiedostotyypistä toiseen. PowerShelliin voi esimerkiksi tuoda csv-tiedoston ja muuntaa tämän html-tiedostoksi. Tiedon muuntamiseen voi olla tarvetta, jos tietoa pitää siirtää kahden eri järjestelmän välillä. PowerShellilla voi myös kerätä tietoa järjestelmästä ja luoda kerätystä tiedosta eri tiedostoja. PowerShellissa on ConvertTo-alkuisia cmdlet-komentoja, joilla voi suorittaa tiedon muuntamista. Käyttäessä kyseisiä komentoja on käytettävä myös Out-File-komentoa, sillä pelkkä ConvertTo-komento muuntaa tiedon vain komentokehoteruudulle (Komento 16). (Hassell, 2017, ss. 49–51)

Komento 16 Haetaan Firefox-prosessit, muunnetaan tieto html-muotoon ja tallennetaan muunnettu tieto C:\temp\firefox.html-tiedostoon

```
Get-Process -Name firefox | ConvertTo-Html | Out-File
C:\temp\firefox.html
```

2.8 Objektit ja niiden ominaisuudet

Objekteilla tarkoitetaan tapaa, miten tietoa esitetään PowerShellissa. Tieto voi olla esimerkiksi taulukkona, tekstinä tai numeroiden muodossa. Kun tietoa säilötään objektiksi, käyttäjä pystyy käsittelemään tätä tietoa tehokkaasti eri komennoilla PowerShellissa. (Hassell, 2017, s. 65)

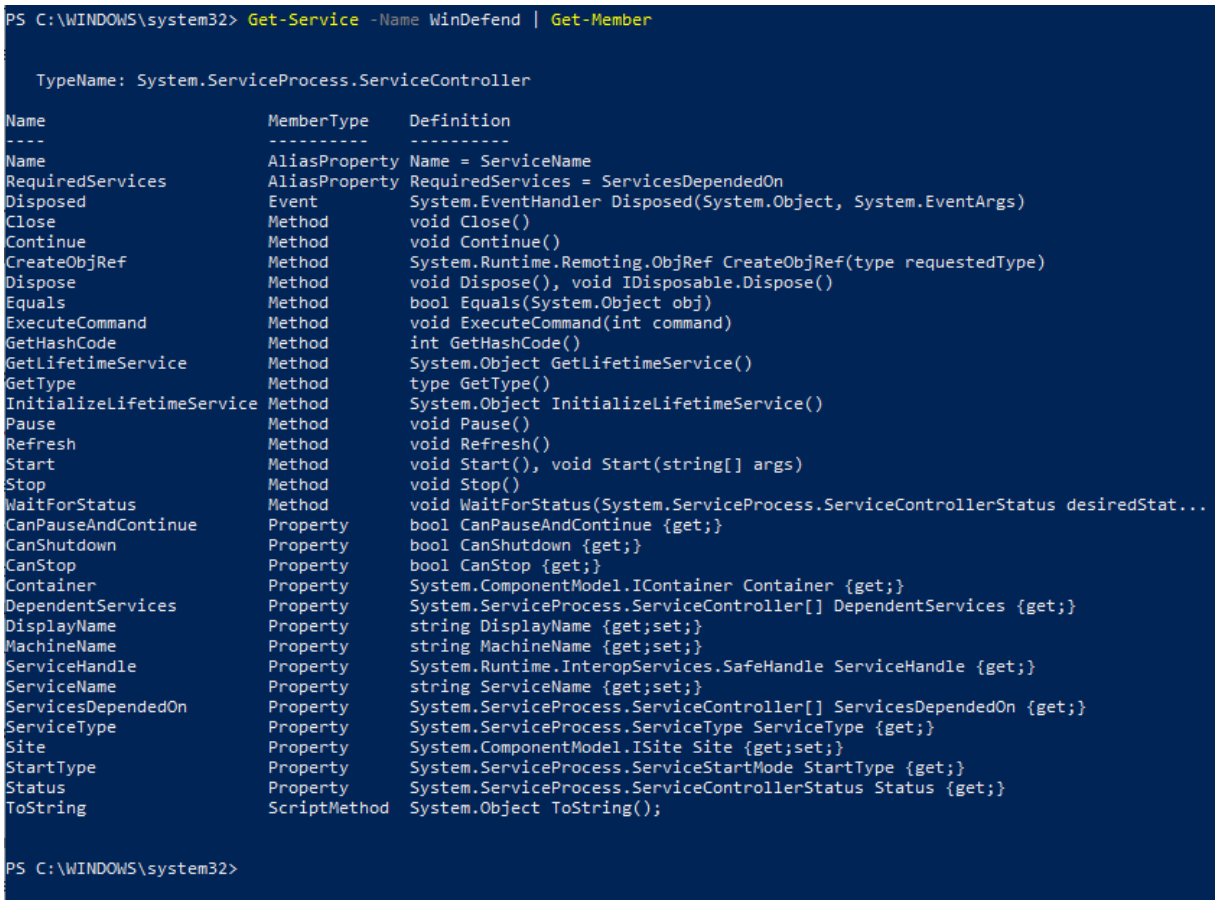
Käsiteltävän objektin tyyppi on yksinkertaista tarkistaa PowerShellissa Get-Member-cmdletilla (Komento 17). Get-Member-cmdlet tulostaa komentokehotteeseen listauksen

erilaisia tietoja valitusta komennosta (Kuva 2). Objektin tyyppi selviää ensimmäiseltä riviltä kohdasta TypeName. (Robbins, 2018, ss. 44–46)

Komento 17 Haetaan WinDefend-palvelu ja putkitetaan objekti tämän jälkeen Get-Member-komennolle

```
Get-Service -Name WinDefend | Get-Member
```

Kuva 2 Get-Member-cmdletilla tulostuu objektin tiedot



Kuvasta 2 selviää, että komennossa käsiteltävän palvelun WinDefend objektityyppi on System.ServiceProcess.ServiceController. Eri objektityyppejä on lukuisia erilaisia. Get-Member-komennolla selviää myös muita tärkeitä tietoja objektista. Kuvassa 2 näkyy objektityypin alapuolella joukko erilaisia metodeja (method) sekä ominaisuuksia (property). Ominaisuudet ovat nimensä mukaisesti erilaisia ominaisuuksia, mitä kyseinen objekti pitää sisällään. Jokaiseen ominaisuuteen liittyy arvo. Get-Member-komennon tuottamasta listauksesta näkee, millainen arvo ominaisuudella voi olla. Esimerkiksi ominaisuudella

ServiceName on string-tyyppinen arvo. Metodit ovat erilaisia toimintoja, joita kyseiselle objektille voidaan suorittaa. Kuvan 2 tapauksessa metodeilla Stop ja Start pystyy sammuttamaan tai käynnistämään WinDefend-palvelun. (Robbins, 2018, ss. 44–46)

Kun objektin tyyppi on selvillä, voi tämän perusteella tarkistaa, minkälaisia cmdlet-komentoja kyseiselle objektille on saatavilla. Komennolla 18 esimerkiksi voidaan tarkistaa, mitä cmdleteja on käytössä edellä mainitulle System.ServiceProcess.ServiceController-objektille. Objektin alkuosan voi jättää komennosta pois, jolloin jäljelle jää viimeisen pisteen jälkeinen sana ”ServiceController”. Käyttämällä Get-Member-cmdletia objektin tarkasteluun ja Get-Command-cmdletia eri cmdlet-vaihtoehtojen tarkasteluun, pystyy käyttäjä selvittämään helposti, mitä PowerShellin avulla voi tehdä eri tilanteissa. Kaikkea ei tarvitse muistaa aina ulkoa, kun tietää, millä komennoilla objekteja ja näiden komentoja voi tarkastella. (Robbins, 2018, ss. 44–46)

Komento 18 Tarkistetaan mitä cmdlet-komentoja on käytössä objektityypillä ServiceController

```
Get-Command -ParameterType ServiceController
```

2.9 Muuttujat

Muuttujia käytetään tyypillisesti silloin, kun halutaan säilöä tietoa myöhempää käyttöä varten. Muuttujiin voi tallentaa mitä tahansa tietoa, esimerkiksi sanoja, numeroita tai listoja. Muuttujaan voi tallentaa tietoa PowerShell-komennosta saadusta tuloksesta tai käyttäjä voi määrittää muuttujaan muun haluamansa arvon. Muuttujien käyttäminen on olennainen osa PowerShell-skriptien kirjoittamista. Muuttuja määritetään PowerShellissa komennon 19 mukaisesti. (Hassell, 2017, ss. 96–98)

Komento 19 Määritetään kolme erilaista muuttujaa

```
$numero = 12345
$teksti = "Tässä on esimerkkilause"
$listaNumeroita = 1, 7, 15, 20
```

Muuttuja määritetään antamalla $-merkki ja muuttujan nimi. Välilyönnin jälkeen tulee yhtäsuuruusmerkki, ja uuden välilyönnin jälkeen haluttu tieto. PowerShell osaa määrittää muuttujan tyypin aina automaattisesti. Mikäli muuttujaan syöttää tekstiä, on teksti syötettävä heittomerkkien tai lainausmerkkien sisään. Muuttujiin voi tallentaa myös PowerShell-komennoilla kerättyä tietoa. Tällöin muuttuja määritetään Komennon 20 mukaisesti. Määritetyn muuttujan voi tulostaa konsoliin milloin tahansa kirjoittamalla muuttujan nimen, esimerkiksi $loki. (Hassell, 2017, ss. 96–97)

Komento 20 Määritetään muuttujaan $loki PowerShellilla haettu Eventlog-merkinnät

```
$loki = Get-EventLog -LogName Application -Newest 6
```

Samalla logiikalla voidaan tallentaa muuttujaan tietoa esimerkiksi csv-tiedostosta. Toimenpide suoritetaan komennon Komento 21 mukaisesti. (Microsoft, n.d.-b)

Komento 21 Tallennetaan muuttujaan $yhteystiedot list.csv-tiedoston tiedot

```
$yhteystiedot = Import-Csv .\list.csv
```

PowerShell sisältää joukon valmiiksi määritettyjä muuttujia, joita voi hyödyntää komennoissa ja skripteissä. Vaikka kyseisiin muuttujiin voi määrittää haluttuja arvoja, Microsoft kehottaa vahvasti käyttämään näitä vain tiedon lukemiseen. Yksi valmiiksi määritellyistä muuttujista on $_ -muuttuja (sama kuin $PSItem-muuttuja). Kyseinen muuttuja sisältää objektin, joka on kyseisellä hetkellä komentoputkessa. $_ -muuttuja on tärkeä apu esimerkiksi tilanteissa, joissa jokin toimenpide pitää suorittaa kaikille objekteille kyseisessä komennossa. (Microsoft, 2021a)

2.10 Objektien käsittely

Usein objekteja käsiteltäessä on tarve tutkia tai erotella objektien sisältämää tietoa eri tavoilla. PowerShellissa on erilaisia cmdlet-komentoja, joiden avulla objektien sisältämää tietoa pystyy käsittelemään.

2.10.1 Sort-Object

Cmdletilla Sort-Object pystyy lajittelemaan kerättyä tietoa monella eri tavalla. Lajittelua pystyy tekemään esimerkiksi nimen tai koon mukaan. Sort-Object-cmdletille putkitetaan aina aiemmin haettu tieto, kuten komennosta 22 nähdään. (Hassell, 2017, ss. 74–77)

Komento 22 Haetaan C:\temp kansiossa oleva sisältö ja lajitellaan ne koon mukaan nousevaan järjestykseen

```
Get-ChildItem C:\temp | Sort-Object -Property Name -Descending
```

2.10.2 Compare-Object

Compare-Object-cmdletilla pystyy vertailemaan kahta eri objektia keskenään. Komento antaa käyttäjälle listauksen tiedoista, missä kohtaa tiedot eroavat toisistaan. Verrattavat objektit voi tallentaa muuttujiin esimerkiksi tiedostoista tai PowerShellissa suoritetuista komennoista. Compare-Object-cmdletissa määritetään -ReferenceObject-parametrilla objekti, joka toimii vertausobjektina. Parametrilla -DifferenceObject määritetään objekti, jota verrataan ensimmäiseen objektiin. Komennossa 23 haetaan vertailtavat tiedot txt-tiedostoista, kun taas komennossa 24 vertaillaan muuttujia, joihin on tallennettu tietoja tietokoneen prosesseista. (Hassell, 2017, ss. 111–117)

Komento 23 Tallennetaan tekstitiedostojen sisältö kahteen eri muuttujaan ja suoritetaan muuttujille vertailu

```
$teksti1 = Get-Content C:\temp\testi1.txt
$teksti2 = Get-Content C:\temp\testi2.txt
Compare-Object -ReferenceObject $teksti1 -DifferenceObject $teksti2
```

Komennon 23 tuloksena syntyy listaus poikkeavista riveistä tekstitiedostoissa (Kuva 3). Kuten kuvasta 3 voi päätellä, vertailun kohteena olevassa tiedossa on poikkeavuus referenssitiedostoon verrattuna. Kohdetiedostossa on kolmannella rivillä teksti: rivi tekstia. Referenssitiedostossa kyseisellä rivillä on teksti: Kolmas rivi tekstia.

Kuva 3 PowerShellin tulostama vertailu

```
PS C:\WINDOWS\system32> $teksti1 = Get-Content C:\temp\testi1.txt
PS C:\WINDOWS\system32> $teksti2 = Get-Content C:\temp\testi2.txt
PS C:\WINDOWS\system32> Compare-Object -ReferenceObject $teksti1 -DifferenceObject $teksti2

InputObject         SideIndicator
-----------         -------------
rivi tekstia        =>
Kolmas rivi tekstia <=


PS C:\WINDOWS\system32>
```

Komennossa 24 haetaan tietokoneen prosessit muuttujaan $prosessit1. Tämän jälkeen tietokoneessa käynnistetään Chrome-selain ja tallennetaan prosessit muuttujaan $prosessit2. Sitten muuttujia vertaillaan keskenään. Parametrilla -Properties valitaan tulostukseen näkyviin prosessin nimi.

