TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Riivo Kiljak 178071IABM

POWERSHELL SCRIPTING LANGUAGE COURSE PROPOSAL FOR TALLINN UNIVERSITY OF TECHNOLOGY

Master's thesis

Supervisor: Siim Vene
MSc

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Riivo Kiljak 178071IABM

POWERSHELLI SKRIPTIMISKEELE KURSUSE ETTEPANEK TALLINNA TEHNIKAÜLIKOOLILE

Magistritöö

Juhendaja: Siim Vene
MSc

Tallinn 2019

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Riivo Kiljak

07.05.2019

Abstract

In the thesis, a recommendation is made to establish a new course at TalTech. The course is intended to teach the *PowerShell* scripting language to students, most importantly in the *IT Systems Administration* programme.

Course material is proposed in the form of *lecture slides*, *home assignments* and *knowledge tests*. All three of which are available in the appendices of the paper. *Design science* is used to pass iterations of improving the content prior the the paper publishing. Academic literature is analysed to determine the included and excluded topics and the teaching methodology. Moreover, input is acquired from scrutinising public information on Microsoft's official PowerShell courses and interviewing subject matter experts who use PowerShell at local companies.

The course material is provided written in LaTeX which means that it can be conveniently modified, version controlled and distributed in the PDF format. Although the proposed course is seen as an online course hosted on Moodle, argumentation is made suggesting a combination with classroom seminars is likely to result in better learning outcomes at the cost of scalability.

In the final chapter, an analysis regarding the validity of the produced materials is made. Moreover, a list of suggestions on how the course could be developed further in the future is put forward.

This thesis is written in English and on 282 pages, including 5 chapters, 4 figures and 3 tables.

Annotatsioon

Kuigi PowerShell on muutunud populaarseks skriptimiskeeleks, mida kasutatakse IT-taristu haldamisel, puudub hetkel TalTechis kursus, kus seda keelt õpetatakse. Lõputöös pakutakse välja kursuse loomine ning esialgsed materjalid, mille abil kursust läbi viia. Eeskätt sobiks kursus IT süsteemiadministreerimise õppekavasse.

Välja pakutavateks materjalideks on loengute slaidid, ülesanded iseseisvaks lahendamiseks ja teadmistekontrollid. Materjalide loomisel kasutatakse tsüklilist lähenemist, kus eesmärk on sisu järkjärgult parendada.

Kursuse sisu loomisel lähtutakse uue e-kursuse lisamist TalTechi Moodle keskkonda. Siiski ei ole lõputöö avaldamise ajaks videoloengud salvestatud. Töös märgitakse, et kuigi täielik e-kursus võimaldab tõhusat koolitamist, on klassiruumides seminaride pidamisel tõenäoline eelis paremate õpitulemuste näol.

Sisend materjalide koostamisse tuleb peamiselt kolmest allikast. Esiteks analüüsitakse Microsofti ametlikke PowerShelli kursuseid. Teiseks viiakse läbi intervjuud praktikutega kohalikes ettevõtetes, kes kasutavad igapäevaselt PowerShelli. Viimaks on sisu mõjutatud ka autori kogemustest ja hinnangutest.

Lõputöö väljundi elemendid on tekstifailid LaTeX keeles, mida on mugav muuta, hallata versioonihaldustarkvaraga ja jagada PDF-failide kuju. Siiski eeldab materjalide kasutamine kursuse läbiviijalt eelnevat kogemust, sest LaTeX ei ole intuitiivne, nagu näiteks PowerPoint.

Kursus jaguneb kaheksaks mooduliks. See on loodud katma õpetatava skriptimiskeele põhifunktsionaalsuse ja Microsofti toodete nagu Windows, Active Directory ning Hyper-V haldamise peamised käsud.

Kokkuvõttele eelnevas peatükis arutletakse töö valiidsuse üle. Ühtlasi analüüsitakse erinevaid võimalusi kursuse arendamiseks tulevikus. Näiteks soovitatakse lisada kursusesse grupitööd, mis hetkel puuduvad. Samuti antakse nõu, et kursusesse võiks hiljem kaasata pilvetehnoloogiate haldamise.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 282 leheküljel, 5 peatükki, 4 joonist, 3 tabelit.

List of Abbreviations and Terms

Active Directory	Microsoft developed directory service for Windows domain networks
API	Application programming interface
CIM	Common Information Model
Cmdlet	A lightweight command used in the Windows PowerShell environment
CPU	Central processing unit
ECTS	European credit transfer and accumulation system
GUI	Graphical user interface
HDD	Hard disk drive
HITSA	Information Technology Foundation for Education
Hyper-V	Microsoft developed hypervisor
Hypervisor	Computer software that creates and runs virtual machines
ICT	Information and communications technology
ISE	Integrated scripting environment
ITIL	A set of practices for IT service management
ITL	Estonian Association of Information Technology and Communications
LaTeX	A document preparation system where the author uses plain text
MOC	Microsoft official course
OS	Operating system
RAM	Random-access memory
REST	Representational state transfer
TalTech	Tallinn University of Technology (formerly abbreviated TUT)
UNIX	a family of operating systems that derive from AT&T Unix
VM	Virtual machine
WMI	Windows Management Instrumentation

Table of Contents

List of Figures

List of Tables

1 Introduction

PowerShell has gained significant prevalence as a tool for infrastructure administration (see, for example, Buyya & Barreto, 2015; Jayaseelan & Charles, 2014; Palumbo, 2017). Yet there is currently no course at TalTech (formerly abbreviated TUT for Tallinn University of Technology) which focuses on teaching its students the syntax and the functionality of this scripting language specifically. After the merger with the former IT College, there is a course "Scripting Languages" in the university's curriculum which however concentrates on Bash and Python (TalTech, 2019).

The goal of the paper is to propose a new course with relevant teaching material to TalTech for delivering the course foremost to students in the *IT Systems Administration* programme. The course may be optional or an elective and is intended to be free for enrolling to all interested students without prerequisite courses.

PowerShell offers an alternative to traditional infrastructure management via GUI tools and allows for simple and automated management of large environments with many nodes such as PCs and servers. Since August 2018, PowerShell is open-source and available on GitHub for Windows, Linux and macOS operating systems (GitHub, 2019b).

Microsoft provides PowersShell *cmdlets* (pronounced "command-lets"), for example, to the Windows operating system, Active Directory services, Exchange e-mail service, Hyper-V virtual machine host and Azure cloud computing service. The language also provides generic tools for object oriented data manipulation such as various calculations, filtering, string analysis, etc. (Microsoft, 2019d). Moreover, several third party providers have created custom PowerShell modules for enabling automated management of their products (e.g. Amazon Web Services, 2019; Veritas, 2019).

Important requirements for the thesis' output are scalability and applicability. The first means in the given context that the number of students in the course may vary while the learning outcomes must stay consistent. In order to achieve the goal, the course content must rely on automated exercise environments and quizzes. Although the course will be proposed with some focus on the needs of TalTech and the Estonian employers, the outcome is assumed be applicable in other cases as well, i.e. by other universities or teaching centres.

Figure 1. Occupied posts in the Estonian ICT sector (Statistics Estonia, 2019)

1.1 Problem Statement

The need for graduate students with scripting skills can be shown via various sources. Firstly, there is academic literature where scripting and automation are commended and seen integral to contemporary computing. Automation is seen as resulting in both improved cost effectiveness and more flexible yet functional information systems (see, for example, Ashraf, 2015; Buyya & Barreto, 2015; Jayaseelan & Charles, 2014; Palmer, 2015).

Secondly, although the OSKA report from 2018 does not mention neither scripting nor PowerShell specifically, it highlights an acute need for additional IT specialists on the Estonian labour market (Sihtasutus Kutsekoda, 2018). The report is published regularly and it gives an overview of of the competences employers expect from employees in Estonia. The latest report emphasised that ICT specialists are needed in all industries, not only in the ICT industry. Additionally, Figure 1 shows a clear trend for a growing number of people working in the ICT sector. Granted, it is an indirect measure of the need for employees with PowerShell scripting skills.

The OSKA report is analysed among others by the Estonian Association of Information Technology and Telecommunications (officially abbreviated as ITL) members who represent employers in meetings with university decision makers. The report can therefore be considered as indirect input to universities' curricula. Moreover, there is a note made in a meeting protocol where a representative of ITL specifically mentions the need for PowerShell educated students (Vene, 2018).

The argumentation above outlines the demand for the thesis. A case could be made that alternative courses with suitable content exist already and that the majority of them are available

online free of charge.

The need for new course material can further be justified by the form of the materials. Namely, the goal of the thesis is to create a new Moodle course. Moodle is a community driven open-source learning platform (Moodle, 2019). It is also the platform used at TalTech and other Estonian universities. Although, Estonian schools are encouraged to use the Moodle environment provided by the Information Technology Foundation for Education (HITSA)[1] (Information Technology Foundation for Education, 2019), TalTech has launched its own Moodle[2].

All of the proposed course material is intended for hosting on Moodle. Furthermore, part of the course material will be tests designed for the Moodle platform. This contributes to the uniqueness of the output.

Additionally, and even more importantly, the material put forward in the thesis will focus also on evaluating students. The aspect of assessing the learning outcome through theoretical and practical quizzes and lab assignments is missing in the publicly available PowerShell courses.

Finally, the teaching material will be made unique by combining ideas from different existing courses with author's own work experience. The proposed course will draw content official Microsoft PowerShell courses, subject matter experts' recommendations and author's personal experience.

TalTech observes its reference universities Aalto University in Helsinki, KTH Royal Institute of Technology in Stockholm and Chalmers University of Technology in Gothenburg. The named institutions did not have a specifically PowerShell related course in their curricula, yet the language is potentially taught in generic scripting courses.

1.2 Scope

The following section defines the scope of the paper and lists aspects on one hand included and on the other hand deliberately excluded from the thesis.

The main focus of the paper is on creating teaching materials for the course which can be divided into different categories. Firstly, this includes slides with notes for classroom presentations or video lectures. Secondly, the materials contain lab assignments which are meant to be performed individually but can be done under supervising or in student groups. Thirdly, the output holds tools for student evaluation. These components of the materials will be compiled into a Moodle course.

[1] Hosted at https://moodle.hitsa.ee/

[2] Hosted at https://moodle.taltech.ee/

Moreover, the validation of the output material is within the scope of the thesis and constitutes a significant part of it. How validation is done is discussed in-depth in section 2.3 and validation is carried our in Chapter 4.

Nevertheless, it is important to note that the output of the thesis cannot be considered a final product. The content is meant to be improved through iterations and feedback. The latter will be given by employers and students. Only a limited number of iterations can be passed before the paper in hand is published. Hence the tweaking and perfecting the materials is out of scope of the thesis.

It is in fact optimal to keep a certain level of flexibility in the course as newer versions of PowerShell might add new and relevant features. Also, IT market standards and demands are likely to change which is likely to affect the optimal content.

The course materials will be in English. Should there be no non-Estonian speaking students in seminars, the instructor can choose the spoken teaching language.

Although one of the principles for the course was scalability to a varying amount of students, videos are required to fulfil the requirement. Notwithstanding, videos are out of scope for the thesis. It is suggested that the first semester seminars, when the course is delivered, are recorded and the recordings are published on Moodle either on the same or following semesters.

Finally, the lab exercises assume that a virtualised test environment exists at the university. Requirements for the environment are described in section 4.2.1, but the detailed steps of setting it up are knowingly excluded from the thesis.

1.3 Structure

The thesis will next introduce the methodology used in creating and evaluating the course material. The methodology covers the principles of course material compilation, teaching approaches and validation of the output.

The majority of the paper will focus on creating and presenting course materials, such as presentations, exercises and tests, but also on analysing the validity of the results. The latter will be done through trialling the tests and lab assignments on students and requesting feedback on the materials from subject matter experts.

2 Methodology

The methodology presents the contemporary situation of academic literature related to the topic and also the approaches used to tackle the defined problem statement. In the case of the paper in hand, the methodology is threefold. Firstly, the general topics of the course materials must be determined by analysing the existing courses and the expectations of local IT infrastructure teams for new team members.

Secondly, the course content and learning methods need to adhere to effective teaching practices. Since the course will be a combination of classroom seminars and independent homework, the relevant approaches need to be analysed and taken into account when creating the content.

Finally, it is imperative that the output of the thesis is validated. The course content will be evaluated by IT infrastructure administrators. Moreover, the lab assignments and quizzes must be trialled in order to confirm the appropriate level of difficulty for undergraduate students.

2.1 Course Material Principles

The following will explain and justify how the course material is compiled. Design science is used to manage the iterations through which the material is created and improved. Additionally, included and excluded themes and topics are discussed.

2.1.1 Design Science

The output of the thesis is created by using the design science approach. It may be opposed to behavioural science. Design science is "fundamentally a problem solving paradigm[; it] seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished" (Hevner, March, Park, & Ram, 2004, p. 76). "The goal of behavioural science research is truth [and the] goal of design science research is utility" (Hevner et al., 2004, p. 80).

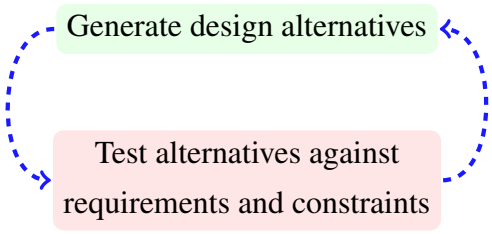


Figure 2. The generate/test cycle. Adapted from Hevner, March, Park, and Ram, 2004, p. 89

The proposed teaching materials are seen as IT artefacts. Hevner et al., 2004, p. 77 states that "IT artefacts are broadly defined as *constructs* (vocabulary and symbols), *models* (abstractions and representations), *methods* (algorithms and practices), and *instantiations* (implemented and prototype systems)" (emphases added). Artefacts address problems which need solving (Marx, Mayer, & Winter, 2012, 26:2). The paper in hand focuses on methods and instantiations.

The slides and the lab assignments can be seen as instantiations whereas the suggested teaching approaches are in essence methods. Artefacts are built and then evaluated in terms of the utility they provide in delivering the course.

Nunamaker and Briggs, 2012, p. 20:10 argue that the definition of an IT artefact is diluted, i.e. its meaning has become so ambiguous that it does not add value. They emphasise the *proof-of-use* over *proof-of-concept* and even *proof-of-value*. In other words, "[p]roof-of-use prototypes [must be] sufficiently robust to be left with users in the field to support real tasks in the workplace" (Nunamaker & Briggs, 2012, p. 20:6).

In the context of the proposed PowerShell course, the proof-of-use would be positive feedback from the students and employers that the knowledge gained from the course could be successfully applied. The possibilities of gaining proof-of-use before the paper is published are limited. The actual usefulness of the course can be evaluated only after it has been delivered for at least one semester.

Figure 2 presents a simple process to illustrate how the generate/test cycle works. It emphasises the iteration between generating news designs and trialling them in practice. In the context of the proposed PowerShell course, it means making small changes to the course content and teaching methods over time. It is possible to pass through only a small number of iterations before the thesis is disclosed.

Carlsson, Henningsson, Hrastinski, and Keller, 2009 wrote a paper on designing management support systems. Although the process of designing structure and materials for a course at a university has significant differences, fundamental principles are still applicable. The process starts by defining the situation and finding support. The second part is data collection. Gath-

ered data is then fused into specifications which are analogous to the course module listing. The third step is development, i.e. creating the course content. Two final steps are implementation and evaluation.

In the thesis, support in the first step for creating the course needs to be obtained from TalTech. Data collection means research into topics that could be included in the course. The specifications are compiled into lessons — also called course modules — that constitute the course (see Table 3 in Chapter 3). The content development is done and explained in Chapter 3. Finally, the evaluation and validation are done in Chapter 4.

Meyer, Rensing, and Steinmetz, 2011 wrote an article emphasising the value of reusing learning resources. It is also important in the context of the thesis that the output may be used outside the proposed course. The produced artefacts may be modified to fit other purposes and parts of them used in other courses.

2.1.2 Course Tailored for Students

As stated in the Introduction, the course content will be unique as it will be created with the specifics of a university course in mind. The differences from Microsoft Official Courses (officially abbreviated MOC) and PowerShell courses available online will be substantial and deserve to be clarified.

Firstly, the content is novel. There are two PowerShell MOCs: *Course 10961C: Automating Administration With Windows PowerShell* and *Course 10962C: Advanced Automated Administration With Windows PowerShell* (Microsoft, 2019a, 2019b). The courses focus on the core functionalities of the language. The latter course goes in-depth with some of the functionalities.

Nevertheless, almost no attention is paid to using the language's modules to manage products such as Active Directory, Exchange, Hyper-V, Azure, etc. Author's personal experience shows and feedback from the local employers identified that students on the labour market would benefit from knowing the basics of managing the listed products via PowerShell.

Secondly, the courses from Microsoft and various other online resources do not provide methods for evaluating students learning outcomes. The output of the thesis will also be tests that can be used to confirm that the students matriculated in the course mastered the content at a sufficient level.

Next, there is a need to create new course material in order to avoid plagiarism and copyright infringement. Using third party materials without their permission or without paying royalty fees could result in penalties and reputation damage to TalTech.