Komento 24 Tallennetaan kahteen eri muuttujaan käynnissä olevat prosessit ja vertaillaan näitä keskenään

```
$prosessit1 = Get-Process
$prosessit2 = Get-Process
Compare-Object -ReferenceObject $prosessit1 -DifferenceObject
$prosessit2 -Property Name
```

Kuva 4 Komennon 24 tuloksena saadaan listaus prosesseista mitkä ovat muuttujassa $prosessit2 mutta puuttuvat muuttujasta $prosessit1

```
PS C:\WINDOWS\system32> $prosessit1 = Get-Process
PS C:\WINDOWS\system32> $prosessit2 = Get-Process
PS C:\WINDOWS\system32> Compare-Object -ReferenceObject $prosessit1 -DifferenceObject $prosessit2 -Property Name

Name                SideIndicator
----                -------------
chrome              =>
chrome              =>
chrome              =>
chrome              =>
chrome              =>
chrome              =>
chrome              =>
chrome              =>
chrome              =>
software_reporter_tool =>
software_reporter_tool =>
software_reporter_tool =>
software_reporter_tool =>


PS C:\WINDOWS\system32>
```

2.10.3 Select-Object

Kun laajasta joukosta eri objekteja haluaa valita vain tietyt objektit jatkokäsittelyyn, voi käyttää Select-Object-cmdletia. Cmdlet soveltuu tilanteisiin, joissa haetaan ensin suuri määrä tietoja eli objekteja, ja tämän jälkeen halutaan eritellä joukosta vain tietyt objektit jatkokäsittelyyn. Komennossa 25 Select-Object-cmdletia käytetään valitsemaan kolme viimeksi kirjoitettua tiedostoa. (Hassell, 2017, ss. 118–120)

Komento 25 Haetaan kansion C:\temp tiedostot, lajitellaan tiedostot viimeisimmän kirjoitusajan mukaan ja valitaan näistä tiedostoista kolme viimeisintä

```
Get-ChildItem C:\temp | Sort-Object -Property LastWriteTime | Select-
Object -Last 3
```

Select-Object-cmdletilla voi valita myös, mitä objektien ominaisuuksia halutaan tulostaa näkyviin. Komento 26 valitaan Select-Object-cmdletilla tulostukseen parametrit kansio, luontiaika sekä tiedostopääte. (Hassell, 2017, ss. 118–120)

Komento 26 Haetaan C:\temp-kansion sisältö ja tulostetaan näkyviin kansio, luontiaika sekä tiedostopääte

```
Get-ChildItem C:\temp | Select-Object -Property Directory,
CreationTime, Extension
```

2.10.4 Where-Object

Cmdletilla Where-Object voidaan suodattaa tietoa tai objekteja erikseen määritettyjen kriteerien ehdoilla. Suodatusta voi tehdä objektien ominaisuuksien perusteella, ja kriteerejä voi olla yksi tai useampi. Vertailua voi tehdä erilaisilla vertailuoperaattoreilla, joista muutamia on lueteltu taulukossa 2. Where-Object-cmdletin lauseke kirjoitetaan aina aaltosulkujen sisään, kuten Komento 27 selviää. Lausekkeessa voi käyttää muuttujaa $_, jolla viitataan viimeisimpään objektiin komentoputkessa. (Hassell, 2017, ss. 81–85)

Taulukko 2 Cmdletin Where-Object vertailuoperaattoreita (Hassell, 2017, s. 82)

Operaattori	Selite
-lt	vähemmän kuin
-le	vähemmän tai yhtä kuin
-gt	enemmän kuin
-ge	enemmän tai yhtä kuin
-ne	erisuuri kuin
-like	käytetään kun etsitään vastaavia tekstejä

Komento 27 Haetaan tietokoneen prosessit ja suodatetaan jäljelle ne prosessit missä suoritinaika on yli 1000 ms ja prosessin käyttämä muistimäärä on yli 20 mb

```
Get-Process | Where-Object {$_.CPU -ge 1000 -and
$_.PrivateMemorySize64 -ge 20MB }
```

2.11 PowerShell-etäyhteydet

PowerShellin yksi tärkeimpiä ominaisuuksia on eri tietokoneiden ja palvelinten hallinta etäyhteyden avulla. Käyttäjä pystyy etäyhteystoimintojen avulla hallitsemaan muita verkon laitteita omalta tietokoneeltaan PowerShellin kautta. Windows PowerShell käyttää etäyhteyden muodostamiseen WSMan- ja WinRM-tekniikoita. WSMan toimittaa komentoja verkossa tietokoneiden välillä http- ja https-tekniikoilla. WinRM kuuntelee WSMan-liikennettä ja vastaanottaa PowerShell-komennon kohdetietokoneessa. Jotta etäkäyttö on mahdollista, pitää tietokoneessa olla käytössä Windows PowerShellin etäyhteystoiminto. Toiminto otetaan käyttöön komennolla Enable-PSRemoting. Komennon suorittaminen vaatii järjestelmänvalvojan oikeudet. Myös tietokoneen verkon tyypin tulee olla joko domain tai yksityinen. Komennon suorittamisen jälkeen etäyhteystoiminto on käytössä vain

järjestelmänvalvojat-ryhmään kuuluvilla käyttäjillä. Mikäli etäyhteystoiminnon haluaa käyttöön muille käyttäjille, pitää nämä määritykset tehdä erikseen. (Hassell, 2017, ss. 139–142) Windows PowerShell tukee myös WMI- ja RPC-pohjaisia etäyhteystekniikoita. PowerShell Core 7 -versiossa on käytössä myös SSH-etäyhteysmahdollisuus. Tekniikka on käytössä Linux- ja Windows-ympäristöissä. SSH mahdollistaa esimerkiksi Linux-tietokoneen etähallinnan Windows-tietokoneelta. (Microsoft, 2020d)

2.11.1 Windows PowerShell -etäkomennot

Windows PowerShellissa on useita erilaisia cmdlet-komentoja, joita voidaan kohdistaa etäyhteyden avulla toisiin tietokoneisiin ilman erillistä etäyhteyden määrittämistä. Tämä onnistuu käyttämällä cmdletiin sisällytettyä ComputerName-parametria. Kyseisellä parametrilla voi määrittää tietokoneen tai useita tietokoneita, joihin kyseinen komento kohdistetaan. Komentoa suoritettaessa kohdetietokoneesta haetaan pyydetty tieto tai tehdään haluttu määritys. Tämän tyyppiset cmdletit käyttävät hyväkseen erilaisia Windows-järjestelmissä toimivia kommunikaatioprotokollia, eivätkä nämä vaadi erillisiä määrityksiä. Esimerkiksi seuraavat cmdletit sisältävät ComputerName-parametrin: Restart-Computer, Get-Process, Set-Service ja Get-WmiObject. (Microsoft, 2020e)

2.11.2 Windows PowerShell -etäyhteydet

Windows PowerShellilla voidaan muodostaa etäyhteys toiseen Windows-tietokoneeseen ajamalla Komento 28. Komento muodostaa uuden PowerShell-session, jonka kohdetietokoneena toimii komennossa määritetty tietokone. Session avaamisen jälkeen PowerShellia voi käyttää samaan tapaan kuin paikallista PowerShell-istuntoa. Etäyhteyssession ollessa käynnissä kaikki komennot kohdentuvat tietokoneeseen, johon etäyhteys on muodostettu. Kun etäyhteys halutaan sulkea, suoritetaan Komento 29. (Hassell, 2017, ss. 142–143)

Komento 28 Muodostetaan uusi PowerShell-sessio tietokoneeseen tietokone1

```
Enter-PSSession -computername tietokone1
```

Komento 29 Suljetaan käynnissä oleva PowerShell-etäyhteyssessio

```
Exit-PSSession
```

Kun halutaan suorittaa PowerShell-komento tai -skripti useassa tietokoneessa samaan aikaan, voidaan käyttää Invoke-Command-cmdletia (Komento 30). Suoritettaessa Invoke-Command muotoista komentoa, lähdetietokoneesta lähetetään komento kohdetietokoneeseen. Kohdetietokone vastaanottaa komennon ja avaa paikallisen PowerShell-prosessin. Komento suoritetaan tämän jälkeen kohdetietokoneessa, ja saadut tiedot lähetetään takaisin lähdetietokoneeseen. Samalla komentoa varten avattu PowerShell-sessio sulkeutuu. Toisin kuin PSSession-komennossa, Invoke-Command-komennolla avataan aina erillinen PowerShell-istunto kohdetietokoneessa komennon suorittamisen ajaksi. Oletuksena Invoke-Command-komento suoritetaan samanaikaisesti maksimissaan 32:ssa tietokoneessa. Rajoituksen tarkoituksena on vähentää verkon ylikuormitusta. Mikäli komennossa määritetään yli 32 tietokonetta, suoritetaan komento 32:n tietokoneen erissä. Rajoitusta voi muuttaa tarvittaessa -ThrottleLimit-parametrilla. (Hassell, 2017, ss. 143–145)

Komento 30 Suoritetaan Get-Process-komento tietokoneissa tietokone1, tietokone2 ja tietokone3 Invoke-Command-cmdletia käyttämällä

```
Invoke-Command -ComputerName tietokone1,tietokone2,tietokone3 -
ScriptBlock { Get-Process }
```

Komento 31 käytetään tietokoneiden listauksessa tekstitiedostoa. Tietokoneiden nimet voi olla listattuna tiedostossa allekkain siten, että yksi tietokone on aina yhdellä rivillä. Nimien perässä ei tarvitse olla merkkejä. (Hassell, 2017, ss. 146–147)

Komento 31 Suoritetaan Get-Process-komento tietokonejoukolle, jotka ovat listattuna tietokoneita.txt-tiedostossa

```
Invoke-Command -ScriptBlock { Get-Process } -ComputerName (Get-Content
tietokoneita.txt)
```

Invoke-Command-komennossa voi määrittää myös erillisen skriptitiedoston, joka sisältää suoritettavan komennon tai useita eri komentoja. Tällöin käytetään -FilePath-parametria -ScriptBlock-parametrin sijasta. Komento 32 on esimerkki -FilePath-parametrin käytöstä. (Hassell, 2017, ss. 145–146)

Komento 32 Suoritetaan komento.ps1 skriptitiedosto joukolle tietokoneita, jotka ovat listattuna tiedostossa tietokoneita.txt

```
Invoke-Command -ComputerName (Get-Content tietokoneita.txt) -FilePath
C:\temp\komento.ps1
```

Tietokoneista saapuvia tietoja voi tallentaa myös muuttujaan tiedon jatkokäsittelyä varten. Komento 33 käytetään muuttujaa tietojen tallentamiseen. (Microsoft, n.d.-c)

Komento 33 Tallennetaan muuttujaan $versio tietokoneista saatavat PowerShell-versiot

```
$versio = Invoke-Command -ComputerName (Get-Content tietokoneita.txt)
-ScriptBlock {$PSVersionTable.PSVersion}
```

Mikäli kohdetietokoneisiin suoritetaan useampia komentoja, on järkevää käyttää sessiotoimintoa ja tallentaa se muuttujaan. Kun käyttäjä määrittää session, kohdetietokoneisiin muodostetaan PowerShell-sessio. Sessio pysyy käynnissä kohdetietokoneissa myös komentojen suorittamisen jälkeen, eikä kohdetietokoneessa tarvitse avata jokaista suoritettavaa komentoa varten uutta yhteyttä. Sessio muodostetaan New-PSSession-cmdletilla, kuten komennosta Komento 34 nähdään. Kun sessio on tallennettu muuttujaan, voi kohdetietokoneita kutsua kyseisellä muuttujalla (Komento 35). Sessio suljetaan lopuksi Remove-PSSession-cmdletilla (Komento 36). (Robbins, 2018, ss. 116–117)

Komento 34 Muodostetaan uusi PowerShell etäyhteyssessio tietokoneisiin Win10V1 sekä Win10V2

```
$yhteys = New-PSSession -ComputerName Win10V1, Win10V2
```

Komento 35 Haetaan tietokoneista käynnissä olevat Chrome-prosessit ja tulostetaan näkyviin prosessin nimi ja käynnistysaika

```
Invoke-Command -Session $yhteys { Get-Process -Name chrome | Select-
Object Name, StartTime }
```

Komento 36 Suljetaan luotu sessio, joka on tallennettu $yhteys-muuttujaan

```
Get-PSSession | Remove-PSSession
```

2.12 PowerShell-skriptit

PowerShell-skripteillä tarkoitetaan valmiiksi luotuja tekstitiedostoja, joihin on kirjoitettu PowerShell-komento tai monia eri komentoja. PowerShellin skriptitiedostot ovat .ps1-päättyviä tiedostoja. Skriptitiedostoja suoritetaan PowerShell-ikkunassa syöttämällä skriptitiedoston kansiopolku, esimerkiksi C:\temp\testiskripti.ps1, tai mikäli PowerShell on avattu valmiiksi kyseiseen kansioon, riittää, että kirjoittaa skriptitiedoston nimen, esimerkiksi .\testiskripti.ps1. Skriptin voi suorittaa myös tiedostokuvakkeesta klikkaamalla hiiren oikealla painikkeella tiedoston päällä ja valitsemalla ”Run with PowerShell”. Skriptitiedostoilla voi suorittaa erittäin monimutkaisia tehtäviä aina käyttöjärjestelmäasennuksiin asti. Tämä tuo tarvittavaa tehokkuutta työasema- ja palvelinympäristön ylläpitämiseen. (Hassell, 2017, s. 95) PowerShell-skriptejä pystyy luomaan helposti Windowsin mukana tulevalla Windows PowerShell ISE -ohjelmalla sekä erikseen ladattavalla Visual Studio Code -ohjelmalla. Näistä kahdesta ohjelmasta Visual Studio Code on uudempi ja toimii PowerShell ISE -ohjelman seuraajana. (Peters et al., 2018, ss. 30–32)

3 Service desk

IT Service deskin eli käyttötuen pääasiallisena tehtävänä on tarjota asiakkaalle IT-tukea. Se voi toimia spoc-periaattella (single point of contact) eli asiakas ottaa yhteyttä kaikissa IT-ongelmissaan aina samaan service deskiin, joka ratkaisee asiakkaan ongelman tai pyynnön. Mikäli ongelmaa tai pyyntöä ei saada ratkaistua, käyttötuki kirjaa työpyynnön ja toimii sovitun toimintamallin mukaisesti. Service deskissä käytetään normaalisti palvelun ylläpitämiseksi jonkinlaista palvelunhallintaohjelmistoa, johon kirjataan työpyynnöt ja muita käsiteltäviä asioita. Yhteisen palvelunhallintaohjelmiston käyttö on tärkeää service desk -ympäristössä, jotta vältytään päällekkäiseltä työltä, ja eri tahoilla on pääsy tarvittaviin resursseihin. Palvelunhallintaohjelmiston avulla saadaan myös tarkkoja raportteja käyttötuen toimista ja työpyynnöistä. Service desk toimii ensimmäisen tason tukena ja lähteenä erilaisille palvelunhallinnan raporteille. (ITILnews.com, n.d.)

3.1 Service deskin tehtävät

Service deskin tehtävänä on toimia asiakkaalle ensimmäisenä yhteydenottokanavana kaikenlaisissa työasemaympäristöön liittyvissä ongelmissa. Työasemaympäristö käsittää mm. tietokoneet, oheislaitteet, ohjelmistot ja verkot. (Field Engineer, n.d.) Service desk toimii ammattitaitoisena osastona asiakkaan ja IT-palveluntarjoajan välissä. Asiakkaat voivat soittaa service deskiin jonkin ongelman ilmaantuessa tai kysyäkseen neuvoa eri tilanteissa. Service desk toimii usein työpyyntöjen hallitsijana ja omistajana koko työpyynnön elinkaaren ajan. Tämä tarkoittaa sitä, että työpyyntöjen ratkaiseminen sovittujen sopimusten puitteissa on service deskin vastuulla. (ITILnews.com, n.d.)

Service desk pyrkii ratkaisemaan asiakkaan ilmoittaman ongelman aina ensimmäisellä yhteydenotolla, mikäli tämä on mahdollista. Jos työpyyntöä ei saada ratkaistua, voi service desk välittää työpyynnön sopivalle asiantuntijataholle tai kolmannelle osapuolelle. Mikäli asiakkaan työasemaympäristössä ilmenee laaja häiriö, joka aiheuttaa merkittävää haittaa asiakkaalle, service deskin tehtävä on käynnistää laajahäiriöselvitys ja toimia häiriöviestintäkanavana asiakkaan suuntaan. (Klassen, 2018)

Service deskin perustehtäviä ovat esimerkiksi erilaiset tunnushallintatehtävät, ohjelmistoasennukset, käytönneuvonta eri sovelluksille sekä kaikenlaisten työasemaympäristössä ilmenevien vikatilanteiden selvittäminen ja korjaaminen. Service deskillä on käytössään erilaisia etäyhteystyökaluja, joilla työasemaympäristöä pystyy ylläpitämään etäyhteydellä. (Dahl, 2021)

3.2 Service deskin nykytilanne ja tulevaisuus

Service deskissä tulee vastaan erilaisia ongelmia, joita tukihenkilöt pyrkivät ratkaisemaan asiakkaalle. Usein kohdataan toistuvia työtehtäviä, joiden ratkaisemiseen on olemassa valmiita toimintatapoja. Nykypäivänä yksi service deskiin kohdistuvista tavoitteista on automatisoida mahdollisimman paljon erilaisia IT-tukitehtäviä. Automatisointia pyritään tekemään esimerkiksi erilaisten virtuaaliassistenttien avulla ja tarjoamalla analytiikan avulla hyödyllisiä ohjeita asiakkaalle eri tilanteissa. Service deskin työpyynnöistä ja loppukäyttäjäpalautteesta saatavien tietojen kautta pystytään myös kehittämään IT-ympäristöä oikeaan suuntaan. (Luhtala, 2020)

Service desk -kokemuksesta pyritään tekemään asiakkaalle aiempaa henkilökohtaisempaa. Tavoitteena on tuoda service desk -palvelut saataville kaikille internetiin liitetyille laitteille ja tehdä service deskistä helposti lähestyttävä. Automaation, virtuaalisten avustajien ja tekoälyn avulla service desk -palvelut ovat tavoitettavissa vuorokauden ympäri, ja asiakkaalla itsellä on mahdollisuus valita miten ja milloin hän käyttää palveluja. Tekoälyn ja kerätyn tiedon perusteella asiakkaalle voidaan tarjota apua silloin, kun asiakas sitä tarvitsee. (Fujitsu, n.d.)

4 Windows PowerShellin hyödyntäminen service deskissä

Tässä luvussa käsitellään erilaisia service deskissä esiintyviä työtehtäviä ja selvitetään, miten kyseiset työtehtävät voidaan suorittaa Windows PowerShellilla. Service desk -työ on välillä hyvin kiireistä, ja asioiden hoitamiseen saattaa olla niukasti aikaa. Tämän takia työtehtävien tehostamisesta on suurta hyötyä service desk -työssä. PowerShell on tehokas työkalu palvelin- ja työasemaympäristöjen hallinnassa, ja se tarjoaa erilaisia automaatiomahdollisuuksia. PowerShell-skripteillä pystyy suorittamaan monimutkaisia ylläpitotoimia ja muutoksia tehokkaasti aikaa säästäen. Luvussa esitellään muutamia erilaisia tapoja, miten Windows PowerShellia voi hyödyntää service deskissä esiintyvissä työtehtävissä. Tässä työssä käytetään Windows PowerShell 5.1 -versiota. Jatkossa Windows PowerShellista käytetään nimitystä PowerShell.

4.1 Haastattelu

Työn käytännön osuutta varten on haastateltu IT-yrityksen service deskissä työskentelevää henkilöä. Kohdeyritys on ICT-palveluja tuottava globaali yritys, joka työllistää Suomessa noin 1700 henkilöä. Haastateltava on toiminut pitkään service deskissä kyseisessä yrityksessä ja kuuluu tiimiin, joka vastaa usean eri asiakkaan käyttötuesta. Haastattelun tarkoituksena oli selvittää, miten kohdeyrityksen service deskissä hyödynnetään PowerShellia eri käyttökohteissa. Haastattelu jakautui neljään eri teemaan, jotka esitellään seuraavissa kappaleissa.

Haastattelun ensimmäisenä teemana oli PowerShellin hyödyntäminen aktiivihakemiston ylläpidossa sekä siihen liittyvissä työtehtävissä. Haastateltavan tiimissä käytetään PowerShellia lähinnä silloin, kun aktiivihakemistoon pitää tehdä suuri määrä muutoksia, esimerkiksi kustannuspaikkatiedon tai käyttöoikeusryhmän lisääminen suurelle joukolle käyttäjiä. Välillä saattaa tulla vastaan työtehtäviä, joissa pitää listata käyttäjätietoja AD-ryhmästä tai hakea käyttäjälistaus tietyn kustannuspaikan perusteella. Tavallisesti PowerShellia hyödynnetään siis työtehtävissä, joissa pitää käsitellä suurta määrää eri käyttäjiä ja tehdä näihin muutoksia, tai aktiivihakemistosta pitää koota isoja listauksia käyttäjistä tai muista objekteista.

Haastattelun toisena teemana oli PowerShellin etäyhteystoiminnot ja näiden hyödyntäminen toimialueen tietokoneiden hallinnassa. Tiimissä ei juurikaan käytetä PowerShellin etäyhteystoimintoja, koska usein työasemaympäristön ja tietokoneiden hallintaan on olemassa jokin toinen järjestelmä, jolla voi esimerkiksi jakaa sovelluksia ja korjausasennuksia tai tarkistaa tietokoneiden tietoja. Lisäksi joissain asiakkuuksissa voi olla rajoitteita tai rajoituksia, jotka estävät PowerShellin etäyhteystoimintojen hyödyntämisen. Näistä huolimatta on kuitenkin joitain työtehtäviä, joita voidaan suorittaa PowerShellilla. Näitä ovat esimerkiksi tiedon kerääminen tietokoneista. PowerShellia voidaan hyödyntää myös tilanteissa, joissa pienelle joukolle tietokoneita pitää tehdä järjestelmämuutoksia. Tästä esimerkkinä Windowsin rekisteriin tehtävät lisäykset tai muutokset.

Kolmantena teemana oli PowerShell-skriptien hyödyntäminen service desk -tehtävissä. Haastateltavan service desk -tiimissä on käytössä erilaisia skriptejä, jotka helpottavat ja nopeuttavat toistuvien työtehtävien hoitamista. Näistä esimerkkinä skripti, joka kopioi AD-käyttöoikeusryhmiä mallitunnukselta toiseen AD-tunnukseen. Tällä skriptillä on saatu tehostettua käyttöoikeuksien lisäämistä uusille tai muokattaville käyttäjätunnuksille, mikäli tunnusta pitää muuttaa manuaalisesti. Valmiita skriptejä löytyy myös esimerkiksi erilaisten käyttäjälistauksien hakemiseen aktiivihakemiston käyttäjistä. Uusia skriptejä voi tehdä kuka tahansa service desk -työntekijä, mutta usein ongelmaksi muodostuu vähäinen kokemus ja PowerShell-osaaminen. Tiimillä on käytössä jonkinlaisia testausmahdollisuuksia uusia skriptejä varten, mutta varsinaista testiympäristöä ei ole käytössä.

Haastattelun viimeisessä vaiheessa tiedusteltiin haastateltavalta, kokeeko hän PowerShell-työkalun tehostavan service desk -työtehtäviä. Haastateltavan mielestä PowerShell on hyvä työkalu, jonka avulla service deskissä suoritettavia työtehtäviä voi suorittaa hyvin tehokkaasti. PowerShell tehostaa työtehtävien suorittamista ja on hyvä lisä muiden järjestelmien rinnalla. Ongelmana on kuitenkin vähäinen PowerShell-osaaminen tiimin työntekijöiden keskuudessa. Tästä syystä usein PowerShellia vaativat työtehtävät kasautuvat tietyille henkilöille. Haastateltava näkee PowerShellin Windows-ympäristön tehokkaimpana ja monipuolisimpana työkaluna ja uskoo sen olevan myös tulevaisuudessa tärkeä työkalu Windows-ympäristön hallinnassa.

Haastattelussa ilmeni, että service deskissä käytetään PowerShellia myös Exchange- ja O365-ympäristön hallintaan. Tämä johtuu osaltaan siitä, että kaikkia toimenpiteitä ei pysty tekemään graafisen käyttöliittymän kautta vaan toimenpiteet on tehtävä PowerShellilla.

Haastattelua varten selvitettiin myös kohdeyrityksen muista service desk -tiimeistä, miten heidän tiimeissään käytetään PowerShellia. Lyhyen tiedustelun pohjalta kävi ilmi, että PowerShellin käyttö on vähäistä, mutta käytössä on joitakin PowerShell-skriptejä, joilla nopeutetaan toistuvia työtehtäviä.

4.2 Testiympäristö

Opinnäytetyötä varten on otettu käyttöön Windows Server 2016 -palvelin. Palvelimelle on asennettu aktiivihakemisto (AD), Hyper-V-palvelin sekä muut toimialueen pystyttämiseen tarvittavat komponentit. Hyper-V-alustan päälle on luotu kaksi Windows 10 -tietokonetta. Palvelin ja Windows 10 -tietokoneet ovat samalla jk.local-toimialueella. Työssä käytettävät komennot ja skriptit suoritetaan tässä ympäristössä. Työtä varten aktiivihakemistoon on luotu käyttäjiä sekä AD-ryhmiä, jotta tarvittavat komennot voidaan suorittaa ja testata.

4.3 PowerShellin hyödyntäminen aktiivihakemistossa

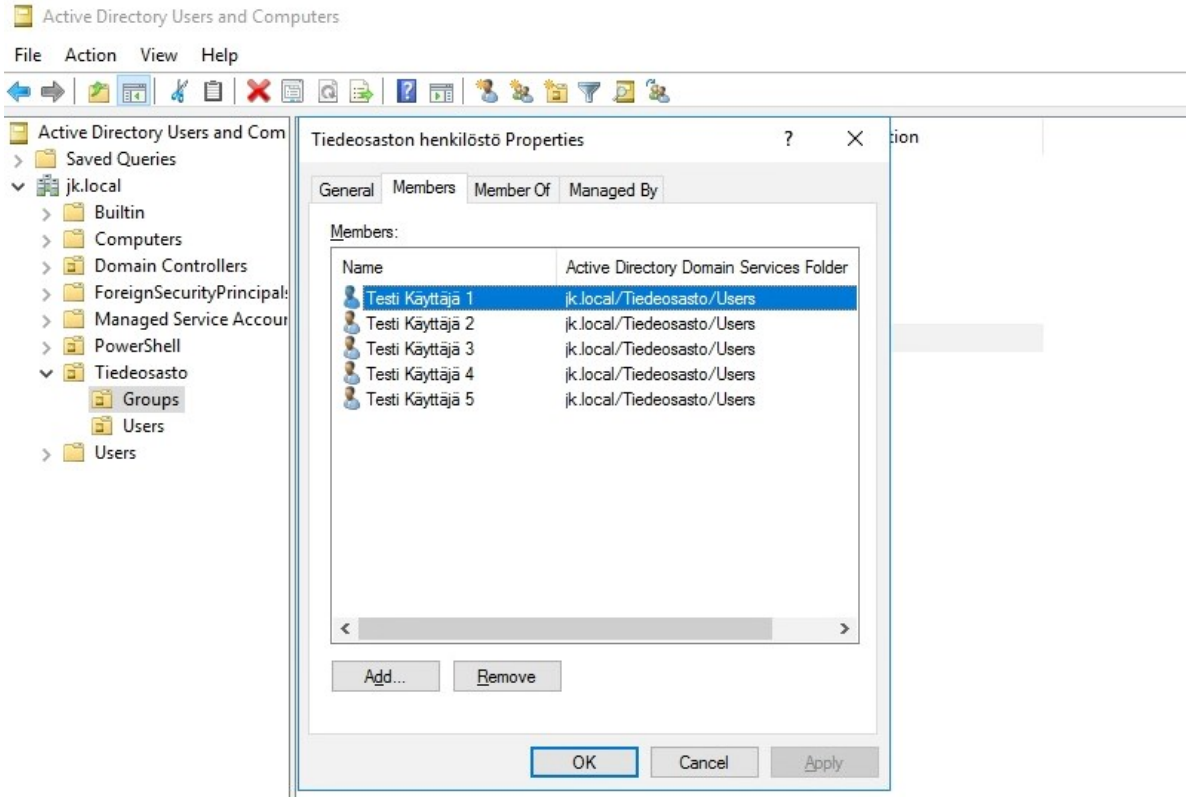
Service desk -tehtävissä tarvitsee tehdä usein muutoksia aktiivihakemistoon ja suorittaa erilaisia ylläpitotoimia. Varsinkin suurissa toimialueissa eri käyttäjiä ja AD-ryhmiä voi olla useita, ja näin ollen tehokas tapa hallinnoida sekä kerätä aktiivihakemiston tietoja on tärkeää.

4.3.1 Käyttäjälistan koonti aktiivihakemiston ryhmästä

Yksi mahdollinen ylläpitotehtävä on käyttäjälistojen koostaminen AD-ryhmistä. Näiden ryhmien käyttäjistä voidaan tarvita kerättävään listaukseen erilaista tietoa, kuten käyttäjätunnuksia ja kustannuspaikkatietoja. Käyttäjälistauksen kokoaminen on hyvin aikaa vievää työtä, jos listan joutuu kokoamaan manuaalisesti. Seuraavassa esimerkissä esitellään, miten listauksen pystyy koostamaan tehokkaasti PowerShellilla.

Tehtävänä on kerätä listaus AD-ryhmän Tiedeosaston henkilöstö -käyttäjistä (Kuva 5). Listassa pitää olla mukana käyttäjän nimi, käyttäjätunnus, kustannuspaikka sekä osasto. Listaus täytyy saada tallennettua tiedostoon, joka voidaan lähettää sähköpostilla esimerkiksi asiakkaalle.

Kuva 5 Tiedeosaston henkilöstö -ryhmän käyttäjät



Käyttäjälistaus voidaan hakea Get-ADGroupMember-cmdletilla, kuten komennosta Komento 37 nähdään. Komento hakee valitun AD-ryhmän käyttäjät. Komentoon lisätään mukaan halutut parametrit. Tässä tapauksessa listaukseen halutaan mukaan käyttäjien nimet, käyttäjätunnus, kustannuspaikka sekä osasto. Kustannuspaikka on tallennettu AD-tunnuksen office-kenttään ja osasto department-kenttään.

Komento 37 Haetaan AD-ryhmän Tiedeosaston henkilöstö -käyttäjät csv-tiedostoon valituilla lisätiedoilla

```
Get-ADGroupMember "Tiedeosaston henkilöstö" -recursive | Get-ADUser -
Properties * | select-object name, SamAccountName,
physicalDeliveryOfficeName, department | Export-Csv
C:\temp\Tiedeosaston_kayttajat.csv -Encoding UTF8 -NoTypeInformation
```

Kuva 6 Komennolla luotu csv-tiedosto

	A	B	C	D	E	F	G	H
1	name,"SamAccountName","physicalDeliveryOfficeName","department"							
2	Testi Käyttäjä 1,"testi1","100","Tiedeosasto"							
3	Testi Käyttäjä 2,"testi2","110","Tiedeosasto"							
4	Testi Käyttäjä 3,"testi3","110","Tiedeosasto"							
5	Testi Käyttäjä 4,"testi4","100","Tiedeosasto"							
6	Testi Käyttäjä 5,"testi5","110","Tiedeosasto"							
7								

Kuva 6 nähdään, millainen listaus komennolla saadaan aikaiseksi. Jaottelemalla saadut tiedot omiin soluihin Excelin automaattisella toiminnolla tiedostosta saa helposti luettavan. Tiedoston voi lähettää tämän jälkeen esimerkiksi asiakkaalle sähköpostin liitteenä (Kuva 7).

Kuva 7 Kerätyt tiedot on jaoteltu omiin soluihin

	A	B	C	D
1	name	SamAccountName	physicalDeliveryOfficeName	department
2	Testi Käyttäjä 1	testi1	100	Tiedeosasto
3	Testi Käyttäjä 2	testi2	110	Tiedeosasto
4	Testi Käyttäjä 3	testi3	110	Tiedeosasto
5	Testi Käyttäjä 4	testi4	100	Tiedeosasto
6	Testi Käyttäjä 5	testi5	110	Tiedeosasto
7				

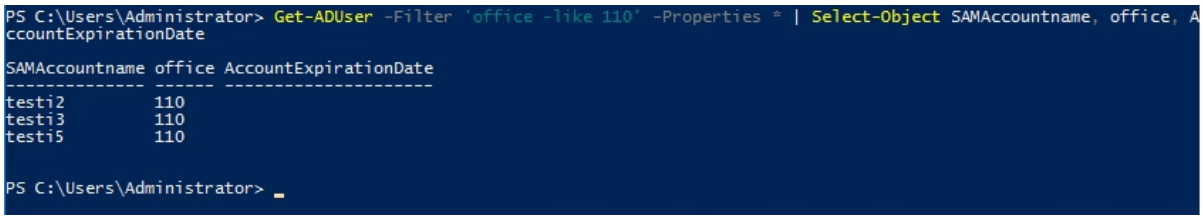
4.3.2 AD-attribuutin muuttaminen joukolta käyttäjiä

Aktiivihakemistossa saattaa joutua tekemään muutoksia isolle joukolle toimialueen käyttäjiä. Vastaan voi tulla tilanne, jossa käyttäjien AD-tunnuksiin pitää tehdä yhtenäinen muutos tai lisätä kokonaan uusi tieto tunnuksiin. Esimerkiksi kustannuspaikan muuttuessa käyttäjien kustannuspaikkatieto pitää muuttaa tai tunnuksiin määritetty voimassaoloaika päivittää tietyille käyttäjille. Seuraavassa esimerkissä muutetaan kustannuspaikkaan 110 kuuluvien käyttäjien tunnuksien voimassaolo 31.12.2021 asti.