Finally, although it is emphasised throughout the paper that the infrastructure structure needed for the course is out of scope, the paper still explains the basics of creating virtual

machines that can be used in lab assignments. In case of MOCs, the environment preparation is fully proprietary information.

2.1.3 Included Skills and Themes

The main motivation for the course is to familiarise students with the concept of automation and provide them with skills to implement automated tasks. The concept of automation is emphasised in many contemporary books and IT frameworks. For example, Limoncelli, Hogan, and Chalup, 2016, pp. 37–39 use the terms *pets* and *cattle* to illustrate servers which on one hand could be customised and need special attention (*pets*) or on the other hand they could be similar to each other and easy to manage in bulk (*cattle*).

In February 2019, ITIL 4 was launched. ITIL is an approach to IT service management. The latest version shifts emphasis to automation which is also a fundamental concept in Agile, DevOps and Lean methodologies (AXELOS, 2019). This paradigm change corresponds well with the purpose and capabilities of the PowerShell scripting language.

A balance must be found between teaching the students cmdlets that they are expected to remember and giving the students the competence of using the help documentation. A combination of both is most likely needed to solve real life situation that the students will encounter during their careers.

The overarching aim of the materials is to include both theoretical and practical aspects of PowerShell, with a focus on practical knowledge and gaining hands-on experience. The integration of knowing and doing is also a cornerstone of the project-based learning (PBL) approach. PBL allows "to solve authentic problems and produce results that matter (Markham, 2011, p. 38)." Granted, PBL works best in team work scenarios which were are not implemented in the proposed course at the current status.

Theory is presented via classroom seminars, or later via video lectures, which will be followed by practical lab assignments. Each student will need to carry out task pertaining to the learned content within each course module.

The goal is to teach *threshold concepts* which are defined as "as core *gateway* concepts that unlock new, previously inaccessible knowledge" (Yeomans, Zschaler, & Coate, 2019, p. 23:3). As a result, students should have enough knowledge to continue learning on their own by referring to more advanced examples and PowerShell official documentation online.

To a certain extent, the creation of the course materials is seen as a (software) development process. The output is code in the LaTeX language uploaded to a Git repository with the purpose of solving a real life problem. Hence, software engineering research methods can be seen applicable as a method for developing the content.

Table 1. Course 10961C: Automating administration with Windows PowerShell

Module	**Name**
Module 1	Getting started with Windows PowerShell
Module 2	Cmdlets for administration
Module 3	Working with the Windows PowerShell pipeline
Module 4	Understanding how the pipeline works
Module 5	Using PSProviders and PSDrives
Module 6	Querying system information by using WMI and CIM
Module 7	Working with variables, arrays, and hash tables
Module 8	Basic scripting
Module 9	Advanced scripting
Module 10	Administering Remote Computers
Module 11	Using background jobs and scheduled jobs
Module 12	Using advanced Windows PowerShell techniques

Basili et al., 2018, p. 44 foster *context-driven research*. They define the term as "research focused on problems driven by concrete needs in specific domains and development projects". Therefore, the approach underscores the practical use of development output. Opposing argumentation could be made that context-driven research produces too specific and non-generic solutions (Basili et al., 2018, p. 45). This should not be the case with the proposed PowerShell course because other universities or teaching centres can customise the materials according to their needs.

Microsoft provides two official courses on PowerShell. Their modules are listed on Table 1 and on Table 2. The module listing and their limited explanations of the courses are scrutinised and used as the basis for the proposed course. Elements from the majority of listed modules are also included in the output of the thesis.

The first course is a five-day course and designed to provide "students with the fundamental knowledge and skills to use Windows PowerShell for administering and automating administration of Windows servers" (Microsoft, 2019a). This objective corresponds loosely to the first lessons in the proposed TalTech course.

The second course focuses on "core scripting skills such as creating advanced functions, writing controller scripts, and handling script errors" (Microsoft, 2019b). Most of these topics will be covered in the final modules of the thesis output.

The precise materials and lab assignments that Microsoft certified training centres use are proprietary and cannot be perused without enrolling to the course. Moreover, the content may

Table 2. Course 10962C: Advanced automated administration with Windows PowerShell

Module	**Name**
Module 1	Creating advanced functions
Module 2	Using Microsoft .NET Framework and REST API in PowerShell
Module 3	Writing controller scripts
Module 4	Handling script errors
Module 5	Using XML, JSON, and custom-formatted data
Module 6	Enhancing server management with DSC and JEA
Module 7	Analysing and debugging scripts
Module 8	Understanding Windows PowerShell Workflow

not be copied or reproduced without Microsoft's permission. It is nevertheless not forbidden to create new content which may approximately follows the same topics.

Academic literature on the topic was found to be scarce. The existing papers were mostly conference proceedings and discussed PowerShell in contexts too advanced for the current introductory course planning. Nevertheless, Palumbo, 2017 listed four issues that he was able to solve using PowerShell at the Lehigh University in the US which are suitable for gathering inspiration. The first one regarded managing a Mozilla Firefox add-on, namely Adblock Plus. He solved the problem by writing two PowerShell scripts. The first one analysed the files in users' profiles and their content and the second one installed the add-on if needed.

The next problem that the author solved pertained to Windows license activation. There were computers in the environment which lost their connection to the Key Management Service server and needed re-activation. A PowerShell script helped to automatically re-register and validate the license decreasing the number of help desk calls (Palumbo, 2017, p. 8).

The third issue was with cabinet and log files that filled users' hard disks. The remedying script deleted the files matching a location and name expression. It also reported the available storage size before and after running the script to a central server.

The final use case where he applied a PowerShell script solution was about Java installation and update. The standard way of updating was not an option as the update required all browser windows to be closed and forcefully closing them could result in data loss. The script installed and configured Java according to the university's standards without interrupting users' work (Palumbo, 2017, p. 9).

All these four examples are real life scenarios which can be solved by PowerShell scripts. The difficulty level of the scripts is reasonably low and the use cases are suitable for presenting in the course proposed in the thesis. The scenarios are used as inspiration for creating lab

assignments.

Another feature of PowerShell that was mentioned in academic literature was Hyper-V management. More specifically, using PowerShell to set up a cost efficient Hyper-V disaster recovery solution was described by Jayaseelan and Charles, 2014. The article narrated a real life scenario where virtual machines are replicated to a physically different location and they can be recovered from it in case of an emergency. This is an argument for including Hyper-V into the proposed course.

The last input source for the course content is interviews with subject matter experts or practitioners. An interview was conducted with a team lead at Swedbank Estonia and it pointed out several aspects that the course should cover in its modules (Swedbank, 2019). For example, the interviewee suggested to include slides about access control lists of objects and code signing options. Moreover, the interviewee agreed on the importance of teaching Active Directory administration in the course. Yet some of the suggestions, such as Active Directory organisational unit delegation and group policy object management, were excluded from the course content, since they were seen as too advanced for the intended course level.

The second consultancy was done with a PowerShell practitioner and trainer at Telia Estonia. As apparent from the discussion, the company expects candidates to scripting jobs to be capable of perusing cmdlets' manuals and investigating cmdlets' functionalities independently (Telia, 2019). There is a myriad of different modules and cmdlets and it would be unrealistic to remember even a fraction of them. Therefore, the company representative underscored that the proposed course should prepare students in finding self-help with cmdlets such as `Get-Help`, `Get-Command`, `Get-Module` and `Get-Member`.

Finally, an interview with the thesis' supervisor showed that Hyper-V and potentially cloud platform Azure should be included as a separate module (Vene, 2019). There is a trend from on-premise platforms toward cloud-based platforms and the school intends to teach modern technologies to its students. While Hyper-V was included, Azure was left out due to course length limitations and challenges of providing a lab environment for Azure. It could be included into the course in later iterations, as discussed in section 4.4.

In summary, the proposed course is made up of expanded MOC content, author's personal experience and input from employers on the local labour market. The limitations of these sources are discussed in section 4.1.

2.1.4 Excluded Advanced Use Cases

As stated above, academic literature pertaining PowerShell usefulness to solve real life problems was sparse. Most of the found articles discussed PowerShell in aspects that are not suitable

with the level of an introductory course. Some found examples are presented below.

PowerShell is increasingly used by threat actors. Anti-virus software has developed and is making it increasingly difficult for attackers to run their malicious code. Consequently, they have started to exploit tools which already exist on the target machines, such as PowerShell (Liu, Xia, Yu, & Liu, 2018, p. 825). PowerShell is an attractive vector for attackers since it is pre-installed and enabled on the majority of Windows machines (Hendler, Kels, & Rubin, 2018, p. 188).

Surprisingly, Wueest, 2016 found that in fact 95.4% of the scripts that were using PowerShell were deemed as malicious. Yet it is difficult to separate malicious scripts from benign ones (Tsuda et al., 2018). Identifying harmful code can be done only via advanced techniques. Rusak, Al-Dujaili, and O'Reilly, 2018 and Hendler et al., 2018 show how deep learning can be used to detect malicious PowerShell commands.

Barakat and Hadi, 2016 published an article showing how PowerShell can be used as a forensics tool. As the dependence on digital devices is increasing, so is the exposure to cyber threats and cyber exploitation (Choo, Fei, Xiang, & Yu, 2016). Barakat and Hadi, 2016 demonstrate how PowerShell can be used in forensics to detect and extract evidences.

PowerShell as a forensics tool is limited to the Windows operating system and relies on six categories of Windows artefacts (Barakat & Hadi, 2016, pp. 41–42):

1. General machine information
2. User account activity
3. File/folder activity
4. Installed programs
5. Internet activity
6. External storage devices

A tool named PowerForensics simplifies using PowerShell for digital forensics (GitHub, 2019a) but the process requires "a deep knowledge of the investigator in the technology used in order to be able to collect, acquire and [analyse] digital devices and systems in the right and effective manner" (Barakat & Hadi, 2016, p. 41).

Another advanced use case in the academic context was narrated by Jordan, Patten, Peterson, and Sellers, 2016 who used PowerShell functionality to create a load generator. More precisely, the authors created a distributed load generator for dynamically generating network traffic.

After analysing the available tools for load generating, the authors concluded that they needed to develop a new set of scripts. PowerShell was the chosen language. The developed script could be distributed on multiple clients and it would effectively test the capabilities of the target Active Directory Domain Services servers.

The named articles show what can be done with PowerShell, if its features are fully utilised. Yet the level of difficulty exceeds that of the one intended in the planned course. Consequently, all these and other similar advanced used cases are excluded from the course slides and lab assignments.

2.2 Teaching Methodologies

The approach taken to teaching will have a clear impact on the learning outcome for the students. Bhagyavati, 2006, p. 2 points out that the different learning styles that students have make it challenging to deliver an effective online course. The same author lists six principles to keep in mind when delivering an online course:

1. The syllabus should be detailed and contain a roadmap with specific deadline dates and expectations
2. Lectures should be provided in various formats to cater the needs of different students, e.g. slides for visual learners and video lectures for auditory students
3. Encourage discussion, both instructor-to-student and student-to-student
4. Assignments should be given frequently to encourage incremental learning
5. The course should end with an overarching learning evaluation
6. Periodic and final feedback from students allow the instructor to develop the course

Bhagyavati, 2006, p. 5 also underscores that IT learning is most efficient with hands-on activities. The above justifies focus on examples on the proposed PowerShell course slides and encouraging the students to try the sample code and its variations on their own during the seminars and self-studying.

Deadlines noted in the first point are also highlighted by Saltz and Heckman, 2018. The authors recommend creating a visual guide for scheduling assignments. A Kanban board in Moodle would give the students a clear overview of the upcoming assignment submission dates.

The third point of encouraging discussion is a non-trivial task. One way of initiating discussion is to publish the different solutions that students proposed to home assignments. The lab

assignments have different answers each with pros and cons. An article by Silva and Moreira, 2003 points out how peer-review fosters constructive debate and improves learning outcomes. Moodle offers a forum platform for this purpose that should be used in the proposed PowerShell course.

The fourth point is controversial as frequent assignments decrease the flexibility that the students have in planning their own time. The possibility to make a personal schedule is considered an advantage of online courses. However, research shows that giving students flexibility in assignments leads to last minute mass submissions and decreased learning outcomes (summarised in Isomöttönen & Tirronen, 2016, p. 1:30).

The peer review of code could be accompanied by automatic evaluation of submissions. Douce, Livingstone, and Orwell, 2005 lists three generations of assessment systems. The earliest examples of the first generation date back to 1960s and allowed the instructor to quickly validate students' answers. The system merely output if the input code was right or wrong. The second generation added the ability to check if the programming style has been applied sensibly. The third generation are web-based and more complex by providing a GUI for assessment and feedback.

A first generation tool would suffice initially for the purposes of the planned course. Code styling evaluation can be provided by peer-review and instructor feedback on Moodle.

Bhavnani, Peck, and Reif, 2008 wrote a paper on the effective and efficient use of learned content. They describe cases where students are able to carry out average difficulty tasks yet through intricate and inefficient methods. There is a risk of the same occurring in the context of the PowerShell course as well. Therefore, it is important that a strategic approach is used which means that the instructor explains when various methods are most efficient. Moreover, students need to be encouraged to be critical towards their own code and find ways of optimising code even when it already has the intended functionality.

Kunkle and Allen, 2016 categorise teaching programming into three categories: 1) objects-first, 2) objects-early and 3) imperative first approach. The difference comes from how soon in the course objects are introduced. Since PowerShell is not an object-oriented language, it is seen as most optimal to adopt the imperative-first approach.

Moreover, the goal is to apply problem-based learning (like project-based learning, also abbreviated PBL) in the course. It is defined as an approach that "makes the student the focus of the learning process, seeking to empower them such that they take responsibility themselves for their own learning" (O'Grady, 2012). It would be ideal to raise curiosity for the topic in students and make them find creative solutions to scripting problems in PowerShell which go beyond the content of the course.

Teo and Gay, 2006, p. 2 see probable problems with e-learning in scenarios where the content is not intuitive enough to stand on its own. Therefore, it is advantageous to trial the materials prior to delivering the course to get feedback on potentially unclear slides and assignments.

2.3 Validation

Validation of the output is needed to evaluate the result and to identify its shortcomings. The process is done through multiple tasks. The following paragraphs lay the foundation for validation which is done in Chapter 4.

Firstly, the output will be compared against other PowerShell courses. It is possible to analyse only publicly available courses. Since these online courses are likely to have a different format, e.g. long descriptive texts, tutorials and blog posts, comparisons should be made focusing primarily on content. Microsoft's official courses' training materials are not made public.

Secondly, the lab assignments and the tests should be trialled. If created properly, it should not be possible to carry out the lab assignments without learning the content from the corresponding slides, unless the student has prior experience with PowerShell.

The same applies to the tests. The scores must be below the passing threshold for students who have not been able to study the modules' slides. It would be a clear indication of a too low difficulty level, if the testes were passable without attending the seminars and self-studying by doing the lab assignments. On the other hand, the tests must be achievable to the maximum score based strictly on content provided in the course.

Next, the materials proposed by the thesis need to be reviewed by subject matter experts, i.e. PowerShell practitioners who manage infrastructures in corporate environments. Their opinions are used both as input to the module listing and as verification for the slides and lab assignments.

Besides feedback from experts, it is vital that feedback is also collected from students who matriculate and pass the course. Although this form of validation is not possible before the thesis is published, it is a key element in shaping the course in the long term.

Additionally to the above, Chapter 4 will include an analysis of deficiencies of the methodology, scope and course materials. The chapter will end with a discussion on potential future improvements that could be done to develop the course.

3 Course Platform and Modules

The following chapter presents the course material in different forms, such as class presentation slides with notes, practical lab assignments and tests.

The course will be based and tested on Windows Server 2019 using PowerShell 5.1, yet the majority of the content should be both backwards and forwards compatible with other versions of Windows and PowerShell. Certain version compatibility issues are also discussed.

Table 3 lists the course lessons which divide the course into modules focusing on different sets of topics. The course content is presented in its current iteration stage at the time of publishing the thesis and future adjustments are likely.

3.1 Lab Environment

Ashraf, 2015 wrote a paper on combining PowerShell with Microsoft Deployment Toolkit (MDT) to install system images. MDT is a viable option for deploying lab machines also for the proposed course. However, the tool is intended for installing physical machines and virtual machines are more conveniently deployed using scripting against the virtual machine host software. The scripting language is PowerShell, if Microsoft Hyper-V is used.

As emphasised in section 1.2, the infrastructure for delivering the course is not within the scope of the thesis. Nevertheless, the paper will explain how to prepare a system image which should be used as the reference image. This is also called *the golden image*. In the following, it is assumed that the lab infrastructure will be based on virtual machines (often abbreviated VM). The image files must therefore reside in a location where the virtual machine host has access to it.

3.1.1 Golden Image

The virtual machine host will use the golden image to deploy any number of clones in which students can carry out their lab assignments. This guarantees an approach which is scalable and reliable. The deployed environments will be identical for all students and initially isolated

Table 3. Overview of the proposed course modules

Part	Title	Topics
1	Introduction	Definition, history, syntax, the ISE
2	Basics 1	Variables, conditionals, operators, aliases
3	Basics 2	Loops, printing output, functions, splatting
4	Basics 3	Filtering and formatting, strings, arrays, objects
5	Windows OS Management	Files, processes, CIM, remote sessions
6	Active Directory	Managing AD objects
7	Hyper-V	Hyper-V VMs, virtual hard disks and virtual switches
8	Custom Modules	Creating custom functions and modules.

from each other. The golden image can be modified from a central location and will affect all new deployments.

Windows Server 2019 — and many other Microsoft products — have free trial versions which are available for limited time evaluation. The evaluation period for Windows Server 2019 is 180 days and the operating system installation files are available for download in the ISO format (Microsoft, 2019c).

The image will be almost identical to the one available immediately after installing and booting Windows Server 2019 with default settings. This means that services, such as Active Directory, will also need to be installed by the student and will be a part of the lab assignments.

The process of creating the golden image can be automated with Packer. It is "an open source tool for creating identical machine images for multiple platforms from a single source configuration" (HashiCorp, 2019). The features of the tool make it suitable and integral to managing the lab environment.

In the thesis, the lab environment is deployed in a local Hyper-V instance. Yet since Packer supports multiple output formats, it will be convenient to modify the input JSON files and deploy the lab VMs in Amazon Web Services, Azure, VirtualBox and on several other platforms (HashiCorp, 2019).

3.1.2 Packer Template

The Packer input file that was used to create a VM for designing and testing the lab assignments can be see on Figure 3. It is minimalist and may be expanded when the course content is developed further. The file does not include a reference an `autoattend.xml` file which means that the Windows installation wizard must be completed manually as part of the Packer

procedure.

Moreover, the local administrator password must be set to the value on line 13 in the Packer recipe file and WinRM needs to be properly configured. Unencrypted connections must be allowed and the default authentication method set to `true` by running

```
1  winrm set winrm/config/service @{AllowUnencrypted="true"}
2  winrm set winrm/config/service/auth @{Basic="true"}
```

after which the Packer process will continue. These steps can be automated by creating an XML file with the Windows System Image Manager tool available in the Windows Assessment and Deployment Kit.

Microsoft Visual Studio Code with PowerShell plug-in is included in the image. The purpose of the editor is to provide a graphical interface for scripting in the lab environment since the Integrated Scripting Environment (ISE) is not available in the Windows Server Core version. The installer and PowerShell plug-in `.vsix` file must be provided together with the ISO file. These files are freely available on the Internet. Microsoft Visual Studio Code can be launched in the lab environment by running `code`, but preferably by creating a new script and running, for example, `code script.ps1`.

Packer's output is a folder which can be imported as virtual machines into Hyper-V. Minor changes to the JSON file would allow to import the output to other virtualisation platforms.

3.2 Seminars

The lectures are designed to be 1.5 hours long. This time is a combination of presenting the slides and live demonstrations of example code. Students are expected to run the code snippets on their computers in parallel. The time spent on lab assignments which follow each lecture will vary on the performance of each student but should also be approximately 1.5 hours.

The slides are written in LaTeX using the Beamer class which means that the PDFs are built from text-based files. This implies that the content can be version controlled, for example using a Git repository, and modified to suit specific circumstances. Furthermore, the style of all slide decks can be modified from one central file `Preamble.tex` (see Appendix 1). The PDFs for seminar presentations are sub-appendices in Appendix 2.

The majority of the slides present example boxes with lines of code. The intention is that students run the code during the seminars in the virtualised lab environment. In case the lectures are recorded, the students are expected to pause the video and try the commands in parallel with watching the recordings. For the first seminars, even personal computers with standard

```
{
  "builders": [
    {
      "vm_name":"pscourse_vm",
      "type": "hyperv-iso",
      "iso_url":
      ↪  "./17763.253.190108-0006.rs5_release_svc_refresh_SERVER_EVAL_x64FRE_en-us.iso",
      "iso_checksum_type": "md5",
      "iso_checksum": "48CD91270581D1BE10C3FF3AD6C41CCE",
      "shutdown_command": "shutdown /s /t 0",
      "http_directory": "./",
      "communicator":"winrm",
      "winrm_username": "administrator",
      "winrm_password": "TalTech123",
      "winrm_timeout" : "10m"
    }
  ],
  "provisioners": [
    {
      "type": "powershell",
      "inline": [
        "Get-Date | Out-File Installed.txt",
        "Set-Location Downloads",
        "Invoke-WebRequest http://$Env:PACKER_HTTP_ADDR/VSCodeSetup-x64-1.32.3.exe -OutFile
        ↪  VSCodeSetup-x64-1.32.3.exe",
        "Invoke-WebRequest http://$Env:PACKER_HTTP_ADDR/ms-vscode-PowerShell-1.11.0.vsix
        ↪  -OutFile ms-vscode-PowerShell-1.11.0.vsix",
        ".\\VSCodeSetup-x64-1.32.3.exe /verysilent"
      ]
    },
    {
      "type": "powershell",
      "inline": [
        "Set-Location Downloads",
        "code --install-extension ms-vscode-PowerShell-1.11.0.vsix"
      ],
      "pause_before": "1m"
    }
  ]
}
```

Figure 3. `Packer.json` file which defines the golden image

Windows editions or Linux can be used to run the example code. The second half of the modules depend on cmdlets available only in Windows or Windows Server operating systems.

Many slides have notes which are meant to assist the instructor and turn attention to nuances on the slides. The notes should not be disclosed to the students with the slides but the slides' content is not designed to be self-explanatory.

3.3 Lab Assignments

Lab assignments were created for each module and they are available in Appendix 3. The assignments can be considered homework or the instructor may decide to host classroom seminars where the assignments are completed under supervision.

Each assignment contains approximately 5 tasks and a common footer text. The footer states that answers should be posted to Moodle in the form of screenshots and that the code must be commented. Most of the assignments can be carried out on any Windows 10 computer, but some tasks, such as Active Directory, require a server version of Windows. The lab environment provides the needed features and capabilities equally for all students. Therefore, the assignments are designed for and should be tested in the school provided lab environment.

Admittedly, checking the submissions on Moodle creates a workload for the instruction, but it will also allow for peer review of different approaches to the exercises. The answers should be made public after the submission deadline. Automated checking of students' code could be added in the future when the course is developed as discussed in section 4.4.

3.4 Knowledge Tests

Besides seminar slides and lab assignments, there are knowledge tests for each of the eight modules. Every module test set consists of ten questions. The test questions are available in Appendix 4.

The majority of the questions are multiple choice where at least one choice is correct. To make answering more challenging for the students, multiple or even all choices may be correct. Some questions are also in the drag-and-drop format which means that students are required to assemble or order the choices into a meaningful chain, most often a command.

Correct answers are given in the LaTeX files as comments. They are not visible in the printed version of the thesis.

It is not within the scope of the thesis to set a fixed evaluation criteria or a link between the tests and the final grade. It is however recommended that the tests are seen as a prerequisite for

passing the course while the tests' results should not affect the final grade. It is also proposed that the tests are scheduled as the final component of each module, i.e. after submitting the home assignment.

If the students are able to take the tests in an uncontrolled environment, they have the possibility to look for answers either from the lecture slides, search online resources or test the question choices in a working shell. In any case, answering the questions correctly will confirm that the student had the expected knowledge at the time of the answering.

3.5 Introduction

The first seminar focuses on providing an overview of the language's capabilities and its history. It is important to define the concept of PowerShell before teaching commands. The concept is defined by stating that it is a scripting language and listing some of its capabilities. The first slides also compare PowerShell with Python and UNIX shells.

Next, the language's history is presented. It is possible, even likely, that students need to work with different PowerShell versions during their daily tasks. Although the first versions will be deprecated with the end-of-life of Windows 7 and Windows Server 2008R2, PowerShell versions 3, 5.1 and 6 will probably be used in parallel in the coming years.

The first seminar ends with demonstrating the Integrated Scripting Environment (ISE). As stated above, all seminars are intended to be 1.5 h long in the classroom. It is the task of the instructor to plan the time so that all topics are covered within the time frame but not faster. The slides show some examples, yet given the opportunity, the instructor can go more in-depth with the topic and show additional examples. If the students follow the examples on their own computers, they are likely to ask additional questions and seminars ending ahead of time are unlikely to be a challenge.

All seminar slide decks conclude with a summary slide which gives the instructor an opportunity to recap the lecture. It is also an opportunity for the students to ask any questions about topics that remained unclear and to give feedback in general.

Introduction module assignments are basic and less technical than those of the following modules. The students are asked to highlight points from the presentation and create a basic script in the ISE which they run from the console. Besides the assignments, it is recommended that the instructor asks the students to confirm that they can log into the virtualised lab environment before the second seminar.

The first knowledge test focuses on theoretical and general questions about PowerShell covered in the seminar. Most of the questions are about essential features of PowerShell while

others evaluate factual knowledge regarding PowerShell and Windows versions.

3.6 Basics 1

The majority of slides in this module focus on example code and the same approach is also applied throughout the following seminars. The example code should be tested by students during the seminars either on their personal computers, classroom computers or in the virtualised lab environment. It should be noted that the latter does not have PowerShell ISE application installed, but includes Microsoft Visual Studio Code.

The main focus of the seminar is on introducing different variable classes and some operators. The topic of variables is discussed relatively in-depth. Firstly, the purpose and naming of variables in PowerShell are discussed. Subsequently, the most common variable classes are illustrated through examples. With the purpose of making the seminars more diverting, the examples are selected to have real life applicability when possible. For instance, the time interval calculation example is based on the Apollo 11 Moon landing mission.

Next, basic operators are covered. These operators are essential for understanding the example code in the next seminars. There are several types of operators in PowerShell and three of them are introduced in Basics 1. Firstly, the arithmetic operators show how PowerShell can be used for simple mathematical calculations. Secondly, comparison operators return `True` or `False` from comparing two values. Finally, assignment operators' table shows how values can be assigned to variables.

Basics 1 is concluded by section Aliases. PowerShell code writers rely on aliases for convenience. The slides allow the instructor to both discuss some of the most often used commands and their shorter aliases at the same time. Students can run the commands in parallel.

The lab assignment tasks for the module cover the basics of variable names and assigning values to variables. Moreover, the students are asked to demonstrate that they are able to use arithmetic and comparison operators. Similarly to the lecture and lab assignments, the test in this module focuses on evaluating knowledge on object and variable classes.

3.7 Basics 2

Basics 2 will expand on the knowledge from previous lectures. The first part of the slide deck illustrates conditional statements by using the conditional operators learned from the previous seminar. Students will gain a basic understanding of `If`, `Else`, `Elseif` and `Switch` conditional operators through examples.

The second section demonstrates the use of different types of loops in PowerShell. Loops might be challenging to comprehend for students with weaker mathematical backgrounds. It is therefore important to allow the students to try different loops with varying input on their own and in their own pace. Moreover, the instructor is expected to explain when the use of one or the other loop is most appropriate.

Next, the slides demonstrate the different methods of printing output to the console and to a text file. The differences between `Write-Output` and `Write-Host` are not intuitive and deserve illustration through examples. `Out-File` has a similar purpose to > and >> in Command Prompt. `Tee-Object` is a useful tool for monitoring time consuming scripts, especially loops, and saving output simultaneously. The section ends with presenting piping to `clip` which is not a PowerShell native cmdlet but a practical way of redirecting output to the clipboard.

Although custom modules in PowerShell are covered in the last course module, the section of slides in this seminar demonstrates how simple custom functions can be created. Functions with arguments, even more so with named arguments, may be an intricate concept to understand for some students. Therefore, the student should be encouraged to scrutinise the examples and create their own example functions. This can be done with instructor's direct help in the classroom or with instructor's or peers' feedback on the Moodle platform.

The seminar ends with explaining the concept of splatting which is a method of passing some or all parameter values to a cmdlet as a variable. It is an effective method of improving code readability. Students can try splatting arguments to the functions that they created before.

Purposely, the assignments become gradually more technically complex. The tasks in this module and the following modules focus on creating functions. Students need to show their skills by creating functions that produce specific output based on given input parameters. In this module, the students are asked to create a function for a simple calculator, birthday date tester and Fibonacci number generator.

The knowledge test is designed to confirm that the students comprehend the concepts of conditional statements, loops, writing output and custom functions. Questions are intentionally more complicated than in the previous modules and it is expected that students turn their attention to nuances in different choices.

3.8 Basics 3

The final basics module concludes the generic approach on PowerShell capabilities. The remaining modules concentrate on more specific individual topics.

Since PowerShell relies on piping output from one cmdlet to input of the next cmdlet, filtering, sorting and formatting are also done by piping. Skilful use of the `Select-Object`, `Where-Object`, `Sort-Object`, `Group-Object` and `Compare-Object` lays a foundation for managing objects in different more advanced use cases in PowerShell. Finally, in this section, `Format-Table` and `Format-List` show the students how to customise the way objects are printed in the console.

The next four sections show more advanced examples of strings, arrays, hashtables and `PSCustomObjects` respectively. Again, these examples are included with the intention of building a solid basis for PowerShell knowledge that could be used in later course modules to discuss more advanced examples. Granted, the majority of examples in this seminar have limited practical value in real life situations in the isolated way they are presented. The students are expected to explore additional methods and operators on their own and also to find creative uses for the learned content. Yet the slides aim to summarise the most basic, common and advantageous variable classes and their related methods and operators.

CSV and JSON are not PowerShell specific formats, however they interface well with PowerShell cmdlets. Presumably, the students would benefit from knowing how structured data can be imported into PowerShell from files or using the `Invoke-RestMethod` against a REST API of a web service. The same applies to exporting PowerShell objects into files and posting to REST APIs in these formats.

The final section in this course module is dedicated to getting help with PowerShell cmdlets. It is possible to either peruse offline documentation or conveniently open online manuals of cmdlets directly from the console. The students are encouraged to read the documentation and discover new parameters and ways of using the cmdlet by inspecting the Microsoft proposed examples.

In the assignments for this module, the students need to demonstrate their knowledge on filtering, formatting and writing output. Also, the tasks regard working with JSON objects and strings. The students must show their ability to use the `-f` operator and the reverse order of string characters in a loop. Most of the questions in this module's knowledge test validate knowledge on PowerShell syntax.

3.9 Windows OS Management

Windows OS Management is the first module which concentrates on managing one specific product instead of discussing the capabilities and peculiarities of PowerShell in general. The content in the module, nevertheless, builds upon the previously learned skill set.

This module is information intensive and has an above average count of sections. The first section covers file management by showing how to navigate between working directories, list files and folders, output to text files with different encodings and carry out file actions, such as copy, move, rename and delete. These are again fundamental skills which are likely to be needed in students' daily tasks once employed as an infrastructure administrator.

The next section shows how processes and services can be managed through PowerShell. This is demonstrated through the basic cmdlets of `Get-Process`, `Stop-Process` and five different `-Service` cmdlets. Some of the actions will require local administrator permissions which excludes the option of using school provided classroom computers for testing. In case the students do not have personal computers or administrator permissions on them, the virtualised lab environment meant for assignments can be used for testing code during the seminars. The same applies to the subsequent sections.

Power management in the Windows OS seminar is represented by examples of using the `Stop-Computer` and `Restart-Computer` cmdlets. More slides are dedicated to defining and demonstrating WMI and CIM. The former stands for Windows Management Instrumentation and is widely used for interfacing with the operating system and even the hardware. It is however becoming replaced in Windows by CIM which means Common Information Model. The aim is to make students aware of both terms, yet the basic examples covered in the seminar will not point out the specific differences between the similar technologies.

The Features and Roles section teaches the students the differences between features and roles in Windows server environments and shows how to use PowerShell cmdlets to manage them. This is a straightforward process and the cmdlets are for seeing the available options, installing and uninstalling. The students can trial these cmdlets in the virtualised lab environment. Installing the Active Directory role service is an environment configuration prerequisite in the next course module.

In the contemporary IT environment, many tasks are carried out remotely. The remote desktop connection allows administrator to open a console window on the remote machine and execute commands as if they were executed locally. PowerShell offers the possibility to run commands from a local console that have an effect on remote machines. This feature is especially practical in automated environments and is taught to the students in the second to last section of the lecture.

The final section covers interfacing PowerShell with the Windows Event Log API. Only the basic steps such as collecting events and creating events are demonstrated. It is hoped that if the students are aware of this API capabilities, they will find creative manners to implement Windows Event Log cmdlets in the scripts they need to create in their careers.

Windows OS Management assignments combine cmdlets learned in this module with the skills learned from the modules which covered the basics. The firsts tasks regard compiling reports from file management, processes, services and hardware. These are likely to correlate with daily tasks of an infrastructure administrator outside the academic world too. The last task pertains to practising the use of cmdlets related to Windows Event Log API. The questions in the module test pertain to cmdlets presented in the lecture and should be relatively straightforward to answer for diligent students.

3.10 Active Directory

Active Directory is a multifaceted topic and may therefore be challenging to explain to students with no prior experience with directory services. It is essential that the introduction to the module covers the basic principles and purposes of Microsoft Active Directory as a directory service in general.

The seminar continues with demonstrating how an Active Directory domain is created by installing the role service on the first domain controller in a new forest. Once the domain is created, the students are encouraged to browse and manage it by testing the presented cmdlets freely and creatively.