Komento 38 Tarkistetaan tunnuksien tämänhetkinen voimassaoloaika. Haetaan tunnukset joiden office-kentässä on kustannuspaikka 110

```
Get-ADUser -Filter 'office -like 110' -Properties * | Select-Object
SAMAccountname, office, AccountExpirationDate
```

Kuva 8 PowerShellin tulostamat tiedot komennon 38 jälkeen



Kuva 8 selviää, ettei kustannuspaikan 110 tunnuksille ole määritetty tällä hetkellä voimassaoloaikaa. Kun on varmistettu, että Komento 38 saatu käyttäjälistaus on oikea ja sisältää oikeat käyttäjät, voidaan siirtyä määrittämään näille tunnuksille uusi voimassaoloaika. Kyseiseen tehtävään soveltuu foreach-komento. Kyseinen komento käy jokaisen putkessa olevan objektin läpi ja tekee sille määritetyn muutoksen (Komento 39).

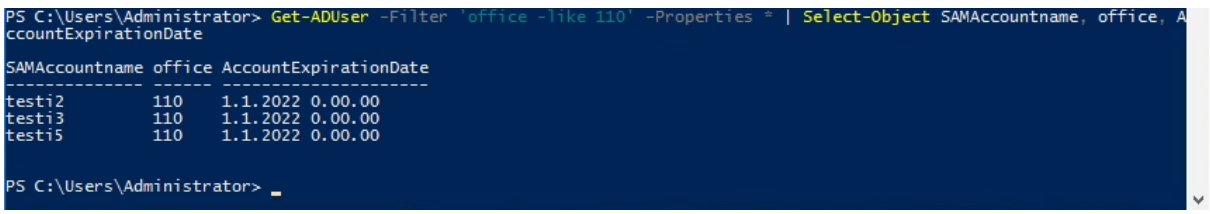
Komento 39 Haetaan office-kentän perusteella käyttäjät ja muutetaan käyttäjien tunnus poistumaan käytöstä 01.01.2022

```
Get-ADUser -Filter 'office -like 110' | foreach { Set-ADUser -Identity
$_.SamAccountName -AccountExpirationDate '01.01.2022' }
```

Tehtyjen määrityksien jälkeen täytyy vielä varmistaa, että muutokset ovat tulleet voimaan. Tämä voidaan tarkistaa suorittamalla uudelleen Komento 38. K

Kuva 9 voidaan todeta, että tunnuksien voimassaoloaika on muuttunut halutulla tavalla.

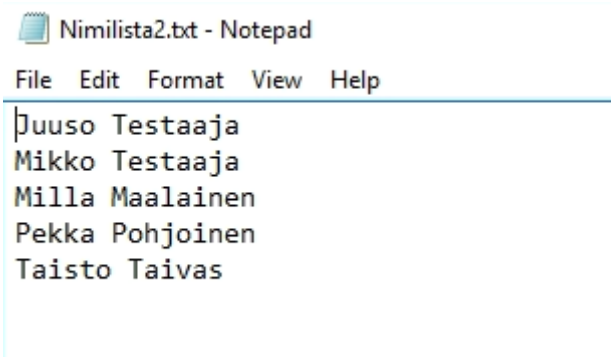
Kuva 9 Tarkistetaan vielä lopuksi, että käyttäjien voimassaoloaika on muuttunut halutulla tavalla



4.3.3 Käyttäjätunnusten hakeminen nimilistan perusteella

Tässä esimerkissä haetaan PowerShellin avulla valmiiksi listattujen henkilöiden käyttäjätunnukset. Käyttäjien nimet ovat listattuna txt-tiedostossa muodossa ”etunimi sukunimi”. Nimet ovat listattuna tiedostossa allekkain (Kuva 10).

Kuva 10 Tiedosto Nimilista2.txt



Ensimmäisenä toimenpiteenä nimitiedot pitää siirtää tekstitiedostosta PowerShelliin. Tämän voi tehdä Get-Content-cmdletin avulla (Komento 40).

Komento 40 Siirretään käyttäjien nimet tekstitiedostosta PowerShelliin ja tallennetaan muuttujaan $lista

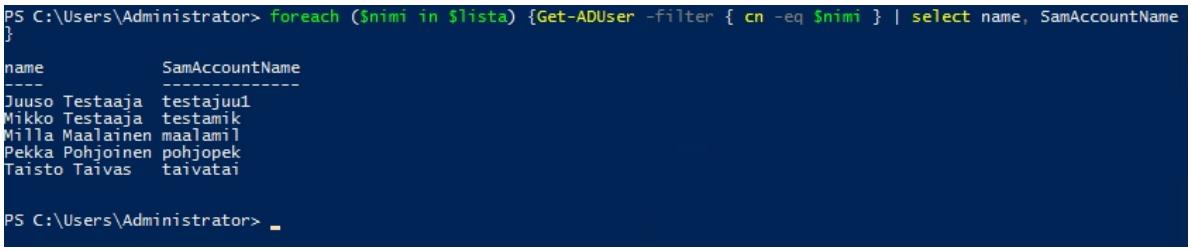
```
$lista = Get-Content -Path C:\temp\Nimilista2.txt
```

Kun nimet on saatu siirrettyä muuttujaan, haetaan muuttujan avulla jokaista nimeä vastaava käyttäjätunnus Komento 41 mukaisesti. PowerShell-ikkunaan tulostuu listaus käyttäjien nimistä sekä käyttäjätunnuksista (Kuva 11).

Komento 41 Haetaan muuttujaan tallennettuja nimiä vastaava käyttäjätunnus ja tulostetaan komentokehotteeseen näkyviin käyttäjän nimi sekä käyttäjätunnus

```
foreach ($nimi in $lista) {Get-ADUser -filter { cn -eq $nimi } |
select name, SamAccountName }
```

Kuva 11 PowerShell tulostaa näkyviin käyttäjän nimen sekä käyttäjätunnuksen



Käyttäjätunnukset sekä nimet voidaan tallentaa tämän jälkeen vielä esimerkiksi csv-tiedostoon (Komento 42). Kyseisen tiedoston voi aukaista Excelissä ja muuntaa tiedot omiin sarakkeisiin Excelin automaattisella toiminnolla (Kuva 12). Kun tiedot on jaoteltu omiin soluihinsa helposti luettavaan muotoon, voi tiedoston lähettää eteenpäin taholle, joka on pyytänyt kyseisiä tietoja.

Komento 42 Haetaan käyttäjien tiedot ja tallennetaan käyttäjien nimi sekä käyttäjätunnus csv-tiedostoon

```
foreach ($nimi in $lista) {Get-ADUser -filter { cn -eq $nimi } |
select name, SamAccountName | Export-Csv -Path
C:\temp\nimi_ja_tunnus.csv -Append -Encoding UTF8 -NoTypeInformation }
```

Kuva 12 Csv-tiedosto, jossa käyttäjätiedot on muutettu omiin soluihinsa Excelin automaattisella toiminnolla

	A	B	C
1	name	SamAccountName	
2	Juuso Testaaja	testajuu1	
3	Mikko Testaaja	testamik	
4	Milla Maalainen	maalamil	
5	Pekka Pohjoinen	pohjopek	
6	Taisto Taivas	taivatai	
7			
8			

4.3.4 Vaihtoehtoiset menetelmät

Edellä kuvatut esimerkkitapaukset ovat työtehtäviä, joita voi tulla vastaan service deskissä. Tapauksesta riippuen muutettavia AD-objekteja tai kerättäviä tietoja voi olla yksi tai useita kymmeniä. Mikäli kyseisiä tehtäviä ei suoriteta PowerShellilla, pitää tiedot hakea tai muuttaa manuaalisesti usein yksi käyttäjä kerrallaan. Yksittäisen tai muutaman AD-objektin tietoja on nopea ja helppo muuttaa graafisen käyttöliittymän kautta, mutta mikäli objekteja on useita, tulee tehtävästä helposti aikaa vievä prosessi.

Haettavan tiedon tai tehtävien muutoksien määrä määrittää sen, kannattaako työ tehdä PowerShellilla vai graafisen käyttöliittymän kautta. Usein yksittäisen tai muutaman käyttäjän tietojen tarkistaminen tai muuttaminen on nopeampaa tehdä graafisen käyttöliittymän kautta. Pitää myös muistaa, että vähemmän PowerShellia käyttänyt henkilö joutuu käyttämään työaikaa oikean komennon löytämiseen ja komennon testaamiseen. Mikäli haettavia tai muutettavia tietoja on paljon, on huomattavasti järkevämpää ja tehokkaampaa suorittaa tehtävä PowerShellin avulla. PowerShellin yksi vahvuus on isojen tietomäärien hallitseminen tehokkailla komennoilla. Muutamalla tarkkaan harkitulla komennolla pystyy suorittamaan satoja tai tuhansia operaatioita muutamassa sekunnissa. On myös hyvä muistaa, että kerran opeteltu tai koottu PowerShell-komento on helppo tallentaa muistiin uudelleenkäyttöä varten tai luoda komennosta skriptitiedosto. Näin saman työtehtävän suorittaminen seuraavalla kerralla on entistä nopeampaa.

4.4 PowerShellin etäyhteystoiminnot

Työasemaympäristössä joutuu tekemään monenlaisia ylläpitotoimia, joista monet työt vaativat etäyhteyden käyttöä. PowerShellin etäyhteystoiminnoilla pystyy suorittamaan monia tehtäviä tiedonkeruusta asetusten muuttamiseen ja asennuksiin. Erilaisia service deskissä suoritettavia työtehtäviä pystyy suorittamaan tehokkaasti PowerShellin avulla aikaa säästäen ja häiritsemättä asiakkaiden työskentelyä. Seuraavaksi käydään läpi muutamia esimerkkejä tällaisista tehtävistä.

4.4.1 Tiedoston siirtäminen ja suorittaminen etäyhteydellä

Toimialueen tietokoneisiin joutuu ajamaan usein erilaisia asennuksia tai päivityksiä asiakkaiden tarpeiden mukaan. Jotkin asennukset saattavat olla hyvin yksinkertaisia, mutta vievät kuitenkin paljon työaikaa, sillä tietokoneita saattaa olla enemmän kuin yksi, ja asiakkaan kontaktoiminen sekä etäyhteyden ottaminen vievät aina oman aikansa. PowerShellilla pystyy siirtämään helposti tiedostoja eri tietokoneille ja suorittamaan esimerkiksi exe-tiedostoja etäyhteydellä. Vaikka tietokoneita olisi useampi, ei aikaa kulu juuri sen enempää kuin yhden tietokoneen käsittelyssä. Tässä esimerkissä siirretään kahteen eri tietokoneeseen pakattu zip-tiedosto, puretaan paketti ja suoritetaan paketista purettu exe-tiedosto.

Ensimmäisenä työvaiheena haetaan txt-tiedostoon tallennetut tietokoneiden nimet PowerShelliin (Komento 43). Tietokoneiden nimet tallennetaan muuttujaan $tietokoneet.

Komento 43 Haetaan valmiiksi allekkain listatut tietokoneiden nimet tietokoneita.txt-tiedostosta

```
$tietokoneet = Get-Content C:\temp\tietokoneita.txt
```

Kun tietokoneiden nimet on tuotu $tietokoneet-muuttujaan, luodaan tämän jälkeen tulevia komentoja varten uusi tyhjä taulukko $err-muuttujaan (Komento 44). Kyseiseen muuttujaan tallennetaan PowerShellin antamia mahdollisia virheilmoituksia.

Komento 44 Luodaan uusi tyhjä taulukko $err-muuttujaan

```
$err=@()
```

Tämän jälkeen siirretään pakattu zip-tiedosto palvelimen C-asemalta kohdetietokoneisiin (Komento 45). Komentoon lisätään mukaan -ErrorVariable-parametri, jotta voidaan tarkastella mahdollisia PowerShellin antamia virheilmoituksia.

Komento 45 Kopioidaan jokaiselle $tietokoneet-muuttujassa määritetylle tietokoneelle zip-tiedosto palvelimelta

```
foreach ($tietokone in ($tietokoneet)) { Copy-Item –Path
C:\temp\testitiedosto.zip –Destination \\$tietokone\c$\temp -Verbose -
ErrorVariable +err }
```

PowerShellilla pystyisi siirtämään exe-tiedoston myös sellaisenaan ilman paketointia. Mikäli tiedostoja on useampia, on normaalisti helpompaa siirtää yksi pakattu tiedosto monen yksittäisen tiedoston sijaan. Tiedoston siirtämisen jälkeen tiedosto pitää purkaa. Purkaminen onnistuu samaan tapaan foreach-komennolla kuin tiedoston siirtäminen (Komento 46).

Komento 46 Puretaan siirretty tiedosto kohdetietokoneen C:\temp\purettu-kansioon

```
foreach ($tietokone in ($tietokoneet)) { Expand-Archive -Path
\\$tietokone\c$\temp\testitiedosto.zip -DestinationPath
\\$tietokone\c$\temp\Purettu -verbose -ErrorVariable +err}
```

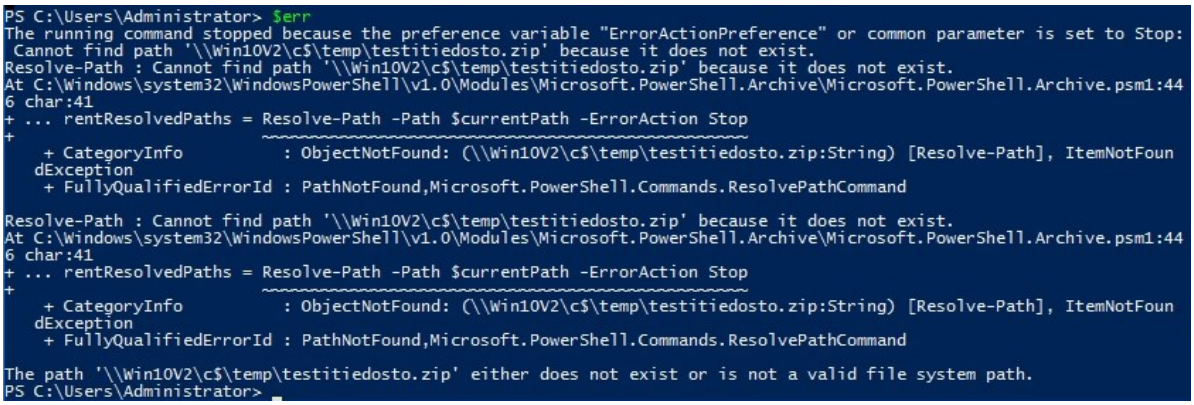
Kohdetietokoneiden C:\temp\Purettu-kansioissa on nyt purettuna exe-tiedosto, joka voidaan suorittaa lopuksi PowerShellin avulla. Kyseinen exe-tiedosto asentaa tietokoneelle 7-zip-ohjelmiston. Asennus suoritetaan hiljaisena asennuksena oletuskansioon (Komento 47).

Komento 47 Suoritetaan exe-tiedosto kohdetietokoneen C:\temp\Purettu-kansiosta

```
foreach ($tietokone in ($tietokoneet)) { Invoke-Command -ComputerName
$tietokone -ScriptBlock { C:\temp\purettu\7z1900-x64.exe /S /D } }
```

Esimerkkitietokoneisiin on asennettu nyt PowerShellin avulla 7-zip-ohjelma käyttäen hyväksi PowerShellin tarjoamia etäyhteystoimintoja. Samalla tavalla tietokoneelle tai useaan tietokoneeseen voi siirtää suoritettavan tiedoston erilaisia asennuksia tai muutoksia varten. Mikäli komentoja suorittaessa tulee virheilmoituksia, niitä pääsee tarkastelemaan $err-muuttujan avulla. Kuva 13 on jäänyt $err-muuttujaan virheilmoituksia, koska tietokoneeseen Win10V2 ei ole saanut yhteyttä komennon suorittamisen aikana.

Kuva 13 Komennon suorittamisen aikana ilmenneet virheilmoitukset



Komennot voisi tallentaa myös valmiiksi tehtyyn skriptitiedostoon, joka suorittaisi automaattisesti samat toimenpiteet. Skriptiin määritettäisiin mukaan kyselyt, joissa käyttäjältä pyydettäisiin ensin siirrettävän tiedoston hakemistopolku ja tarvittavat hakemistopolut kohdetietokoneista.