The majority of the slides show examples of how to create, manage and remove user and group objects. This includes adding and removing users to and from groups. These exemplified tasks constitute the basics of managing Active Directory domains for junior identity management administrators.

The final section of the slides shows examples of cmdlets which regard object types other than users and groups, namely computer accounts, contacts and organisational units. Active Directory sites and their management are out of scope due to limited seminar duration time and the introductory nature of the seminar.

Assignments in this module mostly pertain to creating Active Directory objects from JSON and CSV source files. The first task is creating a function that for example the service desk could use to reset users' password quickly via a console. The next tasks are creating users and contacts from CSV and JSON files respectively and placing the users into a group. The final task is to create more users from a CSV file, but also to place them into groups and to create the missing groups within one loop iteration. Completion of the task relies on knowledge from the previous modules. In this module, the test also contains questions that confirm the students have understood the basics of Active Directory, not only its elementary management with PowerShell.

3.11 Hyper-V

Similarly to Active Directory, virtualisation might be a concept that the students are not familiar with and it needs to be explained before PowerShell cmdlets would seem meaningful. Therefore, the seminar starts by presenting the principles and the purpose of virtualisation in general and explains that Hyper-V is Microsoft's implementation of the technology.

Virtual machines are, like Active Directory, a multifaceted topic and the seminar can only cover the essentials. The first section exemplifies how to create new virtual machines with basic parameters. Students are, as usual, expected to test the commands either on their personal computers or in a school provided lab environment. It needs to be noted though that nesting VMs is currently only supported on Intel CPUs. This may propose challenges in providing students a suitable lab environment, if AMD hardware is used.

The following sections focus on VMs' states and hardware with a separate section for managing virtual switches. These are potentially the most complex topics of the whole course, as they require background knowledge of modern computing. Virtual hardware covers the management of virtual DVD drives, CPU, RAM and VHDs (virtual hard disks), but does not discuss multimedia devices which are rarely used in corporate environments.

The final two sections regard checkpoints and PowerShell Direct respectively. Checkpoints are an integral function of VMs and allow administrators to conveniently save the machine and return to the same state after, for example, testing alternative configurations. PowerShell Direct is a technology that permits administrators to run PowerShell commands on the VMs and copy files between the host and VMs without establishing a working network connection. Admittedly, the learning of these cmdlets will remain theoretical since there will not be enough time during the seminar to install an operating system on the created VMs.

For homework, the students are asked to demonstrate the knowledge of provisioning a Windows server with the learned cmdlets. The tasks in the assignment cover the majority of the learned skills such as defining virtual hardware details, mounting operation system installation ISO, creating a checkpoint and using PowerShell Direct. It is assumed that the school provided lab environment is capable of supporting nested VMs, but the students can also carry out the assignment on their personal computers. Similarly to the questions in the Active Directory module, the questions in this module too partially pertain to measuring generic Hyper-V and virtualisation knowledge.

3.12 Custom Modules

The last course module returns to the core features of the language. Although, the main focus is on creating custom PowerShell modules, the seminar begins by introducing proper error handling in scripts. In certain cases, it is not meaningful or even possible to check all prerequisites for a successful execution of a cmdlet before it is executed. Exception handling gracefully allows the script to continue and take further actions based on the outcome of one or some cmdlets, even when they result in errors. This is made possible by defining `try`, `catch` and `finally` code blocks.

Next sections in the module demonstrate to students how custom modules are created and imported into PowerShell sessions. It is done via a simple example of a module where two mathematical functions are defined. Granted, its applicability to real life is limited, yet the aim is to present an elementary example and concentrate on the task of creating and importing a module. It is hoped that the students are able to apply this practical knowledge of creating custom modules in real life situations including practical functions.

The eighth course module ends by showing how manifest files could be added to custom PowerShell modules and help sections to custom functions. These actions admittedly are likely to exceed the job responsibilities of junior administrators, however, arguably, even junior administrators would benefit from an early knowledge of this PowerShell capability. The module ends, as always, with a summary slide allowing the instructor to recap the seminar.

Predictably, the assignment in the module is for the students to create a custom module on their own. A number of requirements are set on the submitted code. The module that students code needs to extend either the Active Directory or Hyper-V modules. Moreover, it needs to include at least four functions, variable, exception handling, a manifest file and help section in the functions.

The final knowledge test is potentially the most difficult of the eight. It includes questions regarding topics from the lecture, e.g. exception handling, custom functions with multiple parameters, custom modules and custom help pages for functions.

4 Analysis and Conclusion

Following the proposal for the course content in the previous chapter, it is important to critically analyse the materials and discuss its potential weaknesses. The value of criticising the scientific work done and honestly presenting its shortcomings was emphasised, for example, by Feynman, 1974. The failure to analyse the outcome in retrospect would result in work with little academic value.

4.1 Deficiencies of the Methodology

The shortcomings of methodology are analysed in two subsections. Firstly, the deficiencies of teaching methodologies are discussed and secondly the same is done for the approach which was used to compile the course materials. Scope limitations are analysed in section 4.2.

4.1.1 Teaching Methodology

An exhaustive analysis and specific recommendations of teaching methodology are not within the scope of the thesis. Nevertheless, a discussion of teaching approaches is relevant.

There are potential shortcomings that derive from teaching methodologies. The course is intended as an online course, yet video recordings are not done at the time of writing the thesis.

An online course has the advantage of scalability, meaning that the number enrolled students has less importance to administering the course. Students also have greater flexibility in planning their time. Yet on the other hand, it is likely that classroom interaction with an instructor would yield better learning outcomes in small or moderately sized groups.

An important term in modern pedagogy is *collaborative learning*. In essence, it connotes any attempt for two or more persons to learn something together (Dillenbourg, 1999). Bruffee, 1993 proposed a learning method and coined the term *Classroom Consensus Group*. The approach describes how students should be allocated into groups of three and given a task to solve. This method could also be used in the seminars, if the instructor sees it suitable. Additionally, collaborative learning is applied in the course through the suggested peer feedback on

submitted code via the Moodle platform.

From the large number of slides and their share in the course materials that the thesis gives as output it may seem that the seminars are intended as traditional learning via slides and instructor's speech. It must be underscored that the slides are meant for outlining the cmdlets that should be covered in each module. The instructor is expected to facilitate a process where students run the example commands on their computers or in lab environments in parallel with the seminar and understand their purpose.

It has been emphasised throughout the thesis that the students are expected to be active meaning that they should try the example code from slides directly and also generate their personal modifications. Furthermore, they will achieve maximum benefit from the course only if they find creative ways to use the learned content under different scenarios. These principles relate well to the concept of self-determination theory.

Self-determination theory explores the innate psychological needs to learn new skills (Ryan & Deci, 2000, p. 68). Moreover, the theory is concerned with social environments that foster the tendency of wishing to develop oneself (Ryan & Deci, 2000, p. 69).

Enabling self-determination in students is a key factor to the success of the course. Regardless, it is not within the scope of the thesis to present an elaborate discussion of the psychological aspects that need to be considered when delivering the course.

4.1.2 Course Material Compilation Methodology

Iterations of design science approach were used in the paper. Although it allows for continual improvements to the course content, it also means that the materials will never be fully finalised. In the current validation discussion, the final iteration before the thesis was published is analysed.

Moreover, context-driven research was applied. Yet argumentation has been put forward that context-driven research produces too specific and non-generic solutions (Basili et al., 2018, p. 45). Efforts have been made in the thesis to make the output of the thesis generic and applicable in various cases where PowerShell teaching is needed.

Potentially the most significant deficiency of the course materials is that the content selection was to a large extent subjective. Three major sources of input were used. The sources are shown in Figure 4 Venn diagram.

Microsoft's official course content could be used to a limited extent because the materials are not public and they are protected by copyright laws. The interviewed PowerShell subject matter experts at Swedbank and Telia are competent and their knowledge provided valuable input. Nevertheless, their vision of the course is subjective and influenced by personal opinions

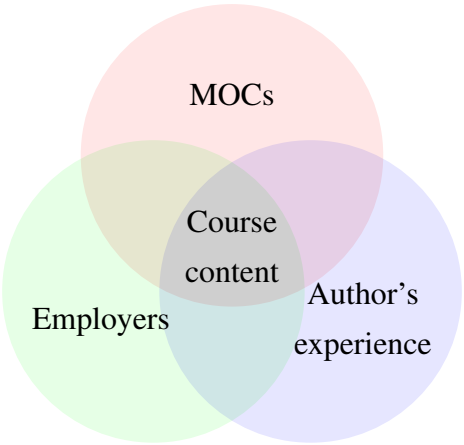


Figure 4. Course content sources

and the same applies to author's experience.

The argumentation above contributes to poor falsification possibilities of the materials. Arguably, the best measure of success of the course would be positive feedback from students and employers who hire the graduating students as junior administrators.

4.2 Scope Limitations

Certain aspects pertaining to the course proposal were deliberately excluded from the paper. The following section lists and discusses these exclusions.

4.2.1 Infrastructure Requirements

It was out of scope for this paper to describe the steps that the university needs to take in order to prepare the virtualised environment for the course. Nevertheless it is important to list the prerequisites for the infrastructure.

The VMs that are provided to the students are expected to run in Windows Server 2019 operating systems, although Windows Server 2016 is also likely to suffice. Moreover, nested virtualisation should be allowed as the feature is needed for running the Hyper-V module cmdlets.

Hardware requirements are not high because the recommended installation of Windows Server is without graphical environment and thus not performance intensive. A single or dual core CPU, 2 GB of dynamically allocated RAM and 8 GB HDD per VM are probably sufficient.

Automation is vital for an efficient management of the lab environment. Palmer, 2015 did a case study in Southampton Solent University and recommended to use PowerShell for this purpose too. Configuring the lab computers with PowerShell allowed to decrease deployment

time by 600%. Moreover, the author explained that automating the shutdown and updates made the lab energy efficient and secure.

Therefore, it is recommended that the management of student dedicated VMs is also automated. It should be enough if the instructor inputs the list of enrolled students' names into a script to automatically provision the VMs.

A common network between the student VMs or access to the Internet from them is not a requirement, but could be beneficial when developing the course. For example, this would allow team work in the assignments where each student has a part of a project which involves interfacing with other students' servers.

The golden image proposed in chapter 3 includes Microsoft Visual Studio Code installation, yet the application and its PowerShell extension request regular updates from the Internet. Although, these updates are unlikely to be critical, they could provide value to students. Additionally, instructor's remote access to students VMs would allow for automated analysis of if and how students use the lab environment.

4.2.2 Limitations of Course Materials

The following subsection aims to identify real and potential problems with the produced course materials. Firstly, the course focuses on Microsoft's products. In spite of the fact that PowerShell is developed by Microsoft, the scripting language is now open source and can be used to manage UNIX platforms as well. Furthermore, PowerShell is capable of managing certain 3rd party software products which were not covered in the course.

Secondly, the course assignments currently contain no team projects. In the real world scripts are written collaboratively in team settings (Cai & Arney, 2018, p. 147). The usefulness of collaboration exercises is also emphasised and seen as an educational tool by Marshall, Pieterse, Thompson, and Venter, 2016, p. 5:2. It is recommended that incorporating group assignments is considered in further iterations of the course. The topic is discussed in the next section.

In order for the course to support a complete e-learning approach, the seminars need to be recorded as videos. A feasible method for this would be to record the first set of classroom seminars and upload the videos to Moodle either during the same or next semester.

Although scalability was set as a prerequisite for the validation of course material, a balance between classroom and independent learning should to be found in the long term. On one hand, classroom education is likely to be more effective because students have the option to get immediate feedback on their questions. On the other hand, it is less efficient since the students will loose the flexibility of planning their own time and the number of students concurrently in

a classroom is limited.

Furthermore, it may be assumed that students have different backgrounds and capabilities. Classroom exercises would be preferred for helping the ones who are new to scripting. It is not possible to answer all questions that they might have in pre-recorded video lectures. Classroom seminars are also more effective on students who are are less motivated for self-studying.

The materials do not prescribe how many ECTS the course should award. Additionally, the thesis does not state if the evaluation outcome must be a pass-fail result or a specific grade is appointed to student. Nevertheless, a recommendation is made that the course should award 2 ECTS and, at least initially, assign no specific grade to students.

The course is divided into eight modules which should be distributed evenly, either weekly or biweekly. 1 ECTS should correspond to 26 hours spent on earning the credit point. This means that the time spent per module would be 6.5 and 9.75 h for 2 ECTS and 3 ECTS respectively. The seminars and home assignments were designed to take 1.5 h each. Combined with self-learning time and tests, it seems reasonable to define the course as rewarding 2 ECTS. Reorganising the course to a 3 ECTS one is also plausible, if potential future improvements discussed in section 4.4 are implemented.

There are certain requirements that the materials set on the instructor of the course. Firstly, LaTeX language knowledge is needed to modify the seminar slides and assignments in their current form. LaTeX is well documented online, yet it is not as intuitive as PowerPoint and may present an additional challenge to the instructor.

Finally, the slides are not self-explanatory or intuitive. In other words, the slides are merely content and example code guidelines. This means that extensive preparation and PowerShell knowledge is needed prior to delivering the proposed course.

4.3 Validation

Validation methodology was described in section 2.3 and in this section, the outcome from the previous chapter is validated. The goal of the validation is to give the produced output more academic weight.

Firstly, the materials were compared against other PowerShell courses. Microsoft's two courses, although not publicly available, were used as inspiration. The first of the two focuses on teaching the basics of PowerShell while the second intends to provide tools for developing custom functions and modules. The proposed course is a combination of both. It could be argued that more emphasis should be on the basics and less on advanced topics such as custom modules in such introductory course. This risk was also raised in the discussion with

a practitioner at Telia. Nevertheless, it is hoped that the students are able to comprehend the development features of PowerShell as well and use the knowledge in their careers.

There are PowerShell courses on Wikiversity and Tutorialspoint which search engines suggest. Although, the content of these tutorials was not analysed prior to creating the content, it became apparent that the contents are relatively similar.

The proposed course and the online courses present the building blocks of the language such as variables, operators, loops, conditions, custom functions, aliases, most used management cmdlets, etc. (Tutorials Point, 2019; Wikiversity, 2019) There are however significant differences in how the elements are divided into modules and the modules ordered. Furthermore, the examples used to illustrate the concepts differ. The course at Wikiversity includes some test questions at the end of each module, but in general, the knowledge tests and lab assignments after each module were found to be unique to the proposed course material.

Efforts were made to analyse how similar cases have been taught at other universities. Results were inconclusive as only limited info is published on universities' web sites. Granted, the few found courses included more elaborate lab work scenarios, probably because the courses had been under development longer (see for example the Python course available at KTH Royal Institute of Technology, 2017).

Secondly, the knowledge tests were trialled. Current students in the IT Systems Administration programme were asked to answer the test questions without having the possibility to peruse lecture slides beforehand. 47 individual test results showed that approximately 23% of questions were answered correctly. The measure is higher than answering the questions fully at random, but below a reasonable passing threshold. Moreover, the majority of incorrect answers were given to questions which had wrong answer choices inspired by other scripting languages such as Python, Bash and JavaScript.

This further justifies the need for a course where PowerShell specifics and peculiarities are taught. Nevertheless, the tests must be re-trialled on students who have attended the course.

Thirdly, assignments were tested in the lab environment. It was possible to carry them out and it is estimated that the students would use approximately 1.5 to 2 hours on each.

Tests and lab assignments must be re-validated once students have attended the lectures and submitted their answers in the real course context. The evaluation should be based on the results that the students obtain and their feedback.

Finally, interviews were conducted with senior PowerShell practitioners at Swedbank and Telia in Estonia. There were significant overlaps between what was initially compiled into the modules and what the subject matter experts suggested the course should cover. Moreover, adjustments to the course material were made based on their feedback and recommendations.

Student feedback was not collected during the creation of initial course materials but it must be an integral part to success evaluation after the course has been delivered for at least one semester. Students' overall evaluation of the course will depend also on the instructor's character, but it is intended to be an important component in developing the course in the long term.

4.4 Potential Future Improvements

The final section of the chapter suggests ideas how the course can be developed. The materials and teaching approach proposed in the thesis are not claimed to be final or fixed.

Implementation of automatic lab assignments evaluation would contribute to the scalability of the course. In the current proposal, the students are expected to upload their code to Moodle for instructor and peer review.

It would be worth investigating whether an automatic analyser could be employed to check the work of students on their VMs, i.e. if the assignments have been carried out correctly. This would however not fully replace human review as pieces of code that result in the same outcome can be written in various ways and style still needs assessment.

As suggested above, the students are likely to benefit from team projects. There is academic literature suggesting that better learning outcomes are obtained via team assignment, especially in international groups (see, for example, Bosnić, Čavrak, & Žagar, 2019). The students could, for instance, work within the same Active Directory forest as administrators of the same or partnering companies.

The technical requirement would be to bridge the VM network adapters to the same virtual subnet on the host machine. This future improvement proposal relates back to the collaborative learning concept and its benefits discussed in section 4.1.

The course could be developed further by including external partners, such as local companies that rely on PowerShell in their infrastructure management. They might have suggestions to direct the course content. Moreover, the students might enjoy ad hoc seminars with practitioners who are experts on certain PowerShell modules. The benefits of including external parties to courses were listed for example by McGill, Armarego, and Koppi, 2012, p. 8:3 and Steghöfer et al., 2018.

Additionally, the course could offer some content flexibility to the students. In other words, the students could have the option to select their focus in the course. Granted, the PowerShell basics would need to be mandatory for all students, but a partially personalised course is likely to raise student motivation, as suggested by Teo and Gay, 2006, p. 3.

In the current proposal, the assignments and knowledge tests per module are common for all students. Parametrised tests, as argued by Brusilovsky and Sosnovsky, 2005, lead to more accurate knowledge evaluation. The idea can be expanded by parametrising assignments. This would decrease the risk of illegal cooperation, i.e. copying each other's work among students.

Cloud technologies are becoming rapidly more popular, but were not included in the course material at this time. As potential improvement suggestions, Azure cloud VM management could replace or augment the module for Hyper-V and Office 365 with Azure AD could do the same for Active Directory.

The final idea pertains to integrating the proposed course with other courses. As it is currently put forward, the course is autonomous. There might be possibilities to align the course with other technical courses, such as Windows Server management, in the IT Systems Administration programme.

5 Summary

In the thesis, a suggestion was made to establish a new course at TalTech which would teach students the PowerShell scripting language. The course is intended foremost for students in the IT Systems Administration programme.

The demand for the course was justified by Powershell's prevalence in modern administration, Estonian Association of Information Technology and Telecommunications' request and the current absence of course materials that would be suitable for the university. The output of the paper was limited to lecture slides, lab assignments and knowledge tests which are available in Appendices 2, 3 and 4 respectively.

Design science was used as the methodological approach to develop the course materials through iterations. The included and excluded topics were determined by Microsoft's official courses' content, subject matter experts' recommendations and author's personal experience.

As a result, a course was composed of eight modules. The first half of the modules teach the basic concepts of PowerShell while the last four modules focus on more specific or advanced topics, such as Active Directory or Hyper-V.

Moodle learning management system hosted by TalTech was the proposed platform for the course. It was left undetermined if the course should be fully online or combine classroom seminars with e-learning. Classroom interactions with an instructor are likely to lead to better learning outcomes, however at the cost of course scalability and students' time planning flexibility. Nevertheless, video lectures are not recorded at the time the thesis is published.

The materials were analysed in retrospect and validated by comparing the output against other similar courses and consulting PowerShell practitioners. Moreover, the scope limitations were discussed emphasising that the infrastructure for the lab environment was not within the scope of the thesis, but it has an integral role delivering in the course.

Finally, suggestions were made regarding how the course could be developed further in the future. For example, the course could benefit from parametrised lab assignments and knowledge tests and also from automated evaluation of students' submissions. Furthermore, teamwork projects could be incorporated into the course. Lastly, it was recommended that the course content is regularly reviewed based on students' and employers' feedback.

References

Amazon Web Services. (2019). AWS tools for PowerShell. Retrieved January 11, 2019, from https://aws.amazon.com/powershell/

Ashraf, M. N. (2015). Deploying and managing state-of-the-art workstation labs like a boss! In *Proceedings of the 2015 ACM annual conference on SIGUCCS* (pp. 43–48). doi:10.1145/2815546.2815570

AXELOS. (2019). From v3 to 4 — This is the new ITIL. Retrieved April 6, 2019, from https://www.axelos.com/news/blogs/february-2019/from-v3-to-4-this-is-the-new-itil

Barakat, A., & Hadi, A. (2016). Windows forensic investigations using PowerForensics tool. In *2016 cybersecurity and cyberforensics conference (CCC)* (pp. 41–47). doi:10.1109/CCC.2016.18

Basili, V., Briand, L., Bianculli, D., Nejati, S., Pastore, F., & Sabetzadeh, M. (2018). Software engineering research and industry: A symbiotic relationship to foster impact. *IEEE Software*, *35*(5), 44–49. doi:10.1109/MS.2018.290110216

Bhagyavati. (2006). Laboratory exercises in online information assurance courses. *J. Educ. Resour. Comput. 6*(4). doi:10.1145/1248453.1248457

Bhavnani, S. K., Peck, F. A., & Reif, F. (2008). Strategy-based instruction: Lessons learned in teaching the effective and efficient use of computer applications. *ACM Trans. Comput.-Hum. Interact. 15*(1), 2:1–2:43. doi:10.1145/1352782.1352784

Bosnić, I., Čavrak, I., & Žagar, M. (2019). Assessing the impact of the distributed software development course on the careers of young software engineers. *ACM Trans. Comput. Educ. 19*(2), 8:1–8:27. doi:10.1145/3274529

Bruffee, K. (1993). *Collaborative learning: Higher education, interdependence, and the authority of knowledge*. Johns Hopkins University Press.

Brusilovsky, P., & Sosnovsky, S. (2005). Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK. *J. Educ. Resour. Comput. 5*(3). doi:10.1145/1163405.1163411

Buyya, R., & Barreto, D. (2015). Multi-cloud resource provisioning with Aneka: A unified and integrated utilisation of Microsoft Azure and Amazon EC2 instances. In *2015 interna-*

tional conference on computing and network communications (CoCoNet) (pp. 216–229). doi:10.1109/CoCoNet.2015.7411190

Cai, Y., & Arney, T. O. (2018). Scripting for administration, automation and security. In *Proceedings of the 19th annual SIG conference on IT education* (pp. 147–147). SIGITE '18. doi:10.1145/3241815.3241829

Carlsson, S. A., Henningsson, S., Hrastinski, S., & Keller, C. (2009). An approach for designing management support systems: The design science research process and its outcomes. In *Proceedings of the 4th international conference on design science research in information systems and technology* (21:1–21:10). DESRIST '09. doi:10.1145/1555619.1555647

Choo, K.-K. R., Fei, Y., Xiang, Y., & Yu, Y. (2016). Embedded device forensics and security. *ACM Trans. Embed. Comput. Syst. 16*(2), 50:1–50:5. doi:10.1145/3015662

Dillenbourg, P. (1999). *Collaborative learning: Cognitive and computational approaches*. Advances in learning and instruction series. Elsevier Science & Technology Books.

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput. 5*(3). doi:10.1145/1163405.1163409

Feynman, R. P. (1974). Cargo cult science. Retrieved January 4, 2019, from http://calteches.library.caltech.edu/51/2/CargoCult.pdf

GitHub. (2019a). PowerForensics - PowerShell digital forensics. Retrieved February 19, 2019, from https://github.com/Invoke-IR/PowerForensics

GitHub. (2019b). PowerShell for every system! Retrieved January 11, 2019, from https://github.com/powershell/powershell

HashiCorp. (2019). Introduction to Packer. Retrieved February 6, 2019, from https://packer.io/intro/

Hendler, D., Kels, S., & Rubin, A. (2018). Detecting malicious PowerShell commands using deep neural networks. In *Proceedings of the 2018 on Asia conference on computer and communications security* (pp. 187–197). ASIACCS '18. doi:10.1145/3196494.3196511

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *Management Information Systems Quarterly*, *28*(1), 75–105. Retrieved from http:%20//dl.acm.org/citation.cfm?id%20=%202017212.2017217

Information Technology Foundation for Education. (2019). Moodle. Retrieved January 18, 2019, from https://www.hitsa.ee/teenused/moodle

Isomöttönen, V., & Tirronen, V. (2016). Flipping and blending — an action research project on improving a functional programming course. *Trans. Comput. Educ. 17*(1), 1:1–1:35. doi:10.1145/2934697

Jayaseelan, G., & Charles, P. J. (2014). Automated secured disaster recovery with Hyper-V replica and PowerShell. In *2014 world congress on computing and communication technologies* (pp. 150–153). doi:10.1109/WCCCT.2014.60

Jordan, P., Patten, C. V., Peterson, G., & Sellers, A. (2016). Distributed PowerShell load generator (D-PLG): A new tool for dynamically generating network traffic. In *2016 6th international conference on simulation and modeling methodologies, technologies and applications (SIMULTECH)* (pp. 1–8).

KTH Royal Institute of Technology. (2017). Laborationer. Retrieved April 30, 2019, from https://www.kth.se/social/course/DD1320/subgroup/vt-2017-298/page/laborationer-188/

Kunkle, W. M., & Allen, R. B. (2016). The impact of different teaching approaches and languages on student learning of introductory programming concepts. *ACM Trans. Comput. Educ. 16*(1), 3:1–3:26. doi:10.1145/2785807

Limoncelli, T., Hogan, C., & Chalup, S. (2016). *The practice of system and network administration: Volume 1: DevOps and other best practices for enterprise IT*. Pearson Education.

Liu, C., Xia, B., Yu, M., & Liu, Y. (2018). PSDEM: A feasible de-obfuscation method for malicious PowerShell detection. In *2018 IEEE symposium on computers and communications (ISCC)* (pp. 825–831). doi:10.1109/ISCC.2018.8538691

Markham, T. (2011). Project based learning: A bridge just far enough. *Teacher Librarian*, *39*(2).

Marshall, L., Pieterse, V., Thompson, L., & Venter, D. M. (2016). Exploration of participation in student software engineering teams. *ACM Trans. Comput. Educ. 16*(2), 5:1–5:38. doi:10.1145/2791396

Marx, F., Mayer, J. H., & Winter, R. (2012). Six principles for redesigning executive information systems—findings of a survey and evaluation of a prototype. *ACM Trans. Manage. Inf. Syst. 2*(4), 26:1–26:19. doi:10.1145/2070710.2070717

McGill, T., Armarego, J., & Koppi, T. (2012). The teaching–research–industry–learning nexus in information and communications technology. *Trans. Comput. Educ. 12*(1), 1:1–1:20. doi:10.1145/2133797.2133798

Meyer, M., Rensing, C., & Steinmetz, R. (2011). Multigranularity reuse of learning resources. *ACM Trans. Multimedia Comput. Commun. Appl. 7*(1), 1:1–1:23. doi:10.1145/1870121.1870122

Microsoft. (2019a). Course 10961c: Automating administration with Windows PowerShell. Retrieved February 16, 2019, from https://www.microsoft.com/en-us/learning/course.aspx?cid=10961

Microsoft. (2019b). Course 10962c: Advanced automated administration with Windows PowerShell. Retrieved February 16, 2019, from https://www.microsoft.com/en-us/learning/course.aspx?cid=10962

Microsoft. (2019c). Microsoft evaluation center. Retrieved February 6, 2019, from https://www.microsoft.com/en-us/evalcenter/evaluate-windows-server-2019

Microsoft. (2019d). PowerShell. Retrieved January 11, 2019, from https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-6

Moodle. (2019). Moodle - Open-source learning platform — Moodle.org. Retrieved January 18, 2019, from https://moodle.org/

Nunamaker, J. F., Jr., & Briggs, R. O. (2012). Toward a broader vision for information systems. *ACM Trans. Manage. Inf. Syst. 2*(4), 20:1–20:12. doi:10.1145/2070710.2070711

O'Grady, M. J. (2012). Practical problem-based learning in computing education. *Trans. Comput. Educ. 12*(3), 10:1–10:16. doi:10.1145/2275597.2275599

Palmer, N. (2015). Work in progress - Automation of a computer networking laboratory. In *2015 IEEE global engineering education conference (EDUCON)* (pp. 348–353). doi:10.1109/EDUCON.2015.7095995

Palumbo, T. (2017). The power of PowerShell: Examples of how PowerShell scripts can supplement a patch management system to solve unusual problems. In *Proceedings of the 2017 ACM annual conference on SIGUCCS* (pp. 7–14). doi:10.1145/3123458.3123479

Rusak, G., Al-Dujaili, A., & O'Reilly, U.-M. (2018). AST-based deep learning for detecting malicious PowerShell. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security* (pp. 2276–2278). CCS '18. doi:10.1145/3243734.3278496

Ryan, R., & Deci, E. (2000). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *The American psychologist*, *55*, 68–78. doi:10.1037/0003-066X.55.1.68

Saltz, J. S., & Heckman, R. R. (2018). A scalable methodology to guide student teams executing computing projects. *ACM Trans. Comput. Educ. 18*(2), 9:1–9:19. doi:10.1145/3145477

Sihtasutus Kutsekoda. (2018). Tulevikuvaade tööjõu- ja oskuste vajadusele: Info- ja kommunikatsiooni-tehnoloogia. Retrieved January 16, 2019, from http://oska.kutsekoda.ee/wp-content/uploads/2018/11/IKT-terviktekst.pdf

Silva, E., & Moreira, D. (2003). Webcom: A tool to use peer review to improve student interaction. *J. Educ. Resour. Comput. 3*(1). doi:10.1145/958795.958798

Statistics Estonia. (2019). Statistical database. PAV011: Job vacancies and occupied posts by economic activity. Retrieved January 28, 2019, from http://andmebaas.stat.ee/

Steghöfer, J.-P., Burden, H., Hebig, R., Calikli, G., Feldt, R., Hammouda, I., ... Liebel, G. (2018). Involving external stakeholders in project courses. *ACM Trans. Comput. Educ. 18*(2), 8:1–8:32. doi:10.1145/3152098

Swedbank. (2019). Private interview.

TalTech. (2019). Skriptimiskeeled. Retrieved January 11, 2019, from http://ois.ttu.ee/pls/portal/ois2.ois_public.main

Telia. (2019). Private e-mail thread.

Teo, C. B., & Gay, R. K. L. (2006). A knowledge-driven model to personalize e-learning. *J. Educ. Resour. Comput. 6*(1). doi:10.1145/1217862.1217865

Tsuda, Y., Nakazato, J., Takagi, Y., Inoue, D., Nakao, K., & Terada, K. (2018). A lightweight host-based intrusion detection based on process generation patterns. In *2018 13th Asia joint conference on information security* (pp. 102–108). doi:10.1109/AsiaJCIS.2018.00025

Tutorials Point. (2019). Powershell Tutorial. Retrieved April 8, 2019, from https://www.tutorialspoint.com/powershell/index.htm

Vene, S. (2018). ITL meeting protocol, 10 October 2018.

Vene, S. (2019). Private interview.

Veritas. (2019). Veritas Enterprise Vault PowerShell cmdlets. Retrieved January 11, 2019, from https://www.veritas.com/support/en_US/doc/96069939-120347322-0/index

Wikiversity. (2019). PowerShell. Retrieved April 7, 2019, from https://en.wikiversity.org/wiki/PowerShell

Wueest, C. (2016). *The increased use of PowerShell in attacks*. Symantec. Retrieved from https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-powershell-in-attacks-16-en.pdf

Yeomans, L., Zschaler, S., & Coate, K. (2019). Transformative and troublesome? Students' and professional programmers' perspectives on difficult concepts in programming. *ACM Trans. Comput. Educ. 19*(3), 23:1–23:27. doi:10.1145/3283071

Appendix 1 `Preamble.tex`

```
\usepackage[T1]{fontenc}
\usepackage{minted}
\usepackage{pgfpages}
\usepackage{xstring}

\usetheme{Berlin}

\institute{TalTech}
\date{}
\subtitle{PowerShell Course}

\setbeamertemplate{note page}{\vspace{1em}Notes for the previous slide\insertnote}
\setbeameroption{show notes}

\setminted{autogobble,breaklines}

\AtBeginSection[]
{
  \begin{frame}
    \frametitle{Table of Contents}
    \tableofcontents[currentsection]
  \end{frame}
}

\title{\StrSubstitute{\jobname}{\string_}{ }}
```

Appendix 2 Slides

2.1 Introduction

Table of Contents

Table of Contents

What Is It?

PowerShell is a general purpose scripting language by Microsoft

Remark

Scripting languages are different from programming languages

Notes for the previous slide

1. Microsoft has launced it initially and is still developing it.
2. More about "general purpose" on the next slide.
3. Scripting languages are interpreter based whereas programming languages are compiler-based. Scripts are intpreted during run time and programmes are compiled into binary files that contain direct instructions for the computer.
4. PowerShell is actually also the name of the command-line shell which is the intepreter for the PowerShell language

What Can We Do with It?

Almost anything to manage the computer, or thousands of them, even remotely. For example,

- Create, copy, move, delete files
- Use loops, calculate, manipulate strings etc.
- Manage processes and services
- Read and edit the registry
- Manage products, such as Active Directory, Exchange, Hyper-V etc.

Remark

There are several custom modules available from third party vendors

Notes for the previous slide

1. The list of features is far from conclusive. It barely scratches the surface
2. Third party vendors like Veritas with Enterprise Vault or Amazon Web Services
3. You can even create your own modules taylored to what you or administrators in your company need to do daily

Why Not Python?

PowerShell

- becomes pre-installed with the OS since Windows 7 and Windows Server 2008 R2
- has a strong focus on supporting Microsoft's products

Note

Granted, PowerShell is slower and less capable than Python in many use cases

Notes for the previous slide

1. It is very convenient to use as it almost never needs installing on Windows machines
2. PowerShell is the language to use to manage an infrastructure based on Microsoft products. This applies even if some endpoint management tool (Miradore, SCCM, Kaseya, Workspace ONE, etc.) is used for managing or monitoring
3. PowerShell does not handle large amount of data as well as Python when it comes to data analysis. Python is much richer in features and libraries

Differences with UNIX

UNIX

- relies heavily on text-based data that is processed with `AWK`, `grep` or `sed`.