4.4.2 Järjestelmämuutokset useaan tietokoneeseen

Toimialueen tietokoneisiin voi joutua tekemään erilaisia korjauksia, jotka kohdistuvat rekisteriin, tietokoneen asetuksiin tai tietokoneessa oleviin ohjelmiin tai ajureihin. Itse muutos voi olla helppo toteuttaa, mutta jos tietokoneita on kymmeniä tai satoja, voi tehtävästä tulla haastava. Tässä esimerkissä näytetään, miten toimialueen tietokoneisiin voi muuttaa Windowsin rekisteriin rekisteriarvon. Rekisterin polusta HKEY_USERS\.DEFAULT\Control Panel\Keyboard halutaan muuttaa rekisteriavainta InitialKeyboardIndicators. Kyseinen rekisteriavain määrittää, onko näppäimistön numlock-painike oletuksena päällä vai pois päältä, kun tietokoneen käynnistää. Tässä esimerkissä rekisteriarvoksi halutaan määrittää 2 eli numlock on oletuksena päällä. Muutokset tehdään PowerShellin etäyhteystoimintoja käyttämällä. Toimenpiteen voi kohdistaa useaan eri tietokoneeseen yhtäaikaisesti.

Ensimmäisenä täytyy määrittää, mille tietokoneille tulevat toimenpiteet suoritetaan. Tässä esimerkissä valittujen tietokoneiden nimet ovat listattuna allekkain tietokoneet.txt-tiedostossa. Tietokoneiden nimet haetaan kyseisestä tiedostosta ensin PowerShelliin, jotta tietokoneisiin päästään kohdentamaan komentoja (Komento 48).

Komento 48 Haetaan tietokoneiden nimet C:\temp\tietokoneet.txt-tiedostosta PowerShelliin muuttujaan $tietokoneet

```
$tietokoneet = Get-Content -Path C:\temp\tietokoneet.txt
```

Kun tietokoneet on määritetty PowerShellin muuttujaan, voidaan määrittää etäyhteys kyseisiin tietokoneisiin. Yhteyden muodostamiseen käytetään PowerShellin New-PSSession-cmdletia. Yhteys tallennetaan muuttujaan $yhteys, jonka avulla tulevat komennot voidaan kohdistaa oikeisiin tietokoneisiin (Komento 49).

Komento 49 Muodostetaan PowerShell etäyhteyssessio $tietokoneet-muuttujaan tallennettuihin tietokoneisiin

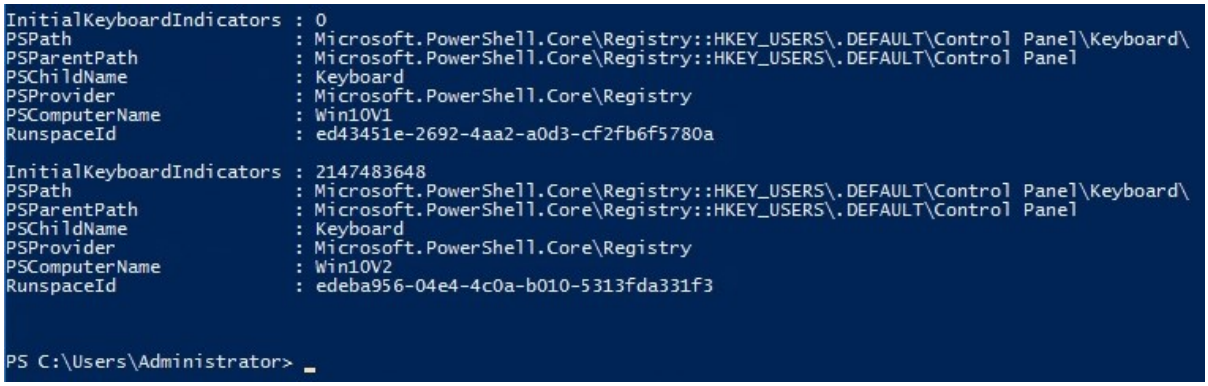
```
$yhteys = New-PSSession -ComputerName $tietokoneet
```

Kun PowerShelliin on muodostettu muuttuja, johon on tallennettu tarvittavat sessiotiedot, voidaan suorittaa haluttuja komentoja sessioon tallennettuihin tietokoneisiin. Tässä esimerkissä tarkistetaan ensimmäisenä, millainen rekisteriarvo tietokoneissa on tällä hetkellä polussa HKEY_USERS\.DEFAULT\Control Panel\Keyboard\InitialKeyboardIndicators (Komento 50).

Komento 50 Tarkistetaan, millainen rekisteriarvo on tallennettu avaimeen InitialKeyboardIndicators

```
Invoke-Command -Session $yhteys -ScriptBlock { Get-ItemProperty -Path
"Registry::HKEY_USERS\.DEFAULT\Control Panel\Keyboard\" -Name
"InitialKeyboardIndicators" }
```

Kuva 14 PowerShell tulostaa Komento 50 haetut tiedot komentokehoteikkunaan näkyviin käyttäjälle



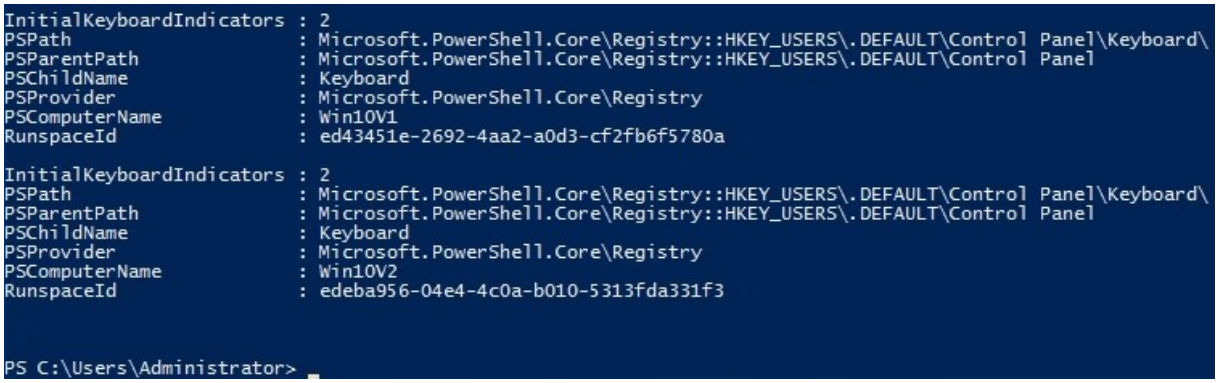
Komentokehoteruudulle tulostuvista tiedoista (Kuva 14) pystytään tarkistamaan, että tietokoneella Win10V1 rekisteriavaimen InitialKeyboardIndicators arvo on 0 ja tietokoneella Win10V2 arvo on 2147483648. Tarkoituksena on määrittää näppäimistön numlock-painike oletuksena päälle tietokoneen käynnistymisen yhteydessä, joten arvoksi pitää asettaa 2. Muutos voidaan suorittaa käyttäen Set-ItemProperty-cmdletia (Komento 51).

Komento 51 Muutetaan tietokoneiden rekisterissä InitialKeyboardIndicators-rekisteriavaimen arvoksi 2

```
Invoke-Command -Session $yhteys -ScriptBlock { Set-ItemProperty -Path
"Registry::HKEY_USERS\.DEFAULT\Control Panel\Keyboard\" -Name
"InitialKeyboardIndicators" -Value "2" }
```

Kun muutos on tehty, voidaan tarkistaa vielä Komento 50 mukaisesti, että rekisteriarvo on muuttunut. Kuva 15 voidaan huomata, että InitialKeyboardIndicators rekisteriavaimen arvo on muuttunut halutulla tavalla.

Kuva 15 Varmistetaan, että InitialKeyboardIndicators-rekisteriavainten arvoiksi on muuttunut 2

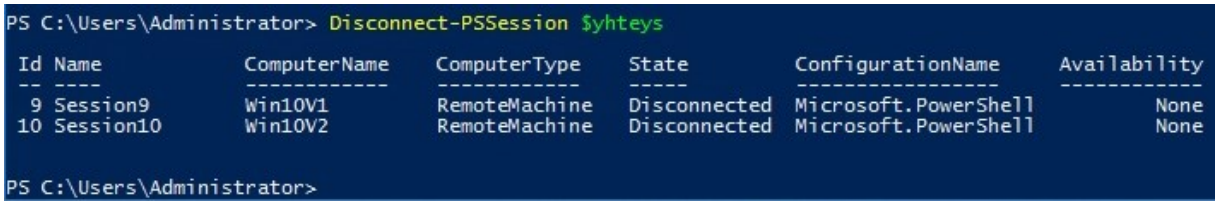


Kun halutut muutokset on saatu tehtyä, voidaan muutoksia varten kohdetietokoneisiin muodostettu PowerShell-sessio sulkea, jotta tietokoneille ei jää auki turhia yhteyksiä (Komento 52). Kuva 16 nähdään, että $yhteys-muuttujaan tallennettu sessio on suljettu.

Komento 52 Suljetaan $yhteys-muuttujaan tallennettu PowerShell-sessio

```
Disconnect-PSSession $yhteys
```

Kuva 16 PowerShell tulostaa näytölle tiedot katkaistusta sessiosta



4.4.3 Vaihtoehtoiset menetelmät

PowerShellin etäyhteystoiminnoilla pystyy suorittamaan tehokkaasti tietokoneiden ja palvelimien ylläpitotehtäviä suoraan ylläpitäjän tietokoneelta tai palvelimelta. Tämän etuna on mahdollisuus tehdä toimenpiteitä ilman kohdetietokoneen käyttäjän kontaktointia tai varsinaisen etäyhteyden muodostamista kohdetietokoneeseen tai palvelimeen. Mikäli tehtäviä ei tehdä PowerShellilla, täytyy muutokset tehdä kohdetietokoneisiin toista etäyhteystyökalua käyttämällä. Näitä ovat esimerkiksi Windowsin oma etätyöpöytäyhteys-

työkalu ja muut kolmansien osapuolien etäyhteysohjelmat. Nämä etäyhteystyökalut vaativat kuitenkin joko asiakkaan kontaktointia etäyhteyden muodostamiseksi tai tietokoneen vapauttamista etäyhteysistunnon käyttöä varten. Joitakin muutoksia pystytään toteuttamaan myös erilaisia jakelujärjestelmiä käyttämällä.

Mikäli useaan eri tietokoneeseen pitää tehdä muutoksia esimerkiksi vikatilanteen vuoksi, on muutos helppo tehdä PowerShellin avulla. Kun tarvittavat komennot on testattu toimiviksi, voidaan komento suorittaa etäyhteystoimintojen avulla tarvittaessa kaikkiin verkossa oleviin tietokoneisiin. Mikäli muutokset tai korjaukset tehdään muulla tapaa, esimerkiksi etätyöpöytäyhteyden avulla, täytyy muutokset tehdä jokaiseen tietokoneeseen erikseen, ja kohdetietokoneen käyttäjän täytyy kirjautua ulos tietokoneelta toimenpiteiden ajaksi.

Ensimmäisessä esimerkissä tehty tiedoston siirto ja kopiointi on mahdollista toteuttaa myös esimerkiksi käytössä olevalla ohjelmistojakelutyökalulla. Tällaista järjestelmää käytetään tavallisesti kuitenkin vain isommille jakeluille, esimerkiksi kun tietty ohjelmisto halutaan käyttöön suureen osaan toimialueen tietokoneista. Pienemmälle joukolle tehtävät ohjelmistoasennukset tai yksittäisten suoritettavien tiedostojen jakelu ei ole usein järkevää toteuttaa tällä menetelmällä. Tämän takia yleinen toimintatapa on ottaa etäyhteys tietokoneeseen ja suorittaa asennus manuaalisesti etäyhteydellä. Tämä toimii hyvin yksittäisissä tapauksissa, mutta mikäli tietokoneita on useita, vie manuaaliasennukset ja etäyhteydenmuodostamiset paljon työaikaa.

Samaan tapaan myös erilaisten järjestelmämuutosten tekeminen ilman PowerShellia etäyhteyden avulla vie huomattavasti aikaa, mikäli muokattavia tietokoneita on useita. PowerShellilla monet muutokset pystytään tekemään yhdellä komennolla, ja kohdentamalla komento useaan tietokoneeseen voidaan suuri määrä tietokoneita korjata yhdellä komennolla tai skriptillä. Lisäksi muutoksen onnistumisen todentaminen onnistuu yhtä helposti PowerShell-komennolla tai määrittämällä skripti kirjoittamaan lokitiedosto tehdyistä toimenpiteistä. Suoritetut toimenpiteet on myös helppo tarkastaa jälkikäteen PowerShellin komentohistorian avulla.

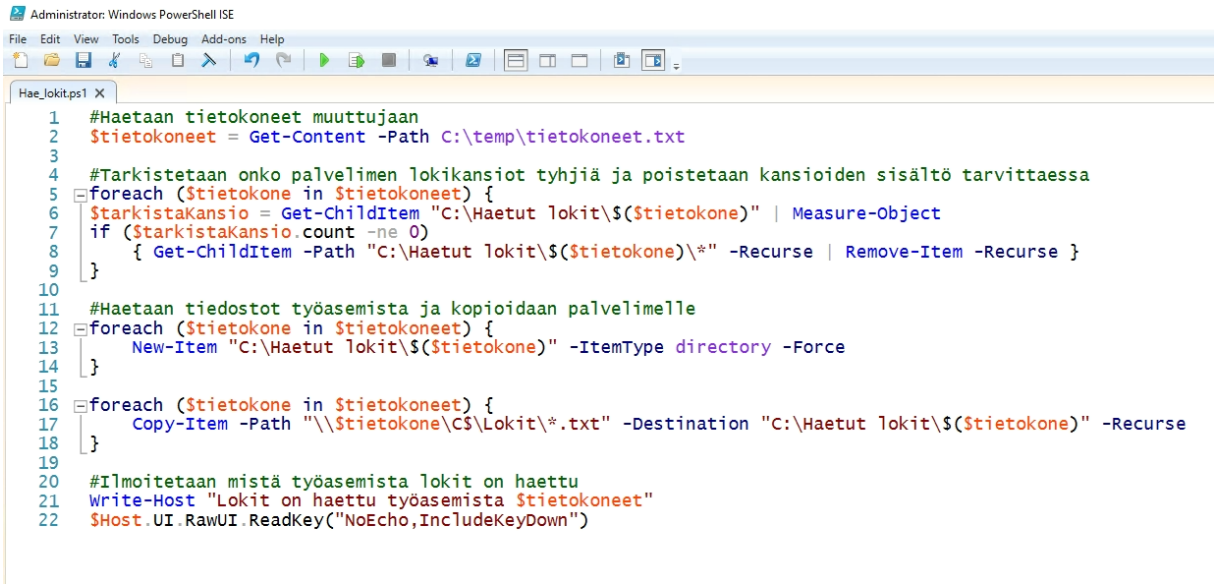
4.5 PowerShell-skriptit toistuvien työtehtävien tukena

PowerShellin tärkeimpiä ominaisuuksia on erilaisten skriptitiedostojen luominen ja suorittaminen. Skriptitiedostoja voi suorittaa PowerShellin kautta tai suoraan skriptitiedoston kuvakkeesta. Skriptitiedostoja voi ajastaa myös käynnistymään automaattisesti tai liittää erilaisten järjestelmien osaksi. Seuraavissa esimerkeissä luodaan erilaisia skriptitiedostoja, joiden tarkoituksena on tehostaa toistuvia työtehtäviä. Skriptitiedostot tehdään Windows PowerShell ISE -ohjelmalla.

4.5.1 Lokitiedostojen hakeminen toimialueen tietokoneista

Tarkoituksena on luoda skriptitiedosto, joka hakee toimialueen tietokoneista valitusta sijainnista txt-tiedostot ja kopioi ne palvelimelle määritettyyn kansioon. Tämän kaltainen skripti soveltuu tapauksiin, joissa tietokoneisiin muodostuu ennalta määrättyyn paikkaan tiedostoja, jotka pitää siirtää toiseen sijaintiin talteen tai jatkokäsittelyyn. Tässä esimerkissä käytetään txt-tiedostoja. Tiedostot haetaan kohdetietokoneista palvelimelle ohjelmakoodin 1 mukaisesti.