PowerShell

- uses objects

Notes for the previous slide

1. The names of the UNIX tools are just informative, they are not relevant in the context if this course
2. More about objects in the variable types section
3. UNIX systems management with PowerShell is possible but rarely applied

Table of Contents

PowerShell 1.0 and Before

- Initially the project was developed under the name *Monad*
- Version 1.0 was released in November 2006 for Windows XP, Windows Server 2003 and Windows Vista

Notes for the previous slide

1. The first version did not gain significant popularity, probably because it needed manual installtion

PowerShell 2.0

- Many important improvements over version 1.0
- It was integrated with Windows 7 and Windows Server 2008 R2

Notes for the previous slide

1. It was widely used and supported
2. This version will lose its importance when Windows 7 and Server 2008 R2 reach end of life

PowerShell 3.0

- Various improvements, especially IntelliSense
- Shipped with Windows 8 and Windows Server 2012
- Since these products are replaced by Windows 8.1 and Server 2012 R2, so is the version of PowerShell

Notes for the previous slide

1. IntelliSense is a general term for a variety of code editing features that include code completion, parameter info, quick info, and member lists
2. Windows 8, not 8.1, is unsupported already as of January 2016

PowerShell 4.0

- PowerShell 4.0 is integrated with Windows 8.1 and with Windows Server 2012 R2
- Added Desired State Configuration

Notes for the previous slide

1. Quite popular since Windows Server 2012 R2 is or at least was widely used
2. DSC is a management platform in PowerShell that enables you to manage your IT and development infrastructure with configuration as code

PowerShell 5.0 and 5.1

- PowerShell 5.0 included some new features, but oriented for more advanced use cases
- PowerShell 5.1 comes pre-installed with Windows 10, Windows Server 2016 and Windows Server 2019 operating systems

Notes for the previous slide

1. At the time of the presentation, this is the version to learn and will be used also in the course in a Windows Server 2019 lab environment
2. The version differences are not huge and the majority of syntax is backwards and forwards compatible
3. Newever versions of PowerShell can be installed on older operating systems as a rule of thumb

PowerShell Core 6.0 and Beyond

- PowerShell Core 6.0 became available in January 2018
- It has no significant new functionality
- But it is open-source and also available for Linux and macOS
- Microsoft expects to release a new minor version for PowerShell Core 6.0 twice a year

Notes for the previous slide

1. Notice the *Core* in the name
2. PowerShell on Linux and macOS is not in the scope for the course, but you are encouraged to test it on your own

Table of Contents

Verbs and Nouns

Cmdlets have a `Verb-Noun` name syntax

Example

```
Get-Content
Set-Location
Copy-Item
Stop-Process
```


Notes for the previous slide

1. Comandlets have a verb-dash-noun syntax. It is therefore somewhat similar to everyday language, for example Do-This

Parameters and Arguments

Parameters come after the cmdlets and dictate how they behave and what they actually do. All parameters have an explicit name

Example

```
Remove-Item -Path File1
```

`-Path` is the parameter name and `File1` is the argument.

Notes for the previous slide

1. The example removes File1 from the current working directory

Positional Parameters

In some cases, it is not needed to specify the parameter name

Example

```
Remove-Item File1
```

The cmdlet will assume that the *first unnamed* (also called *positional*) parameter is the file path

Notes for the previous slide

1. This example has the same effect as the previous one
2. Be mindful of which method you use. Leaving out parameters is a quicker way of writing code but will make it more difficult for others to read, especially when multiple positional parameters are used together

Piping

Piping is passing output from one command as input to the next command. PowerShell uses it heavily

Example

```
Get-Process |
Where-Object { $_.WorkingSet -gt 20MB } |
Sort-Object Name -Descending |
Select-Object -First 10
```


Notes for the previous slide

1. The example chains cmdlets together and outputs 10 processes that use more than 20 MB of memory sorted by name alphabetically starting from Z
2. It is a good idea to try typing in the command, so you would know where to find all the special characters on your keyboards

Comments

Comments are preceded by the # sign

Example

```
Get-Service # shows services and their statuses

<# this is a multi line comment
Remove-Item File1
and commands here are not actually executed #>
```


Notes for the previous slide

1. Comments are useful for annotating the code making it easier for others to understand, also for yourself later when you read it later
2. File1 is of course not removed

Basic Variable types

Variables are preceded by the $ sign

Example

```
$x = 1
$y = "some string"
$z = 3, 5.6, "a" # an array with three elements
$y # prints the value of variable $y
```

Note

There are many more different classes of variables

Notes for the previous slide

1. Spaces around the equal sign are optional
2. Variable z is an array which is a basically list of values. They are very useful in loops
3. Note how commas separate elements and decimals are separated by a full stop

Table of Contents

Integrated Scripting Environment (ISE)

The ISE provides a more convenient interface to PowerShell than the regular console window

Lab Assignments

The ISE will not be available in the lab assignments

Notes for the previous slide

1. You can open the ISE on your Windows client PCs and try the examples during the seminars
2. Do not get drawn into it though and remember to listen
3. It is very similar to integrated development environments or IDEs

Shortcuts

By default, the programme opens with one untitled file and one PowerShell tab (console). You can create new files by pressing `Ctrl+n` and additional tabs by pressing `Ctrl+t`

Notes for the previous slide

1. Running something in one tab will block it from running new commands until the previous finishes or is cancelled
2. Tabs are useful for running several commands or scripts in parallel

Running Lines and Files

- You can run a line or the selected part of it by pressing `F8`
- To run the whole file, press `F5`

See what running `"test"` and `2+3*5` outputs in the console

Notes for the previous slide

1. There are of course many other shortcuts, but we will not discuss more now

IntelliSense

- An important feature of the ISE is IntelliSense. It is a general term for a code-completion aid. IntelliSense can help complete commands, parameters, values and show helpful tooltips.
- ISE usually displays potential endings automatically, but you can also manually see what is available by pressing `Ctrl+Space`. To accept the highlighted item, press `Enter` or `Tab`

Notes for the previous slide

1. Try not to rely on IntelliSense too much. It will not always be available, e.g. in the lab assignments

Deprecation

That said, the ISE is deprecated in PowerShell Core. Since Windows Server 2019 ships with PowerShell 5.1, the ISE is still available. Microsoft recommends that cross-platform coding of PowerShell in the future is done in Visual Studio Code

PowerShell Version

The current version of the shell can be seen by running

- `Get-Host` or
- `$PSVersionTable`

.ps1 File Extension

The ISE saves files with the `.ps1` file extension. It can be used to conveniently run all commands in the file and call it from within other scripts as well. In order to run a script from the console, use the `.\Script.ps1` syntax

Table of Contents

What Did We Learn?

PowerShell

- is a scripting language by Microsoft
- is pre-installed in Windows operating systems
- is open-source since version 6.0
- can be used for a wide range of management tasks
- can be written in ISE when GUI is available

2.2 Basics 1


Variables
Operators
Aliases
Summary
Basics 1
PowerShell Course
TalTech
TalTech
Basics 1

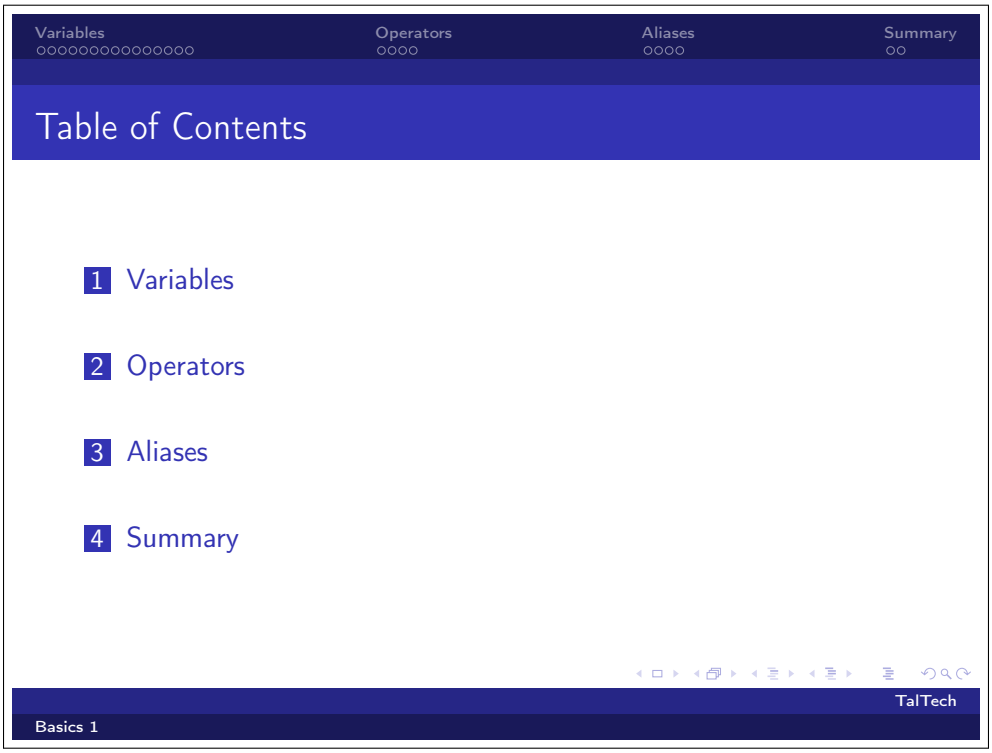

Variables
Operators
Aliases
Summary
Table of Contents
1 Variables
2 Operators
3 Aliases
4 Summary
TalTech
Basics 1

Table of Contents

Purpose

The purpose of variables is storing data so you could use it later in code

Example

```
$a = 5
$b = 10
$a + $b * $a # 55
$c = "test"
$c.ToUpper() # TEST
$d = $b - $a
$c+$d # test5
"$d$c" # 5test
```

Notes for the previous slide

1. These are very simple examples. Let's talk through them. Once variables are defined, we can call them again and again in different scenarios
2. Comment behind the line shows output. This syntax will be used throughout the course
3. `$d+$c` will not print `5test`

Naming Variables

Variables are prefixed with a $. Spaces and dashes are not recommended, but are allowed when surrounded by curly brackets

Example

```
$myVariable
$another_Variable1
${a-very-odd variable name}
```

Naming Variables

Invalid Variables

```
without_dollar_sign
$some-variable
$my variable
```

Remark

Variable names are case *insensitive*. `$variable` is the same as `$VARIABLE` or `$VaRiAbLe`

Notes for the previous slide

1. Without $ the PowerShell interpreter will see the input as a function. More on them later

Types of Variables

Variables can hold data in a variety of forms. Some of the most common types are

Type	Description
`[Int]`	Integer
`[String]`	Text
`[Bool]`	True or false value
`[Float]` or `[Single]`	Floating point number
`[Array]`	Collection of elements
`[DateTime]`	Time stamp
`[TimeSpan]`	Time interval or period
`[Hashtable]`	Collection of key-value pairs
`[PSCustomObject]`	PowerShell object

Notes for the previous slide

1. We will discuss them one by one
2. The type needs to be defined in front of the variable or a function that outputs value
3. The list of far from conclusive

Integers

An integer is a number that can be written without a fractional component. PowerShell does usually a good job guessing what you mean, but sometimes you need to be specific

Example

```
$input = Read-Host "Enter a number" # let's enter 3
$input * $input # 333
[Int]$input2 = Read-Host "Enter a number" # 3 again
$input2 * $input2 # 9
```


Notes for the previous slide

1. The asterisk can have different meanings. For integers, it calculates the mathematical product. Yet the first factor may be a string. The second factor will always be intpreted into an integer. If the first factor is a string, the string is as many times as the integer value of the second factor

Strings

A string is traditionally a sequence of characters. PowerShell allows to do a actions on them which we'll cover in the next seminars

Example

```
"test" # test
"test's" # test's
"`"test`"" # "test"
# 'test`'s' would result in an error
$a = "PowerShell"
"Variable `$a is $a" # Variable $a is PowerShell
'I like $a' # I like $a
"I like $a" # I like PowerShell
```


Notes for the previous slide

1. Strings are surrounded by single or double quoatation marks
2. In order to get an actual double quoatation mark in output, escape it with a backtick. The same applies for the dollar sign in double quoatation marks, but not for the single quoatation mark
3. Variables between double quoatation marks are replaced with their values

Boolean

Bool comes from the Boolean data type that has one of two possible values (usually denoted true and false)

Example

```
[bool]1 # True
[bool]0 # False
[bool]"some text" # True
[bool]"0" # True
[bool]"" # False
[bool]-100 # True
```


Notes for the previous slide

1. Note how it behaves. String, even just the character 0 is still true. Negative numbers also yield true

Floats

Simply put, floats are numbers with fractions, for example 1.5 and 3.4e-2 (which is the same as writing 0.034)

Note

PowerShell always uses "." as the decimal separator for input. It may be a comma or a point in output

Example

```
[float]1.2 # 1.2
[single]"3,4" # 34
[single]"-5.6e-3" # -0.0056
```


Notes for the previous slide

1. If the locale is set to Estonian, the output will separate decimals with a comma
2. But comma in input would not work as expected
3. `[float]` is the same as `[single]`

Floats

[double] allows to store very large values

Example

```
[double]2e100 # 2E+100
[float]123123123123 # 1,231231E+11
```

Note

The values are stored for calculations. Not all digits are necessarily stored

Arrays

Arrays are list of elements. To create an array, separate elements by a comma

Example

```
"a", "b", "c"
3, "string", 4.5
1..20
@() # an empty array
@("element1", "element2")
```

Notes for the previous slide

1. Elements in one array may be of different types
2. Arrays with numbers can be created quicly by using first..last syntax. It works decendingly too
3. PowerShell usually understands when you want to create an array, but you can be explicit with the `@()` syntax. `@()` creates an empty array
4. Arrays are especially useful in loops

Time Stamps and Intervals

PowerShell has separate variable types for times which are useful in time calculations

Example

```
$start = [DateTime]"16 July 1969 13:32:00 GMT"
$moonLanding = [DateTime]"07/20/1969 8:17:40PM GMT"
$end = [DateTime]"July 24, 1969, 16:50:35Z"
$duration = $end - $start
$duration # [TimeSpan] object showing 8d 3h 18m 35s
(Get-Date) - $moonLanding # time since the landing
```

Notes for the previous slide

1. As with floats, the output depends on system locale settings
2. Note how the string formats are deliberately different. PowerShell will accept all these and some more it can understand
3. The example is based on the Apollo 11 mission
4. `[DateTime]` differences are `[TimeSpan]` objects
5. `(Get-Date)` returns an object with the current system time

Intervals

Example

```
$menRecord = [timespan]"2:01:39"
$womenRecord = New-TimeSpan -Hours 2 -Minutes 17
↪   -Seconds 1
($womenRecord - $menRecord).TotalSeconds # 922
```

Notes for the previous slide

1. Time spans can also be created with the `New-TimeSpan` cmdlet
2. In the last example, we don't print the whole object but only the `TotalSeconds` attribute from it. More on that in the next slides and seminars

Hashtables

Hashtables are data structures that store one or more key/value pairs

Example

```
$cool_car = @{
  model = "Shelby"
  make = "GT500"
  year = 1967
}
$cool_car.make # GT500
$cool_car # a table with keys and value
```

Notes for the previous slide

1. Values can be of any data type, even other hashtables
2. You can then call a single attribute from the hashtable

PSCustomObjects

PowerShell objects are perhaps the most powerful variables. Most cmdlets output PSCustomObjects and it's possible to use them as inputs to other cmdlets

Example

```
[PSCustomObject]@{
  model = "Shelby"
  make = "GT500"
  year = 1967
}
$cool_car.year # 1967
$cool_car # a table where columns are keys
```

Notes for the previous slide

1. PSCustomObjects are similar to hashtables and they are defined almost the same way
2. PSCustomObjects however work much better as array elements. More on that next seminar

Table of Contents

Arithmetic Operators

Operator	Description	Example	Output
+	Addition	10+25	35
-	Subtraction	23-8	15
*	Multiplication	4.5*3.4	15.3
/	Division	2/5	0.4
%	Modulus	7%3	1

Notes for the previous slide

1. These are quite self explanatory
2. Modulus, also called modulo operation, finds the remainder after division of one number by another

Comparison Operators

Comparison Operators return `True` or `False` and are used for comparing values

Operator	Description	Example	Output
eq	Equals	`"a" -eq "b"`	False
ne	Not equals	`1 -ne 2`	True
gt	Greater than	`1+2 -gt 4.5`	False
ge	Greater than or equals	`2 -ge 2`	True
lt	Less than	`6 -lt 4`	False
le	Less than or equals	`2 -le 5`	True
not	Negates the following	`-not (2 -le 5)`	False

Notes for the previous slide

1. You are encouraged to test on your own different operators and different objects or their attributes
2. You will need to use parenthesis around cmdlets with parameters

Assignment Operators

Operator	Description	Example
=	Simple assignment operator	`$a = 100` `$a # 100`

+=	Add AND assignment operator	`$b = 5`
		`$b += 4`
		`$b # 9`
		`$c = "x","y"`
		`$c += "z"`
		`$c # "x", "y", "z"`

-=	Subtract AND assignment operator	`$d = 5`
		`$d -= 2`
		`$d # 3`

Notes for the previous slide

1. You can increase and decrease the value of a number this way
2. You can also add elements to an array, but not remove
3. Adding also works with strings and appends them
4. In some rare cases, you might need to use *= and /=. They work too

Table of Contents

Aliases

The `Verb-Noun` syntax is recommended. For convenience, many cmdlets have pre-defined shorter aliases

Cmdlet	Alias(es)	Description
`Get-ChildItem`	`ls`, `dir`, `gci`	Lists all files and folders in the current or given folder
`Get-Content`	`cat`, `gc`, `type`	Prints the content of the target file
`Move-Item`	`mi`, `move`, `mv`	Moves files and folders
`Copy-Item`	`cpi`, `copy`, `cp`	Copies files and folders

Notes for the previous slide

1. You can also define your own aliases
2. It may be challenging to find a balance between the convenience of aliases and the readability of cmdlets' full names

Aliases

Cmdlet	Alias(es)	Description
`Remove-Item`	`del`, `rm`, `ri`	Deletes files or folders
`Rename-Item`	`rni`, `ren`	Renames a single file, folder or link
`Get-Command`	`gcm`	Lists available commands
`Select-String`	`sls`	Prints lines matching a pattern
`Set-Location`	`cd`, `sl`, `chdir`	Changes the working path
`Get-Process`	`gps`, `ps`	Lists running processes
`Stop-Process`	`spps`, `kill`	Stops a process

Aliases

Cmdlet	Alias(es)	Description
`Select-Object`	`select`	Selects object properties
`Sort-Object`	`sort`	Sorts objects by values
`ForEach-Object`	`foreach`, `%`	Performs operations against each item in a collection of input objects
`Where-Object`	`where`, `?`	Selects objects based on their property values

Remark

`Get-` may be omitted. See all aliases with `Get-Alias`

Notes for the previous slide

1. Some cmdlets even have one letter aliases
2. Since `Get-` may be omitted, even just `alias` will show the same listing of all aliases

Table of Contents

What Did We Learn?

- Different variable classes
- Arithmetic, comparison and assignment operators
- Aliases

2.3 Basics 2


Conditionals
Loops
Printing Output
Custom Functions
Splatting
Summary
Basics 2
PowerShell Course
TalTech
TalTech
Basics 2

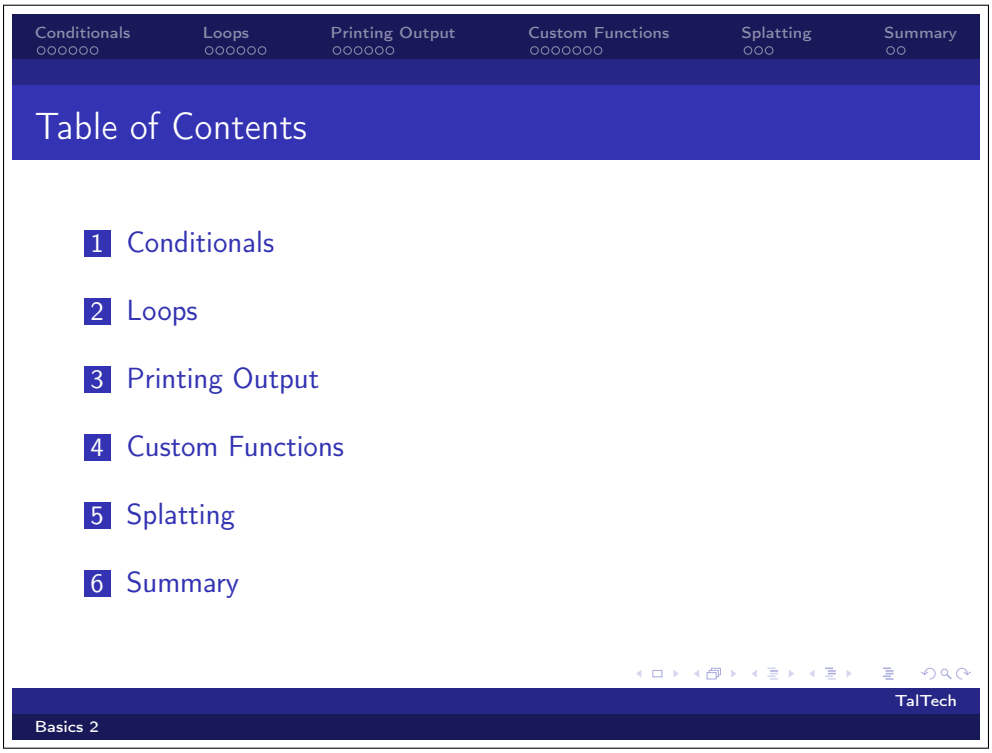

Conditionals
Loops
Printing Output
Custom Functions
Splatting
Summary
Table of Contents
1 Conditionals
2 Loops
3 Printing Output
4 Custom Functions
5 Splatting
6 Summary
TalTech
Basics 2

Table of Contents

Conditionals

Conditional statements (also expressions or constructs) are features which perform different actions depending on whether a boolean condition evaluates to True or False

If

Example

```
if ($True) {
  # this will always be run
}
if ($False) {
  # this will never be run
}
if ($var -gt 5) {
  # this will be run, if $var greater than 5
}
```


Notes for the previous slide

1. `$True` and `$False` return True and False. We could have also written some comparisons

Else

Example

```
if ($var -lt 10) {
  '$var is less than 10'
}
else {
  '$var is at least 10'
}
```


Notes for the previous slide

1. As soon as If is true, the intepreter does not look at Else
2. Notice the use of single quotation marks. Why?
3. Indention and new lines are recommended for readability but not needed for the interpreter
4. `If` and `Else` are again not case sensitive

Elseif

Example

```
if ((Get-Date).DayOfWeek -eq "Monday") {
  "Oh no! It's Monday. I hate Mondays..."
}
elseif ((Get-Date).DayOfWeek -eq "Friday") {
  "TGIF"
}
else {
  "It's neither Monday nor Friday"
}
```


Notes for the previous slide

1. As soon as If is true, the intepreter does not look at Elseif
2. As soon as Elseif is true, the intepreter does not look at Else

Switch

Example

```
switch ((Get-Date).DayOfWeek) {
  "Monday" { "Oh no! It's Monday. I hate Mondays..." }
  "Friday" { "TGIF" }
  default { "It's neither Monday nor Friday" }
}
```


Notes for the previous slide

1. The example above behaves the same as the previous one
2. Switch statement is useful, if we need to compare the same variable against different values. Note how in case of Elseif, the variable does not need to be the same

Table of Contents

While

Loops are useful for repetitive actions. The `While` loop is probably the simplest

Example

```
$i = 1
while ($i -lt 5) {
  "At iteration $i"
  $i++ # it's the same as $i += 1
} # outputs 4 lines
```

Notes for the previous slide

1. The same two lines of code are passed 4 times, i.e. until the condition is False for the first time
2. `While` loop is useful for looping until the boolean value of a condition changes

For

`For` loop is slightly more compact and thus more difficult to read. Yet is essentially does the same as `While` loop did

Example

```
for ($i=1
  $i -lt 5
  $i++) {
    "At iteration $i"
  }
```

Notes for the previous slide

1. Let's use the exact same example, just with a different loop

For

Example

```
for ($i = 1; $i -lt 5; $i++) {
  "At iteration $i"
}
```

Notes for the previous slide

1. Often the readability is better when the initial value, condition and repeat action are separated by a semicolon
2. A semicolon actually allows to have multiple commands on one line in general
3. `For` is best suited for scenarios where the iteration count is known before the loop starts

ForEach

`ForEach` in PowerShell is a bit tricky because it's a statement and an alias for a cmdlet `ForEach-Object`. Let's first consider the statement

Example

```
$fruits = "apples", "bananas", "pears"
foreach ($fruit in $fruits) {
  "I like $fruit"
} # outputs a line for liking each fruit
```

Notes for the previous slide

1. Defining variable `$fruits` is optional. It's also possible to place the array directly after `in`
2. `ForEach` allows the script writer to loop through arrays

ForEach-Object

`ForEach-Object` cmdlet is useful for piping output to it. Let's refactor the previous example

Example

```
$fruits = "apples", "bananas", "pears"
$fruits | ForEach-Object { "I like $_" }
```

Notes for the previous slide

1. Refactoring is changing code without chaning behaviour
2. We are using `$_` which is the variable that will hold the individual array element value at each iteration
3. You could use carriage returns around curly brackets
4. `ForEach-Object` is usually slower than `ForEach` but uses less memory in case of extremely large input objects

Table of Contents

Write-Output

`Write-Output` is the most common way of outputting text. It can be piped forward to the next cmdlet, but it will be printed, if the line ends

Example

```
"Hello World"
echo "Hello World"
Write-Output "Hello World"
Write-Output "Hello World" | Out-Default
```


Notes for the previous slide

1. All lines do the same but they are gradually more explicit
2. `echo` is an alias and `Out-Default` is always implied, even if not written

Write-Host

`Write-Host` should be used to log info immediately to the console

Example

```
if ($var -gt 100) {
  Write-Host "Value too high" -ForegroundColor Red
}
else {
  Write-Host "Task finished" -ForegroundColor Green
}
```


Notes for the previous slide

1. `Write-Host` will not provide input to following cmdlets
2. The difference is a nuance but important to note
3. You can also format the color of output

Out-File

It's often useful to send output into a file, e.g. log events for scheduled tasks

Example

```
"First line" | Out-File -FilePath "File 1.txt"
"Second line" | Out-File -Append -FilePath "File
↪  1.txt"
Get-Process | Out-File -FilePath
↪  "C:\temp\Processes.txt"
"First line" > "File 2.txt" # not recommended
"Second line" >> "File 2.txt" # not recommended
compare (gc "File 1.txt") (gc "File 2.txt") # $null
```


Notes for the previous slide

1. Relative `FilePath` value (not starting with `C:\` or `\\` or similar) will be the file or path starting from the current working directory
2. `Out-File` overwrites the previous content by default. `Append` is needed for adding content
3. The last command will write all running processes to a file in the temp folder. Note that the folder must exist
4. `File 1.txt` and `File 2.txt` will be identical, but redirection with `>` is not the PowerShell native way
5. `Compare-Object` shows the line-by-line differences of objects. No output means that the objects are identical. More on this in the next seminar

Tee-Object

Example

```
"Test" | Tee-Object -Variable var # "test" is printed
↪  and saved to variable $var
"Test" | tee -FilePath file.txt # "test" is printed
↪  and saved to file file.txt
"Test" | tee -FilePath file.txt -Append # the same but
↪  appended
```


Notes for the previous slide

1. Tee comes from the capital letter T which splits the vertical line at the top
2. It's useful for monitoring output and saving it at the same time
3. Values are stored only after the whole command has finished. This is important when running time consuming commands
4. Both `-FilePath` and `-Variable` cannot be used at the same time

clip

`clip` is not a PowerShell cmdlet, it's from `cmd`

Example

```
Get-ChildItem | clip # places current working
↪  directory listing to clipboard
```


Notes for the previous slide

1. Actually all `cmd` commands work in PowerShell too. Try for example `ipconfig`

Table of Contents

Functions

Functions or subroutines are a series of instructions to perform a specific task

Example

```
function Write-ProcessFile {
  $time = Get-Date -Format yyyyMMdd-hhmmss
  $path = "C:\logs\$time processes.txt"
  $ps = Get-Process
  "Created at $time" | Out-File -FilePath $path
  Out-File -InputObject $ps -FilePath $path -Append
}
Write-ProcessFile
```

Notes for the previous slide

1. Once you define the function, you can just call its name and all the defined content will be done

Remark

- `Verb-Noun` syntax is recommended but not mandatory
- You can see all recommended verbs with `Get-Verb`

Functions with Arguments

User-defined functions get more powerful with arguments

Example

```
function Get-Factorial {
    $counter = $args[0]
    $factorial = 1
    while ($counter) {
      $factorial *= $counter--
    }
    $factorial
}
Get-Factorial 6 # 720 because 1*2*3*4*5*6=720
```


Notes for the previous slide

1. `$args[0]` stands for the first positional argument
2. `*=` and `$counter-` should be familiar

Functions with Multiple Arguments

Example

```
function Get-Rectangle {
  $circumference = 2 * ($args[0] + $args[1])
  $area = $args[0] * $args[1]
  [PSCustomObject]@{ Circumference = $circumference
    Area = $area }
}
Get-Rectangle 4 8
# Circumference Area
# ------------- ----
#            24   32
```


Functions with Named Arguments

Example

```
function Get-Current {
  Param([float]$Voltage, [float]$Resistance)
  $current = $Voltage / $Resistance
  "$current A"
}
Get-Current -Voltage 4.5 -Resistance 2 # 2.25 A
```

Notes for the previous slide

1. Parameters are defined within `Param()`

User Input

`Read-Host` allows to ask for user input

Example

```
$name = Read-Host "What's your favourite scripting
↪  language?"
if ($name -eq "PowerShell") {
  "As expected: PowerShell!"
}
else {
  "We'll see if we can do something about you still
  ↪  liking $name"
}
```

Table of Contents

Splatting

Splatting is passing parameters to a cmdlet as a hashtable

Example

```
$parameters = @{
  Voltage = 12
  Resistance = 5
}
Get-Current @parameters # 2.4 A
```

Notes for the previous slide

1. Notice the `@` instead of `$`

Splatting

Example

```
$text = "Success!"
$other_parameters = @{
  Object = $text
  ForegroundColor = "Green"
}
Write-Host @other_parameters -BackgroundColor
↪  DarkYellow
```

Notes for the previous slide

1. Hashtable items can contain other variables or even cmdlets and functions
2. You can splat some and manually specify other parameters

Table of Contents

What Did We Learn?

- `If`, `Elseif`, `Else`
- `While`, `For`, `ForEach`
- `Write-Output`, `Write-Host`
- Functions
- Splatting

2.4 Basics 3

Basics 3

PowerShell Course

TalTech

Table of Contents

Filtering and Formatting Strings Arrays Hashtables PSCustomObjects CSV and JSON Getting Help Summary

Table of Contents

1 Filtering and Formatting
2 Strings
3 Arrays
4 Hashtables
5 PSCustomObjects
6 CSV and JSON
7 Getting Help
8 Summary

TalTech
Basics 3

Select-Object

`Select-Object` selects objects or object properties

Example

```
[TimeSpan]"1:2:3" # TimeSpan object with many
↪  attributes
[TimeSpan]"1:2:3" | Select-Object Hours, Minutes
# Hours Minutes
# ----- -------
# 1         2
Get-Process | select * # elaborate data
```


Notes for the previous slide

1. `Select-Object *` force selects all attributes from the input object
2. `select` is an alias for `Select-Object`

Where-Object

`Where-Object` filters objects from a collection based on their property values

Example

```
Get-Service | Where-Object { $_.Status -eq "Stopped" }
ls | where { $_.Name -match "a" -and $_.Length -ge
↪  100kB }
ls | ? { $_.CreationTime -ge (Get-Date).AddDays(-1) }
```


Notes for the previous slide

1. The first example lists all stopped services
2. The second example lists all files with "a" in the name and size at least 100 kB
3. Note the use of aliases
4. The third example shows object created within the last 24 h

Sort-Object

`Sort-Object` sorts objects by property values

Example

```
"b", "c", 1, "a" | Sort-Object # 1, "a", "b", "c"
ls | sort -Property Length -Descending
1..5 + 3..7 | sort -Unique # 1..7
```


Notes for the previous slide

1. The second example sorts files by size decreasingly

Group-Object

`Group-Object` groups objects that contain the same value for specified properties

Example

```
Get-Process | Group-Object -Property Responding
ls | group Attributes
ls -File | group Extension | % {
  "There are $($_.Count) $($_.Name)'s"
}
```


Notes for the previous slide

1. `Group-Object` returns an object where similar elements are bundled
2. It allows to quickly see the count and loop through the elements of a kind
3. We need to use the `$($_.Attribute)` syntax in strings
4. For all of these cmdlets, `-Object` is optional

Compare-Object

`Compare-Object` compares two sets of objects

Example

```
$array1 = "a", "c", 1
$array2 = "a", "b", 2
Compare-Object $array1 $array2
# InputObject SideIndicator
# ----------- -------------
# b           =>
# 2           =>
# c           <=
# 1           <=
```


Notes for the previous slide

1. The positional parameters are for `-ReferenceObject` and `-DifferenceObject` respectively
2. Matching elements can be included with `-IncludeEqual`

Tables and Lists

Whenever PowerShell outputs data, it tries to decide the best format for the console. In some cases, it might be useful to format it manually. Let's focus on three options

- `Format-Table`
- `Format-List`
- `Out-GridView`

Format-Table

`Format-Table` is used by default if the output can be fitted into columns within the console width

Example

```
Get-Process # prints processes
Get-Process | Format-Table # same output
Get-Process | Format-Table -Property ProcessName, Id,
↪  Responding, MainModule -Wrap
```

Notes for the previous slide

1. The last one prints specific columns and wraps long lines

Format-List

`Format-List` is used by default if the output cannot be fitted into columns within the console width