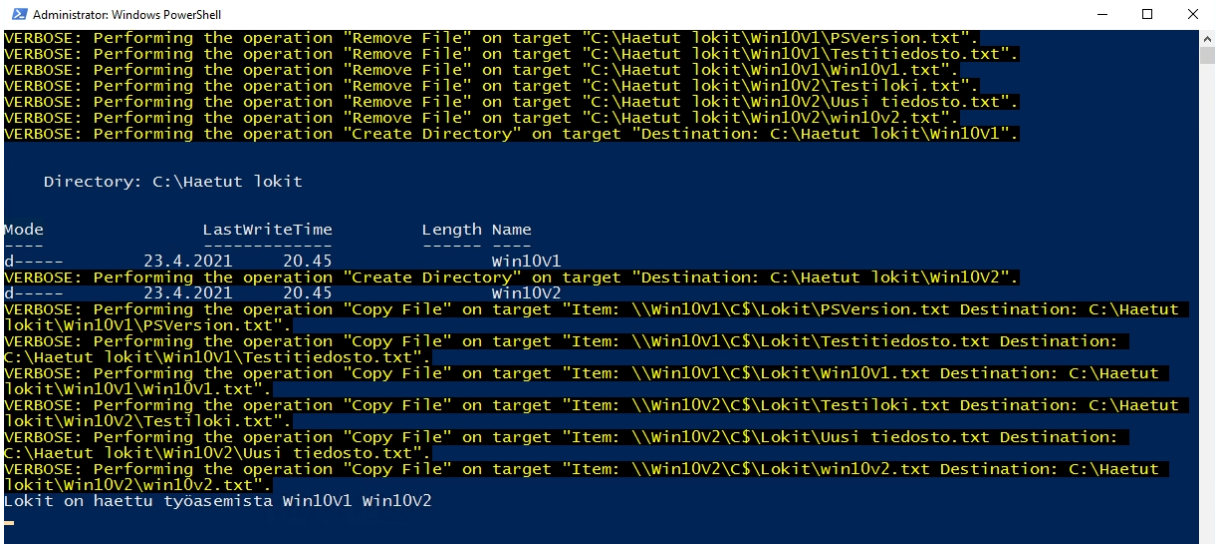
Ohjelmakoodi 1 Skriptitiedostoon määritetyt komennot



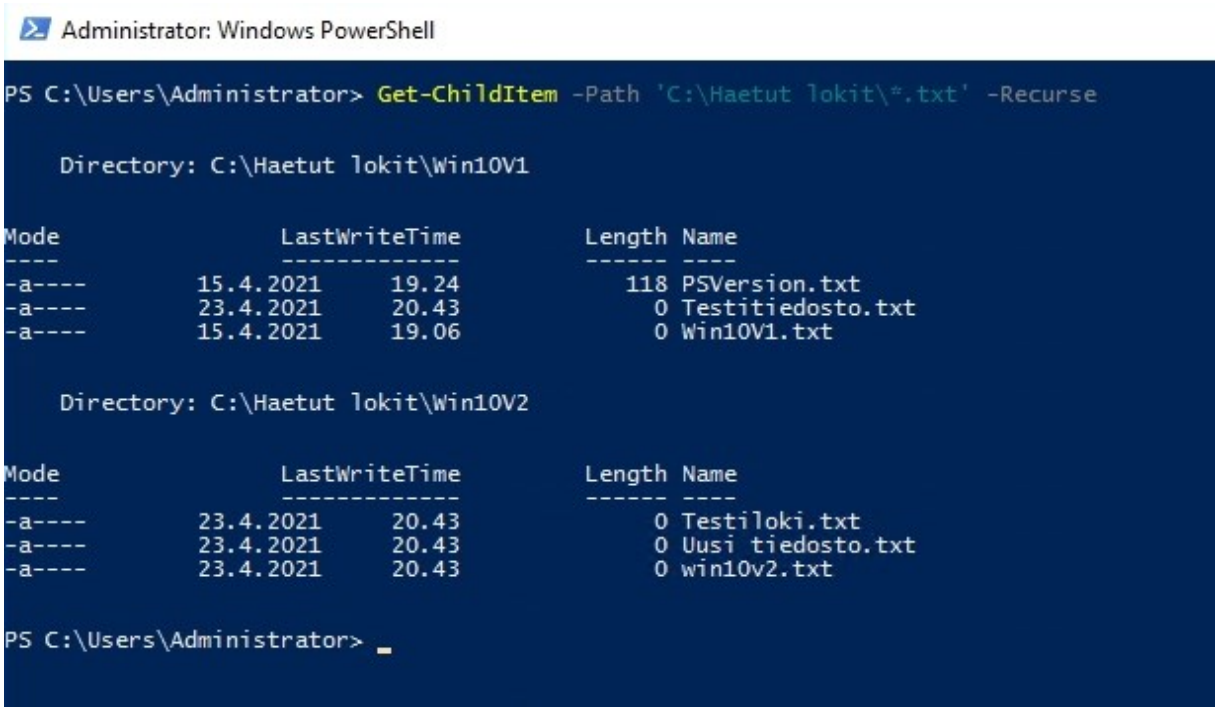
Tietokoneet, joista lokitiedostot noudetaan, on listattuna palvelimella C:\temp\tietokoneet.txt-tiedostossa. Skripti hakee tietokoneiden nimet muuttujaan ja

tarkistaa tämän jälkeen, onko palvelimella lokitiedostojen tallennuspaikassa jo valmiiksi aikaisempia lokitiedostoja. Jos tiedostoja löytyy, skripti poistaa tiedostot ja siirtää tämän jälkeen uudet tiedostot oikeisiin kansioihin. Skripti osaa muodostaa palvelimelle C:\Haetut lokit-kansioon kohdetietokoneen nimellä kansion, jonne skripti siirtää kyseisen tietokoneen lokitiedostot. Lopuksi käyttäjälle ilmoitetaan, mistä tietokoneista lokeja on noudettu. Kuva 17 nähdään, mitä skriptitiedosto suorittaa. Työvaiheet saa näkyviin, kun parametri -Verbose lisätään komentojen perään. Kuva 18 nähdään, että skripti on hakenut tiedostot kohdetietokoneista ja tallentanut ne palvelimelle C:\Haetut lokit-kansioon.

Kuva 17 Skriptin suorittamat välivaiheet



Kuva 18 Palvelimen C:\Haetut lokit-kansion sisältö

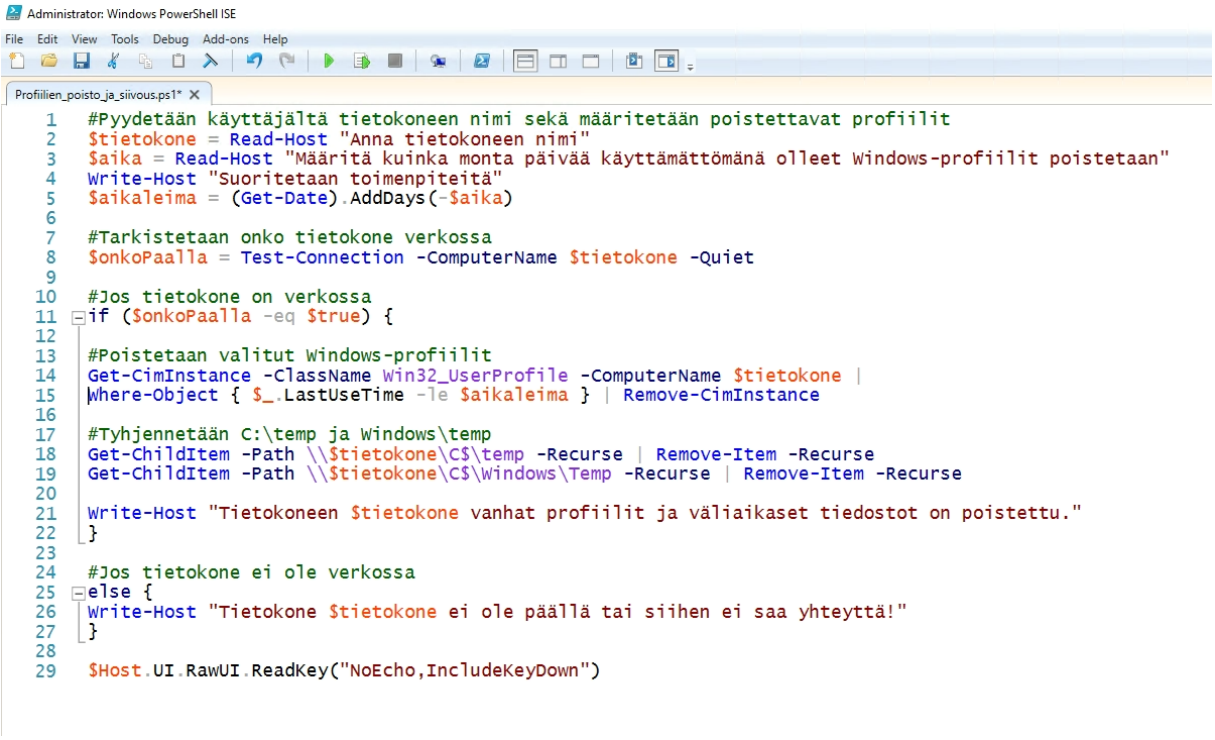


4.5.2 Tietokoneen siivous ja Windows-profiilien poistaminen

Service desk -tehtävissä kohdataan usein tilanteita, joissa joutuu poistamaan yhden tai useamman Windows-profiilin tietokoneesta. Profiilin poistoon voi olla monia eri syitä. Välillä vastaan saattaa tulla tietokoneita, joihin on kertynyt vuosien saatossa suuri määrä eri käyttäjien Windows-profiileja, jotka vievät kovalevyllä suuren määrän tilaa. Usein profiileja joutuu uusimaan myös korjauskeinona ilmenneeseen ongelmaan. Profiilien poistaminen Windowsin asetuksien kautta on aikaa vievä toimenpide varsinkin, jos niitä pitää poistaa suuria määriä. Profiileja on mahdollista poistaa myös PowerShellin avulla joko yksittäin tai useampia kerrallaan. Seuraavassa esimerkissä luodaan skripti (Ohjelmakoodi 2), joka poistaa tietokoneesta halutut profiilit perustuen viimeisimpään kirjautumiskertaan.

Samalla, kun skripti (Ohjelmakoodi 3) poistaa tietokoneesta valitut profiilit, skripti käy läpi ennalta määrätyt kansiot ja poistaa näistä sisällön. Tarkoituksena on poistaa väliaikaisia tiedostoja ja vapauttaa tietokoneen kovalevylle tilaa.

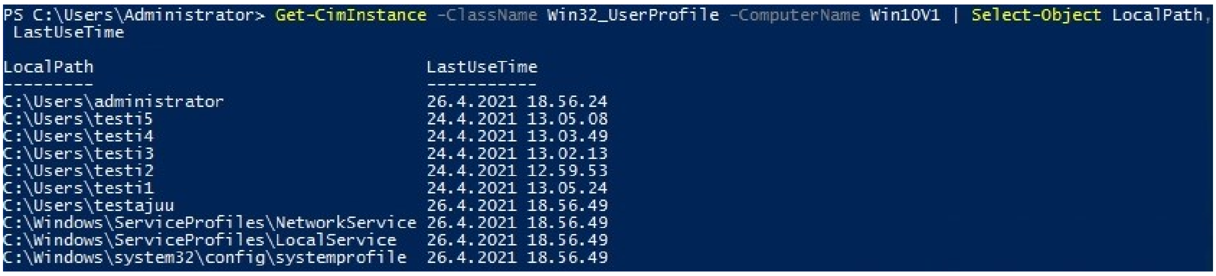
Ohjelmakoodi 4 Skripti, joka poistaa profiilit ja väliaikaiset tiedostot



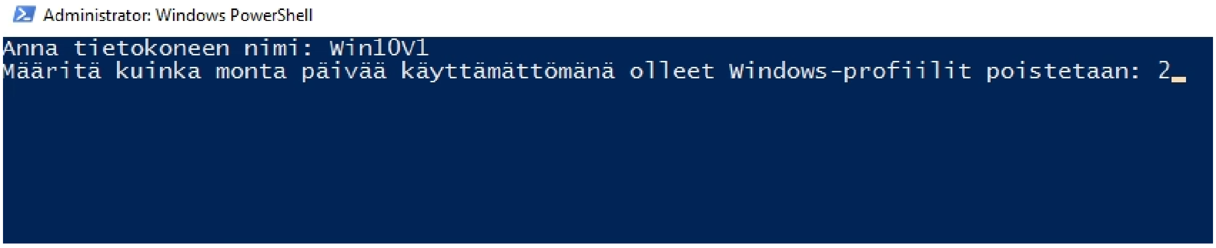
Skripti kysyy käyttäjältä ensimmäisenä tietokoneen nimen, johon toimenpiteet kohdistetaan. Tämän jälkeen käyttäjältä kysytään vielä, kuinka monta päivää käyttämättömänä olleet Windows-profiilit poistetaan. Näiden kysymysten jälkeen skripti tarkistaa, saako käyttäjän määrittämään tietokoneeseen yhteyden. Jos tietokoneeseen ei saa yhteyttä, ilmoitetaan siitä käyttäjälle, ja skriptin suorittaminen keskeytetään. Mikäli tietokone vastaa verkossa, suoritetaan Windows-profiilien poisto sekä ennalta määritettyjen kansioiden tyhjennys. Lopuksi käyttäjälle ilmoitetaan, että suoritetut toimenpiteet on tehty. PowerShell-skriptin suorittamisen jälkeen kohdetietokoneesta on poistettu vanhat Windows-profiilit ja temp-kansiot on tyhjennetty.

Skriptin toimivuuden varmistamiseksi voidaan varmistaa, että skripti tekee halutut toimenpiteet. Kuva 19 nähdään tämänhetkiset Windows-profiilit tietokoneesta Win10V1. Tarkistamisen jälkeen käynnistetään skripti ja syötetään tietokoneen nimi sekä aikamääre (Kuva 20). Tietojen syöttämisen jälkeen, skripti alkaa käymään läpi poistettavia Windows-profiileja sekä tyhjättäviä kansioita (Kuva 21). Kun skripti on tehnyt kaikki toimenpiteet ja sulkeutunut, voidaan tarkistaa tietokoneen Win10V1 Windows-profiilit. Kuva 22 nähdään, että skripti on poistanut tietokoneesta yli kaksi päivää vanhat profiilit.

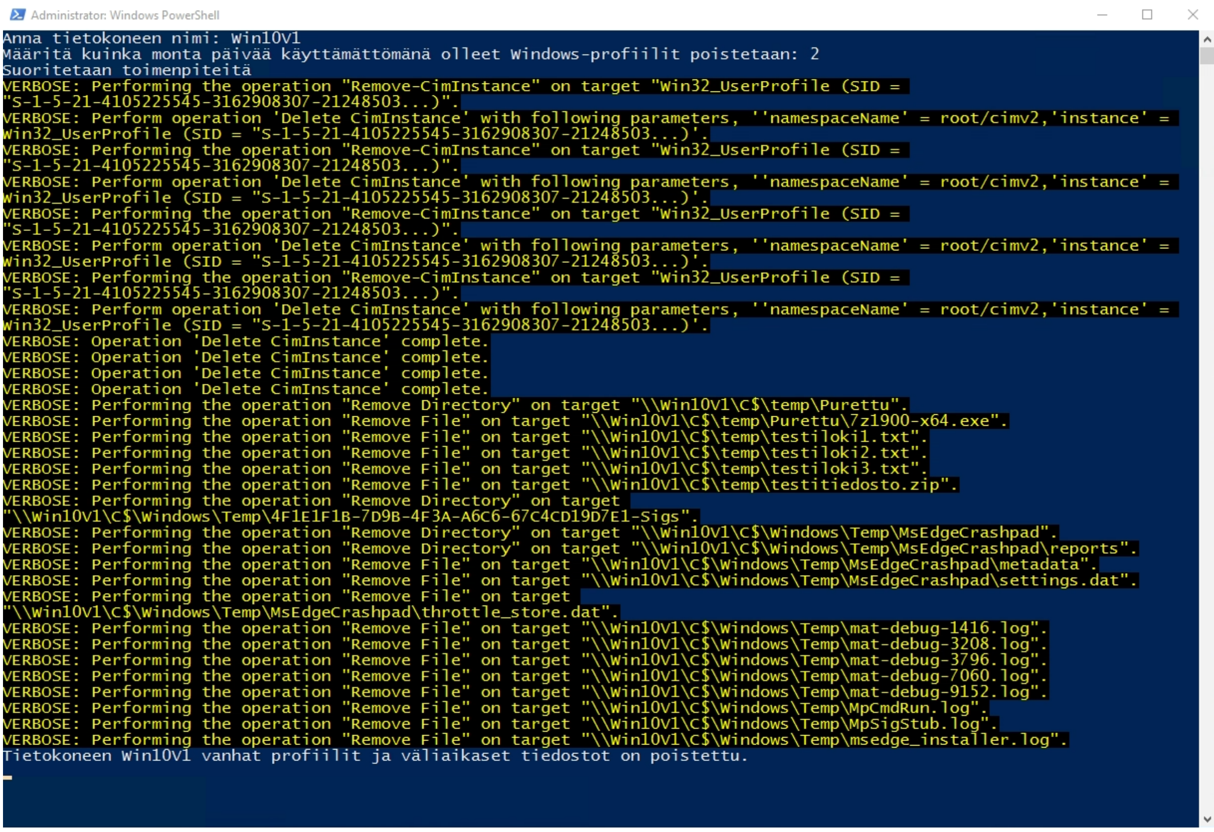
Kuva 19 Tarkistetaan ensin tämänhetkiset Windows-profiilit tietokoneesta Win10V1 ennen skriptin suorittamista



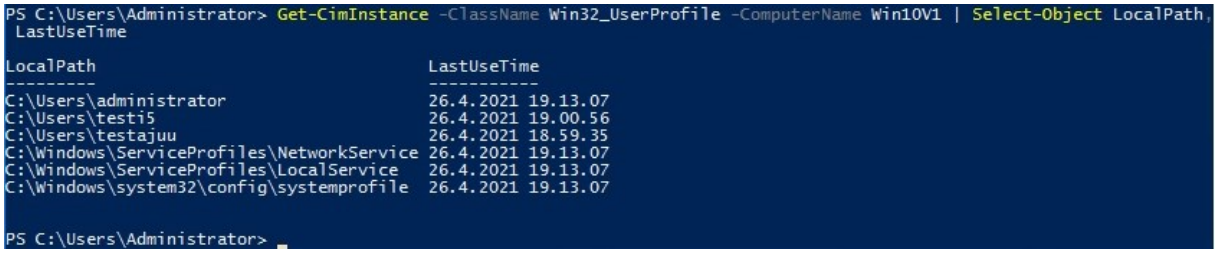
Kuva 20 Skriptiin syötetään ensin tietokoneen nimi sekä aikamääre



Kuva 21 Skripti poistaa valitut profiilit ja väliaikaiset tiedostot C:\temp- ja C:\Windows\Temp-kansioista



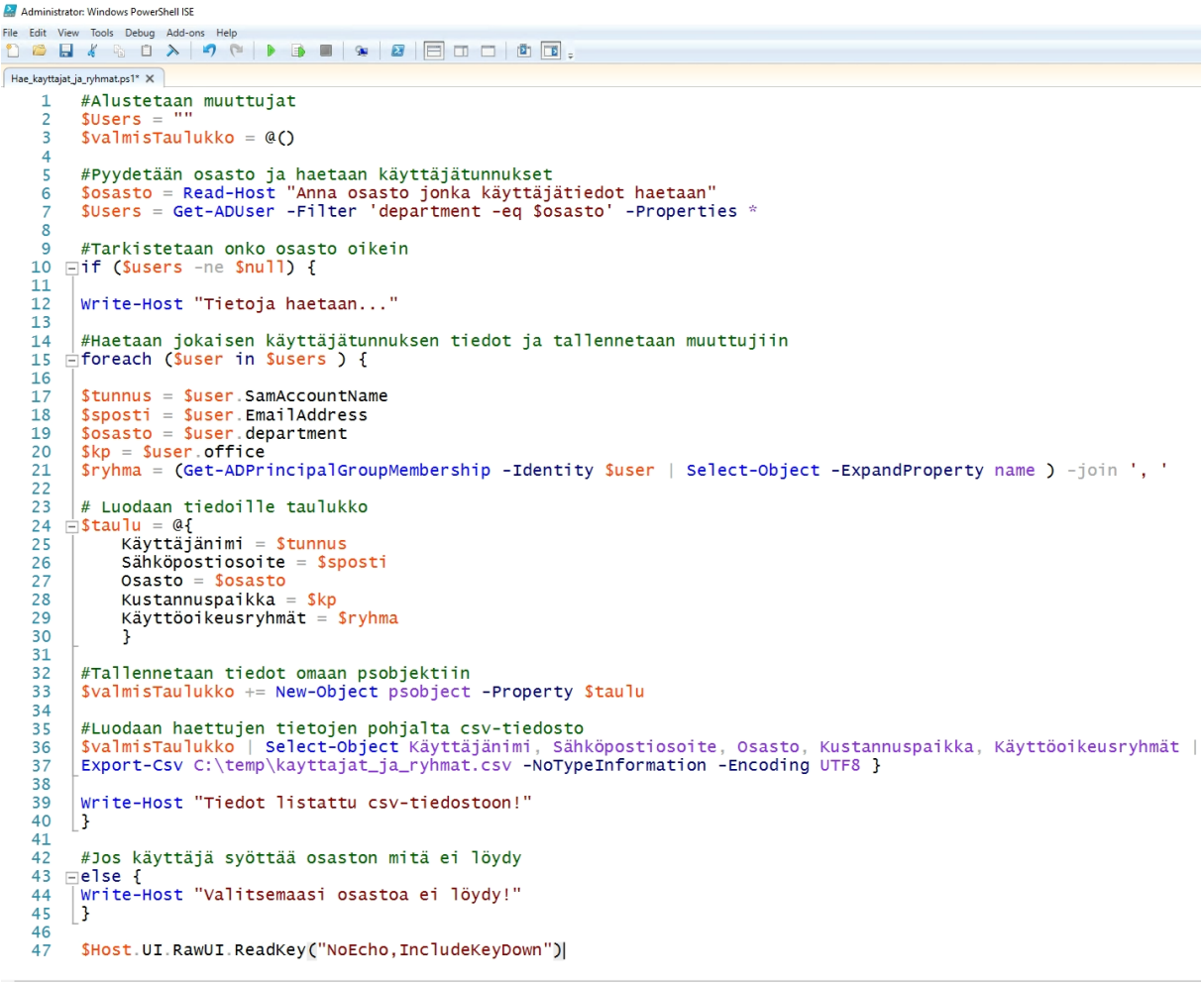
Kuva 22 Varmistetaan, että PowerShell-skripti on poistanut yli kaksi päivää vanhat Windows-profiilit



4.5.3 Käyttöoikeuksien listaaminen

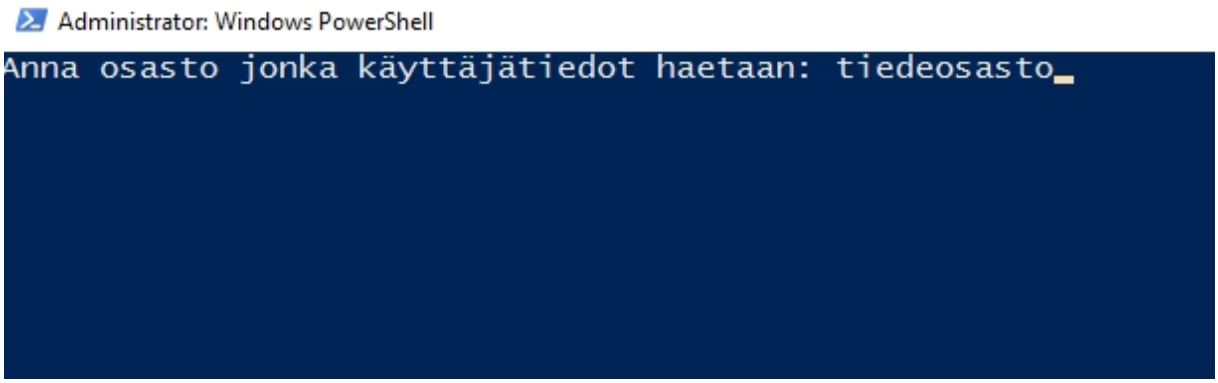
Käyttäjien ja näille liitettyjen käyttöoikeusryhmien listaaminen on pitkällinen prosessi, mikäli tehtävää alkaa suorittamaan manuaalityönä. Työn voi suorittaa manuaalisesti, jos käyttäjiä on muutama, mutta esimerkiksi koko osaston tai organisaation käsittävän listauksen tekemiseen kuluisi liian kauan aikaa. PowerShellin avulla tällaisen listan tekeminen onnistuu, kunhan tietää, millä komennoilla tarvittavat tiedot saa haettua aktiivihakemistosta, ja miten tiedot saa tallennettua haluttuun muotoon. Tässä esimerkissä haetaan tietyn osaston käyttäjien tiedot csv-tiedostoon. Csv-tiedostoon tallennetaan käyttäjätunnus, sähköpostiosoite, osasto, kustannuspaikka sekä käyttöoikeusryhmät, joihin käyttäjän tunnus on liitetty (Ohjelmakoodi 5).

Ohjelmakoodi 6 PowerShell-skripti, joka pyytää osaston nimen ja hakee kyseisen osaston käyttäjät ja näiden käyttöoikeusryhmät

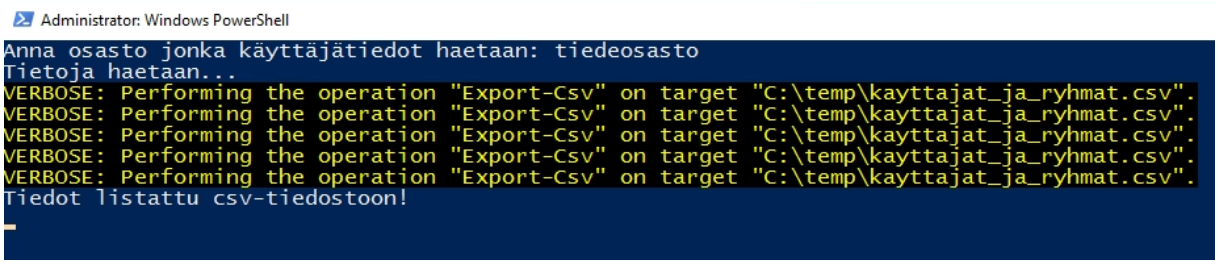


Skripti pyytää käyttäjältä ensin osaston nimen, jonka perusteella käyttäjiä haetaan (Kuva 23). Kun käyttäjä on syöttänyt osaston nimen, skripti tarkistaa löytyykö kyseistä osastoa. Skripti käyttää osastotietona AD-käyttäjien department arvoa. Jos osasto löytyy, skripti hakee kaikki kyseisen osaston käyttäjät ja tallentaa näistä käyttäjänimen (samAccountName), sähköpostiosoitteen (emailAddress), osaston (department), kustannuspaikan (office) sekä AD-ryhmät, joihin käyttäjä kuuluu (Kuva 24). Kun kaikki käyttäjät on käyty läpi, skripti muodostaa haetuista tiedoista csv-tiedoston, joka tallennetaan C:\temp-kansioon (Kuva 25). Csv-tiedostossa oleva tieto kannattaa järjestää omiin soluihinsa, jotta tiedot ovat helposti luettavassa muodossa (Kuva 26). Mikäli käyttäjä kirjoittaa osaston, jota ei löydy miltään käyttäjältä, skripti keskeytyy ja käyttäjälle ilmoitetaan, ettei kyseistä osastoa löydy.

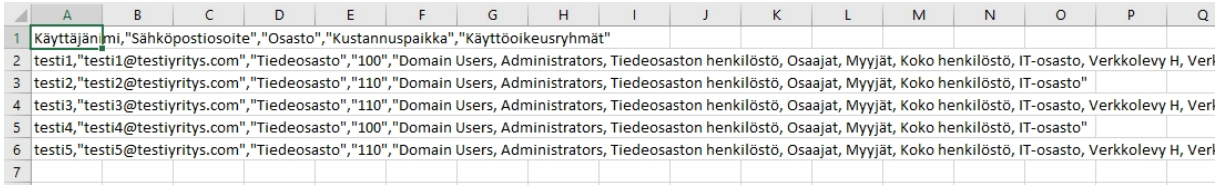
Kuva 23 Skripti pyytää käynnistymisen jälkeen osaston



Kuva 24 Osaston syöttämisen jälkeen skripti hakee tiedot ja lisää ne csv-tiedostoon

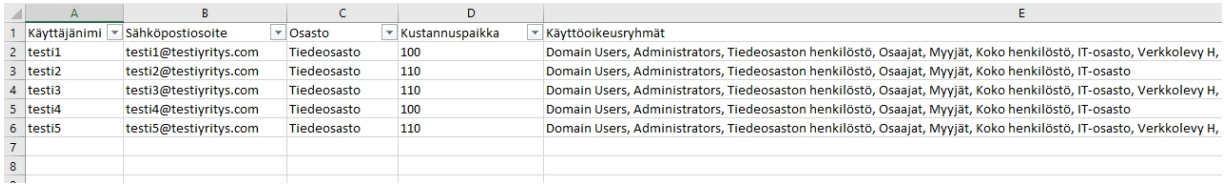


Kuva 25 Skripti luo csv-tiedoston, jossa on lueteltu attribuutit ensimmäiselle riville ja käyttäjätiedot seuraaville riveille



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Käyttäjänimi,"Sähköpostiosoite","Osasto","Kustannuspaikka","Käyttöoikeusryhmät"																
2	testi1,"testi1@testiyritys.com","Tiedeosasto","100","Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto, Verkkolevy H, Verl																
3	testi2,"testi2@testiyritys.com","Tiedeosasto","110","Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto"																
4	testi3,"testi3@testiyritys.com","Tiedeosasto","110","Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto, Verkkolevy H, Verl																
5	testi4,"testi4@testiyritys.com","Tiedeosasto","100","Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto"																
6	testi5,"testi5@testiyritys.com","Tiedeosasto","110","Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto, Verkkolevy H, Verl																
7																	

Kuva 26 Jakamalla tiedot omiin soluihinsa Excelin teksti sarakkeisiin -toiminnolla listauksesta saa helppolukuisen ja tietoja pääsee suodattamaan



	A	B	C	D	E
1	Käyttäjänimi	Sähköpostiosoite	Osasto	Kustannuspaikka	Käyttöoikeusryhmät
2	testi1	testi1@testiyritys.com	Tiedeosasto	100	Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto, Verkkolevy H,
3	testi2	testi2@testiyritys.com	Tiedeosasto	110	Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto
4	testi3	testi3@testiyritys.com	Tiedeosasto	110	Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto, Verkkolevy H,
5	testi4	testi4@testiyritys.com	Tiedeosasto	100	Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto
6	testi5	testi5@testiyritys.com	Tiedeosasto	110	Domain Users, Administrators, Tiedeosaston henkilöstö, Osaajat, Myyjät, Koko henkilöstö, IT-osasto, Verkkolevy H,
7					
8					

4.5.4 Vaihtoehtoiset menetelmät

PowerShell-skripteillä on mahdollista suorittaa melkein mitä tahansa työasemaympäristön tai palvelinympäristön ylläpitotehtäviä. Skriptit voivat olla todella monimutkaisia, ja ne voivat suorittaa useita erilaisia toimenpiteitä halutulla tavalla. Tässä työssä käytettiin esimerkkeinä tiedostojen siirtoa tietokoneilta palvelimelle, Windows-profiilien sekä temp-kansioiden siivousta ja käyttöoikeuksien listaamista. Kaikki mainitut tehtävät pystytään suorittamaan skriptin avulla muutamassa sekunnissa. Mikäli kyseiset toimenpiteet tehtäisiin ilman PowerShellia, kuluisi näiden tekemiseen huomattava määrä aikaa. Tiedostojen kopiointi pitäisi tehdä resurssienhallinnan kautta hakemalla ensin kohdetietokoneelta tiedostot ja siirtämällä ne haluttuun sijaintiin palvelimella. Windows-profiilien poistamista varten pitäisi ottaa erikseen etäyhteys kohdetietokoneeseen jollain etäyhteysohjelmalla. Tähän kuluu paljon aikaa, sillä Windowsiin kirjautuminen vie oman aikansa, minkä lisäksi Windows-profiilien poistaminen graafisen käyttöliittymän kautta on hidasta ja epäkäytännöllistä, mikäli poistettavia profiileja on suuria määriä. Viimeisessä esimerkissä suoritettu käyttäjätietojen sekä käyttöoikeusryhmien hakeminen ja listaaminen erilliseen tiedostoon on manuaalityönä epäkäytännöllinen toteuttaa, mikäli käyttäjiä on suuri määrä.

Kun tulee vastaan tilanne, jossa työn voi suorittaa PowerShell-skriptin avulla tai vaihtoehtoisella tavalla, on järkevää pohtia, pitääkö kyseinen toimenpide suorittaa tulevaisuudessa uudestaan vai onko kyseessä yksittäinen toimenpide. Mikäli kyseessä on yksittäinen toimenpide, voi olla järkevämpää toteuttaa työ manuaalisesti, vaikka siihen kuluisi hieman aikaa. PowerShell-skriptien tekemiseen kuluu myös aikaa, sillä tarvittavia komentoja voi joutua selvittämään pitkään ja tietoa voi joutua hakemaan eri lähteistä. PowerShell-skriptit on myös syytä testata hyvin ennen varsinaista tuotantokäyttöä, sillä väärin määritetyllä komennolla tai skriptillä voi saada merkittävää vahinkoa aikaiseksi. Jos taas kyseistä toimenpidettä joutuu toistamaan useamman kerran, on järkevää tehdä PowerShell-skripti, jonka avulla toimenpiteen pystyy suorittamaan helposti ja nopeasti. Skriptin avulla toimenpiteen voi automatisoida kokonaan, ja mahdolliset muutokset on helppo muuttaa tai lisätä skriptiin. Tämän lisäksi PowerShell-skriptiin voi sisällyttää erilaista lokitietojen keruuta.

5 Johtopäätökset ja pohdinta

Opinnäytetyön tuloksena syntyi opas PowerShellin käyttöön. Opas soveltuu henkilöille, jotka eivät ole käyttäneet PowerShellia aiemmin tai jotka haluavat perehtyä PowerShellin peruskäytäntöihin ja komentoihin. Työn teoriaosassa pyrittiin selvittämään, millainen rooli PowerShellilla on tulevaisuudessa palvelin- ja työasemaympäristöjen ylläpidossa. Kerättyjen aineistojen perusteella PowerShell tulee olemaan yksi tärkeistä komponenteista erilaisten palvelinympäristöjen ylläpidossa. PowerShell tarjoaa mainittuihin teknologian kehityssuuntiin, eli digitaaliseen muutokseen ja pilvipalveluiden yleistymiseen, tehokkaan työkalun, jolla järjestelmien ylläpitäjät pystyvät vastaamaan alan muuttuviin tarpeisiin. Samalla, kun palvelinympäristöt laajentuvat ja palvelimien määrä kasvaa, on tärkeää saada palvelinten ylläpidosta mahdollisimman tehokasta. PowerShellin avulla tämä on mahdollista. Lisäksi PowerShell on tärkeässä osassa automatisoinnin mahdollistajana. Monet Microsoftin tarjoamat hallintatyökalut on rakennettu PowerShell-komentojen päälle, eikä graafisista käyttöliittymistä löydy aina kaikkia samoja toimintoja, joita PowerShell-komennoilla pystyy tekemään.

Tuomalla PowerShellin Windows-alustojen ulkopuolelle, Microsoft laajentaa myös PowerShellin käyttäjäkuntaa ja tekee PowerShellista alustariippumattoman ylläpitotyökalun. Microsoftin siirryttyä avoimen lähdekoodin malliin kehitysyhteisö on päässyt tuomaan oman näkemyksensä ja panostuksensa PowerShellin nykyisiin ja tuleviin versioihin.

Työn teoriaosioon on kerätty laajasti erilaisia komentoja, joiden avulla PowerShellin käytössä pääsee alkuun, ja joiden avulla on mahdollista suoriutua erilaisista service deskissä esiintyvistä työtehtävistä. PowerShell on aiheena hyvin laaja, ja siitä syystä aiheen rajaaminen oli mietittävä tarkkaan. Työssä on onnistuttu keräämään hyvin tarvittavat perustiedot PowerShellin käyttöön. Myös PowerShellin tulevaisuuteen vaikuttavista trendeistä on saatu kerättyä hyviä poimintoja, joiden pohjalle voi maalata tulevaisuudenkuvaa PowerShellille.

Käytännön osassa näytettiin, miten service deskissä esiintyviä työtehtäviä voidaan suorittaa PowerShellilla. PowerShellin tehokkuus tulee esille, kun tehtäviä muutoksia on useita, esimerkiksi tietty asetusmuutos pitää suorittaa useaan eri tietokoneeseen tai

aktiivihakemistoon pitää tehdä laajoja muutoksia. PowerShellin avulla myös tiedonkeruu onnistuu tehokkaasti niin toimialueen tietokoneista kuin aktiivihakemistostakin. Lisäksi valmiiksi tehdyt PowerShell-skriptit ovat tehokas tapa parantaa työtehtävien tuottavuutta. PowerShell tehostaa toistuvien työtehtävien suorittamista ja mahdollistaa työtehtävien automatisointia. Työtä varten pidetyssä haastattelussa kävi ilmi, että PowerShellin käytöllä pystytään tehostamaan service deskin työtä. Ongelmana on kuitenkin usein vähäinen PowerShell-osaaminen service deskin työntekijöiden keskuudessa.

Käytännön osaan on saatu erilaisia esimerkkejä service desk -tehtävistä. Tähän osioon olisi voinut saada esimerkkejä useammasta eri järjestelmästä, kuten Exchangesta tai O365-palvelusta. Näitä palveluja ei ollut mukana pystytetyssä testitoimialueessa, joten ne jäivät työn ulkopuolelle.

6 Yhteenveto

Opinnäytetyön teoriaosassa tutustuttiin PowerShellin komentoihin ja selvitettiin, millainen rooli PowerShellilla on tulevaisuudessa työasema- ja palvelinympäristöjen ylläpidossa. Työhön saatiin kerättyä olennaisia poimintoja IT-alan tulevaisuudennäkymistä ja trendeistä. Näitä tulevaisuudennäkymiä vasten pohdittiin, miten PowerShell pystyy vastaamaan kyseisiin muutoksiin. Tähän liittyvää tutkimusta ja pohdintaa olisi voinut olla vielä hieman enemmän, ja aihetta olisi voinut käsitellä vielä laajemmin.

Käytännön osassa pyrittiin näyttämään, miten PowerShell tehostaa service desk -työtehtäviä. Työhön saatiin kerättyä haastattelun avulla käytännönläheisiä esimerkkitehtäviä, joita voi suorittaa tehokkaasti PowerShellilla, ja joiden suorittaminen ilman PowerShellia veisi huomattavasti enemmän aikaa. Osiossa tehtyä vertailua PowerShellin ja vaihtoehtoisten toimintatapojen välillä olisi voinut käsitellä yksityiskohtaisemminkin.

Opin työn aikana paljon uutta PowerShellista. Olen halunnut opetella käyttämään PowerShellia jo pitkään, koska tiedän, millainen potentiaali sillä on, ja kuinka tehokas työkalu se on oikein käytettynä. Tämä työ antoi minulle mahdollisuuden perehtyä PowerShellin perusteisiin. Pääsin selvittämään myös, miten voin hyödyntää PowerShellia työelämässä service desk -työtehtävissä. Kertyneen tiedon ja osaamisen pohjalle on hyvä lähteä opettelemaan vielä syvällisempiä kokonaisuuksia PowerShellista. Uskon myös, että PowerShell-osaamisesta on minulle hyötyä tulevaisuudessa. Opin työn aikana paljon PowerShellin historiasta ja sain hyvän yleiskuvan siitä, mihin suuntaan PowerShellia ollaan kehittämässä, ja mitkä asiat ajavat tätä kehitystä.

PowerShell on tehokas työkalu työasema- ja palvelinympäristöjen ylläpidossa. Se on kulkenut pitkän matkan paikallisista Windows-ympäristöistä monialustapohjaiseksi tehokäyttäjien ylläpitotyökaluksi. On mielenkiintoista nähdä, millaisen jalansijan PowerShell saa tulevaisuudessa Windows-ympäristöjen ulkopuolella, etenkin erilaisten pilvipalvelujen ylläpitotyökaluna. IT-alan trendit ja globaali pyrkimys tuottavuuden parantamiseen tarjoavat PowerShellille hyvän alustan nousta tärkeäksi eri järjestelmien ja ympäristöjen hallintatyökaluksi.

Lähteet

Aiello, J. (2020). *Announcing PowerShell 7.1 | PowerShell Team*. https://devblogs.microsoft.com/powershell/announcing-powershell-7-1/

AWS. (n.d.). *AWS Tools for PowerShell*. Retrieved March 26, 2021, from https://aws.amazon.com/powershell/

B. McHugh. (n.d.). *Gartner's IT Automation Predictions for 2021*. Retrieved March 26, 2021, from https://www.advsyscon.com/blog/gartner-it-automation/

Dahl, W. (2021). *Ovatko helpdesk ja Service Desk sama asia? | Etevä blogi*. https://www.etevat.fi/blogi/ovatko-helpdesk-ja-service-desk-sama-asia-enta-sitten-itsm

Field Engineer. (n.d.). *Service Desk Analyst | Definition, Job description, Salary*. Retrieved April 10, 2021, from https://www.fieldengineer.com/skills/service-desk-analyst

Fujitsu. (n.d.). *Service Desk Services : Fujitsu Finland*. Retrieved April 10, 2021, from https://www.fujitsu.com/fi/services/digital-workplace/intelligent-services/service-desk/index.html

Google. (n.d.). *Cloud Tools for PowerShell | Google Cloud*. Retrieved March 26, 2021, from https://cloud.google.com/powershell

Hassell, J. (2017). *Learning PowerShell* (1st ed.). De Gruyter. https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=4947050

ITILnews.com. (n.d.). *ITIL Service Support Service Desk Overview | ITILNews.com*. Retrieved April 8, 2021, from https://www.itilnews.com/index.php?pagename=Service_Support_Service_Desk

Jones, D. (2018). *Can We Talk About PowerShell Core 6.0? – PowerShell.org*. https://powershell.org/2018/01/can-we-talk-about-powershell-core-6-0/

Klassen, M. (2018). *What Are the Primary IT Service Desk Responsibilities?* https://www.cherwell.com/it-service-management/library/blog/it-service-desk-responsibilities/

Krause, J. (2016). *Mastering Windows Server 2016*. Packt Publishing. https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=4729372

L. Dignan. (2021). *Top cloud providers in 2021: AWS, Microsoft Azure, and Google Cloud, hybrid, SaaS players*. https://www.zdnet.com/article/the-top-cloud-providers-of-2021-aws-microsoft-azure-google-cloud-hybrid-saas/

Luhtala, T. (2020, December 8). *Nykyaikainen Service Desk automatisoi ratkaisuja ja vähentää käyttäjien kuormitusta*. https://www.cgi.com/fi/fi/blogi/nykyaikainen-service-desk-automatisoi-ratkaisuja-ja-vahentaa-kayttajien-kuormitusta

Microsoft. (n.d.-a). *Get-Help - PowerShell | Microsoft Docs*. Retrieved April 1, 2021, from https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/get-help?view=powershell-7.1

Microsoft. (n.d.-b). *Import-Csv - PowerShell | Microsoft Docs*. Retrieved April 2, 2021, from https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/import-csv?view=powershell-7.1

Microsoft. (n.d.-c). *Invoke-Command PowerShell | Microsoft Docs*. Retrieved May 19, 2021, from https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/invoke-command?view=powershell-7.1&viewFallbackFrom=powershell-7.1.

Microsoft. (n.d.-d). *Read-Host - PowerShell | Microsoft Docs*. Retrieved April 24, 2021, from https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/read-host?view=powershell-5.1

Microsoft. (2017). *Using Tab Expansion - PowerShell | Microsoft Docs*. https://docs.microsoft.com/fi-fi/powershell/scripting/learn/using-tab-expansion?view=powershell-7.1

Microsoft. (2018a). *Introducing the Windows PowerShell ISE - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/windows-powershell/ise/introducing-the-windows-powershell-ise?view=powershell-7.1

Microsoft. (2018b). *What's New in PowerShell Core 6.0 - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-powershell-core-60?view=powershell-7.1

Microsoft. (2018c). *What's New in PowerShell Core 6.1 - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-powershell-core-61?view=powershell-7.1

Microsoft. (2020a). *Cmdlet Overview - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/developer/cmdlet/cmdlet-overview?view=powershell-7.1

Microsoft. (2020b). *PowerShell Core Support Lifecycle - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/powershell-support-

lifecycle?view=powershell-7.1

Microsoft. (2020c). *PowerShell Core Support Lifecycle - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/powershell-support-lifecycle?view=powershell-5.1

Microsoft. (2020d). *PowerShell Remoting Over SSH - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/ssh-remoting-in-powershell-core?view=powershell-7.1

Microsoft. (2020e). *Running Remote Commands - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/running-remote-commands?view=powershell-7.1

Microsoft. (2020f). *What's New in PowerShell 7.0 - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-powershell-70?view=powershell-7.1

Microsoft. (2021a). *Automatic Variables - PowerShell | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_automatic_variables?view=powershell-7.1

Microsoft. (2021b). *Install the Azure Az PowerShell module with PowerShellGet | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/azure/install-az-ps?view=azps-5.6.0

Microsoft. (2021c). *Microsoft Power BI | PowerShell*. https://aka.ms/psgithubbi

Microsoft. (2021d). *What is PowerShell? | Microsoft Docs*. https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.1

Miller, B. (2019). *Azure CLI vs Azure PowerShell - Why not both? · Brett Miller*. https://millerb.co.uk/2019/12/07/Az-CLI-vs-Az-PowerShell.html

Minasi, M. (2014). *3 reasons to learn PowerShell - YouTube*. https://www.youtube.com/watch?v=5m-co2k4dyY

Peters, J.-H., das Neves, D., & Peters, J.-H. (2018). *Learn PowerShell Core 6.0*. Packt Publishing. https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=5477752&query=

R. Stiennon. (2019). *The Three Stages Of Cloud Transformation: Application, Network, Security*. https://www.forbes.com/sites/richardstiennon/2019/02/15/the-three-stages-of-cloud-transformation-application-network-security/?sh=6082d53248ee

Robbins, M. F. (2018). *PowerShell 101 The No-Nonsense Beginner's Guide to PowerShell*. Lean Publishing.

Snover, J. (2018). *PowerShell 2018: State of the Art by Jeffrey Snover - YouTube*. https://www.youtube.com/watch?v=us4HTxtjfa8&t

Liite 1: Aineistonhallintasuunnitelma

Kehitysprojekti:

Kehitysprojektin aikana pidetään päiväkirjaa, johon kerätään teknistä tietoa projektissa käytetyistä PowerShell-komennoista ja -skripteistä. Tämä tieto analysoidaan opinnäytetyötä varten. Kaikki aineisto luodaan Vcommander-palvelussa luotuun testiympäristöön. Ympäristössä ei käsitellä mitään luottamuksellista tietoa. Päiväkirjaa säilytetään tekijän tietokoneen C-asemalla, ja siitä tehdään säännöllisesti varmuuskopioita OneDrive- ja Google Drive -pilvipalveluun. Päiväkirjaa säilytetään C-asemalla ja Google Drive -pilvipalvelussa ainakin vuoden verran opinnäytetyön valmistumisesta.

Kehitysprojektin aikana pidetyistä kokouksista pidetään pöytäkirjoja, jotka säilytetään tekijän tietokoneen C-asemalla sekä OneDrive- ja Google Drive -pilvipalveluissa.

Projektin aikana pidetyn haastattelun tallenne säilytetään työn tekijän tietokoneen C-asemalla sekä työpaikan OneDrive-palvelussa.