Example

```
Get-Process # prints processes in columns
Get-Process | Format-List # same data as a list
Get-Process | Format-List -Property * # elaborate
```

Note

Output is not suitable for piping

Notes for the previous slide

1. The last one prints everything Get-Process gets about processes
2. `Format-` cmdlets should be used as the last commands. Their output is not suitable for piping

Out-GridView

`Out-GridView` creates a graphical window with a minimalistic interface to sort and filter the data

Example

```
Get-Service | Out-GridView
Get-Service | Select-Object * | Out-GridView
```

Note

The cmdlet is only available in the GUI versions of Windows

Notes for the previous slide

1. `Out-GridView` does not have `-Property` parameter

Table of Contents

String Methods

A method is a procedure associated with a class. Strings are a class

Example

```
"Test" | Get-Member
"Test".ToUpper() # TEST
"Car".Replace("C", "B") # Bar
"a.b,c.d".Split(",.") # @("a", "b", "c", "d")
" padded  ".Trim() # padded (without spaces)
"slaughter".Substring(1, 5) # laugh
"ABC".Remove(1, 1) # AC
"PowerShell".IndexOf("e") # 3
"PowerShell"[-1] # l
```


Notes for the previous slide

1. `Get-Member` can be used with any object to see its type and available methods
2. The characters are *case sensitive*
3. `@()` is not printed
4. Method `.Split()` takes one argument or considers each character as a separator
5. First character is at position 0
6. Omitting the second paramter for `.Substring()` will return the string until the end
7. Trimming is useful for handling any input — user input, poor CSV files, etc.
8. Number in square brackets outputs the character at that position. Use `[-1]` to get the last position

Concatenation

To concatenate means to link (things) together in a chain or series

Example

```
$first_name = "Elvis"
$last_name = "Presley"

$first_name + " " + $last_name # Elvis Presley
"$first_name $last_name" # Elvis Presley

$array = @($first_name, $last_name)
"$array" # Elvis Presley
```


Notes for the previous slide

1. Concatenation is just a fancy word for linking things together in a chain
2. Three examples of achieving the same result
3. `@()` is optional, here we're just explicit about creating an array

String Operators

Operators are somewhat similar to methods

Example

```
"abcd" -split "bc" # @("a", "d")
"Visit them" -replace "them", "us" # Visit us
"book" -ireplace "B", "C" # Cook
"book" -creplace "B", "C" # book
"cool" -match "oo" # True
"abc123" -match "a.1" # False
"abc123" -match "a..1" # True
"PowerShell" -like "*Shell" # True
```


Notes for the previous slide

1. Operator `-split` takes the whole string as one parameter
2. `i` means case *insensitive* which is also the default
3. `c` means case *sensitive*
4. Operator `-match` supports regular expression which is complicated but powerful
5. `.` means any single character
6. Operator `-like` supports the wild card character `*`

-f Format Operator

Example

```
$number = 123
"The most important number is {0}" -f $number
# The most important number is 123

"Your username is {0}" -f $env:USERNAME

$array = "cats", "dogs", "mice"
"My favourite pets are {0}, {1} and {2}" -f $array
# My favourite pets are cats, dogs and mice
```


Notes for the previous slide

1. `$env:` provides access to environment variables

-f Format Operator

Example

```
"{0} double is {1}" -f $number,($number*2)
# 123 double is 246
"{0} in hex is {0:x} and percentage {0:p}" -f $number
# 123 in hex is 7b and percentage 12 300,00%
"'{0}' padded to 5 characters is '{0,5}'" -f $number
# '123' padded to 5 characters is '  123'
"The same can be done for hex: {0,5:x}" -f $number
# The same can be done for hex:    7b
"Custom time: {0:HH}.{0:mm}/{0,4:ss}" -f (Get-Date)
# Custom time: 15.42/  04
```


Notes for the previous slide

1. :x is for the hexadecimal value of the integer
2. Number after the comma pads but does not limit the length
3. There are many more format strings

Table of Contents

Array Operators

Example

```
1..4 -join "" # 1234
-join @(1..4) # 1234
1..4 -join "-" # 1-2-3-4
10, 15, 20 -gt 14 # 15, 20
"ab", "bc", "cd" -match "b" # "ab", "bc"
"ab", "bc", "cd" -notmatch "c" # "ab"
4 -in 1..6 # True
"Shell" -in "Windows", "PowerShell" # False
"Windows", "PowerShell" -contains "Windows" # True
```

Notes for the previous slide

1. `-match`, `-like` and others also have corresponding `-not` operators
2. `-in` and `-contains` search for an exact match only

Looping through Elements

Example

```
$cars = "Ford", "Audi", "BMW"
$cars | % { $_.ToUpper() } # "FORD", "AUDI", "BMW"

foreach ($car in $cars) {
  if ($car -notmatch "o") {
    "I like $car"
  }
} # will not print "I like Ford"

$cars[1] # Audi
```

Notes for the previous slide

1. % is an alias for `ForEach-Object`
2. `$array[$number]` allows to access a certain element in the array

Nested Loops

Example

```
$cars = "Ford", "Audi", "BMW"
$pronouns = "I", "You"
$cars | % {
  $car = $_
  $pronouns | % { "$_ like $car" }
}
```

Note

Be careful when nesting `ForEach-Object` loops. Old `$_` gets overwritten in new loops

Notes for the previous slide

1. A combination of `ForEach-Object` and `foreach` is also of course possible
2. Yet as soon as you create another `ForEach-Object` or even `Where-Object` you overwrite the value of `$_`

Table of Contents

Hashtables

Example

```
$planets = @{
  Earth = @{
    Mass = 5.9722E24
    Radius = 6378.1
  }
  Mars = @{
    Mass = 6.39E23
    Radius = 3396.2
  }
}
```


Notes for the previous slide

1. Let's define a hashtable to use as an example
2. Note that we have two hashtables in one
3. Mass is in kg and equatorial radius in km

Hashtables

Example

```
$planets.Mars.Mass # 6,39E+23
$planets["earth"]["Mass"] # 5,9722E+24
$planets.Keys # "Earth", "Mars"
$planets.Mars.Keys # "Radius", "Mass"
$planets["Earth"].Values # 6378,1, 5,9722E+24
$planets.Count # 2
```


Notes for the previous slide

1. Values can be access via either putting a dot in between keys or the keys in square brackets

Adding and Using Values

Example

```
$planets = $planets + @{
  Venus = @{
    Mass = 4.8675E24
    Radius = 6051.8
    Volume = 9.2843E11
  }
}
$planets.Earth.Add("Volume", 1.08321E12)
$planets.Keys | foreach { $planets.$_.Mass } | measure
↪  -Average -Sum
```


Notes for the previous slide

1. There are different ways to add values
2. Volume is in cubic km
3. `measure` is an alias for `Measure-Object`

Create a Hashtable Dynamically

Example

```
$snapshot = @{}
foreach ($service in Get-Service) {
  $snapshot.Add($service.Name, $service.Status)
}
```


Notes for the previous slide

1. `@{}` creates an empty hashtable

Table of Contents

Creating PSCustomObjects

PSCustomObjects are similar to hashtables and can be created from them

Example

```
# In PowerShell v2
New-Object -TypeName PSObject -Property @{
  Name = "John"
  Age = 23
}
# An alternative since PowerShell v3
[PSCustomObject]@{ Name = "John"; Age = 23 }
```

Notes for the previous slide

1. The commands yield in essence identical objects
2. Don't mind the new lines. Either a semicolon or a new line start a new attribute in both cases

Hashtables and PSCustomObjects in Arrays

Example

```
$hash_array = @()
$hash_array += @{ Friend = "John"
  Age = 23 }
$hash_array += @{ Friend = "Mary"
  Age = 21 }
$hash_array # Two columns --- Name (Key) and Value
```

Hashtables and PSCustomObjects in Arrays

PSCustomObjects behave better in arrays

Example

```
$psobject_array = @()
$psobject_array += [PSCustomObject]@{ Friend = "John"
  Age = 23 }
$psobject_array += [PSCustomObject]@{ Friend = "Mary"
  Age = 21 }
$psobject_array # Two columns --- Friend and Age
```


Table of Contents

CSV and JSON

- CSV stands for comma separated values, although they may be semicolon or some other character separated as well
- JSON is JavaScript object notation, often used in web development

Output to CSV Files

Example

```
Get-Process | Export-Csv -Path Processes.csv
Get-Process | Export-Csv -Path Processes.csv
↪  -Delimiter ";" -NoTypeInformation
Get-Process | select Name, Path, Responding, Id |
↪  Export-Csv -Path Processes.csv -Delimiter ";"
↪  -NoTypeInformation
```

Note

The output of `Format-Table` is not suitable for `Export-Csv` input

Notes for the previous slide

1. The first example is very simple, but has problems
2. The second example does not have an extra line in the beginning and opens better in Excel where the regional delimited is a semicolon instead of a comma
3. The third example limits the columns to what we might actually need
4. The third example limits the columns to what we might actually need
5. `Format-Table` output should not be used for piping forward

Input from CSV files

Example

```
"Name;Symbol;Atomic number
Helium;He;2
Carbon;C;6
Iron;Fe;26" | Out-File -Path Elements.csv
$elements = Import-Csv Elements.csv -Delimiter ";"
$elements # Prints the created PSCustomObject
```

ConvertTo-Csv and ConvertFrom-Csv

Example

```
"Name;Symbol;Atomic number
Helium;He;2
Carbon;C;6
Iron;Fe;26" | ConvertFrom-Csv -Delimiter ";"

Get-Process powershell | select Name, Id |
↪  ConvertTo-Csv -NoTypeInformation -Delimiter ";"
```


Input from JSON

Example

```
$todo = Invoke-RestMethod -Uri
↪  "https://jsonplaceholder.typicode.com/todos"
$users = Invoke-RestMethod -Uri
↪  "https://jsonplaceholder.typicode.com/users"

$todo | where { -not $_.completed }
```

Note

There are no `Import-Json` and `Export-Json`

Notes for the previous slide

1. https://jsonplaceholder.typicode.com provides free service with dummy data
2. There are also free services for e.g. weather and financial markets' data
3. The URL provides just text but PowerShell will recognise the JSON format automatically and convert it to a PSCustomObject
4. The cmdlet again outputs a PSCustomObject

ConvertTo-Json and ConvertFrom-Json

Example

```
ps powershell | select Name, Id | ConvertTo-Json
#[
#  {
#    "Name":  "powershell",
#    "Id":  19788
#  }, {
#    "Name":  "powershell",
#    "Id":  29156
#  }
#]
```

Notes for the previous slide

1. These work just like their CSV alternatives
2. In more advanced examples, PowerShell objects can be converted to JSON text and submitted to REST APIs with the POST method
3. JSON is similar to but less verbose to XML

Table of Contents

Getting Help

There is a help page for each command

Example

```
Get-Help -Name Get-ChildItem
help ls
help Where-Object -Examples
help sort -Full
help ps -Online
Update-Help
```


Notes for the previous slide

1. `-Name` can be a positional parameter
2. `help` is an alias. Remember that `Get-` is always optional
3. `-Online` opens the documentation in the default browser
4. `Update-Help` downloads the latest documentation to local cache

Table of Contents

What Did We Learn?

- Filtering and formatting
- String and array methods and operators
- Hashtables and PSCustomObjects
- CSV and JSON objects in PowerShell
- Accessing cmdlets' manuals

2.5 Windows OS Management

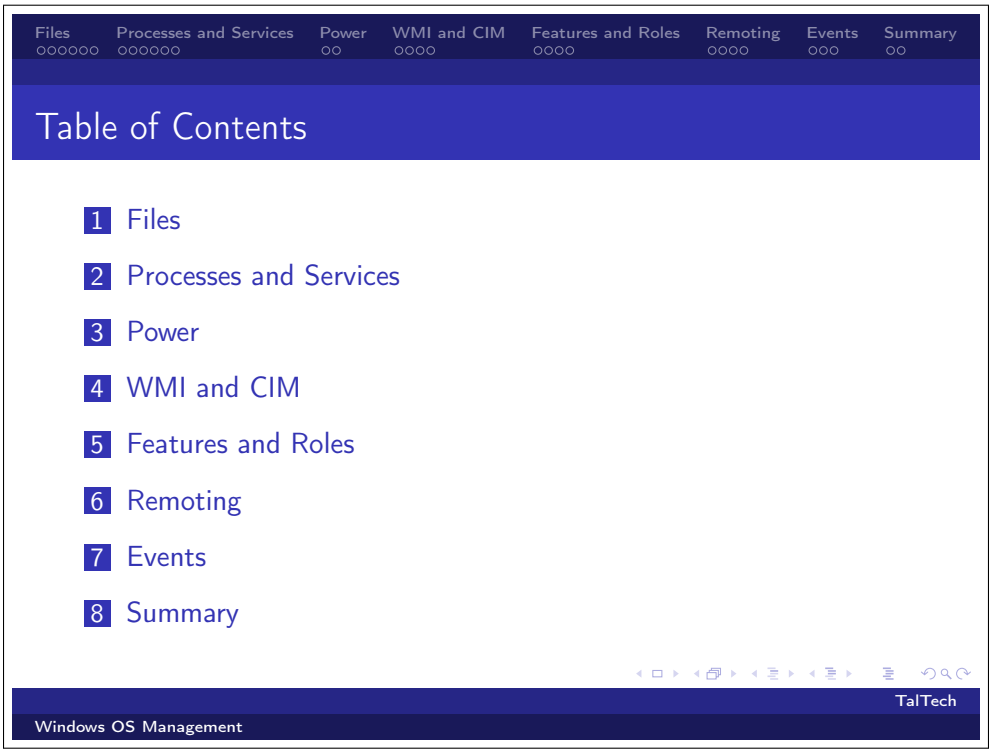


Table of Contents

Navigating

Example

```
Get-Location # prints the working directory
pwd # alias to the above (present working directory)
Set-Location -Path "D:\"
cd \
sl "HKLM:"
sl ~ # ~ is the same as $env:USERPROFILE or $HOME
```

Notes for the previous slide

1. `cd` and `sl` are aliases for `Set-Location`
2. `HKLM:` allows to navigate into the Windows Registry

Listing Files and Folders

Example

```
Get-ChildItem # lists files and folders in pwd
Get-ChildItem -Path "C:\" -Name # lists only names
dir -Attributes ReadOnly # lists only read-only files
dir "HKLM:\SOFTWARE\" # lists the "software" hive
ls *.txt -Recurse # .txt files in all subfolders too
ls -File "*a*" # lists files with "a" in the names
ls -Directory -Recurse # lists folder structure
ls -Recurse | % { $_.FullName } # shows full paths
ls | select * # shows all available attributes
(ls).Count # returns the count of files and folders
```

Notes for the previous slide

1. Items have many attributes. See them all by selecting all
2. It's often useful to combine the output with `Where-Object`, but the preferred (faster) way is to filter output already with `Get-ChildItem` parameters
3. `$_.FullName` is the full path
4. Almost all objects have the `.Count` attribute

Saving Text to Files

Example

```
$txt = "abc"
$path = "C:\temp\test.txt"
$txt | Out-File -FilePath $path # UTF-16 encoding
$txt | Out-File -Encoding utf8 $path # UTF-8 BOM
[IO.File]::WriteAllLines($path, $txt) # UTF-8
```

Notes for the previous slide

1. We will not discuss encodings in depth, but be aware that PowerShell outputs UTF-16 by default
2. UTF-8 can be used easily by setting the `-Encoding` parameter
3. Disabling the byte-order-mark is more complicated
4. The last example uses .NET methods which PowerShell supports but are not covered in this course
5. Some other programmes might be picky about the encoding of input files

Copy, Move, Rename and Delete

Example

```
Copy-Item -Path File1 -Destination File2
cp Some_folder Folder_copy -Recurse
Move-Item -Path File1 -Destination File2
mv File1 File2 -WhatIf
Rename-Item -Path File1 -NewName New_and_better_name
ren File1 New_and_better_name
Remove-Item -Path Useless_file
del *.txt -Confirm
```

Notes for the previous slide

1. Simple examples
2. All these cmdlets have `-Confirm` and `-WhatIf` parameters to make their uses more involved

Access-Control Lists

Example

```
Get-Acl File.txt
(Get-Acl File.txt).Access

Get-Acl -Path File1.txt | Set-Acl -Path File2.txt
```

Notes for the previous slide

1. An ACL is a list of permisisons attached to an object, i.e. who can do what with the object
2. The last example copies the ACL of `File1.txt` to `File2.txt`
3. ACLs are not limited to file ACLs. Many objects in Windows and Microsoft products have an ACL

Table of Contents

Get-Process

Example

```
Get-Process
Get-Process -Name powershell, explorer
Get-Process -IncludeUserName # requires administrator
↪  permissions
Get-Process -IncludeUserName | Group-Object UserName
```


Notes for the previous slide

1. We have used this cmdlet as an example many times before
2. Some cmdlets or their parameters require the shell to be started in elevated user rights

Stop-Process

Example

```
Stop-Process -Name "notepad"
Stop-Process -Id 4214 -Confirm
kill -Name chrome -Force
```


Get-Service

Example

```
Get-Service
Get-Service -Name "net*"
Get-Service -DisplayName "Windows *"
"net*" | service
service | where { $_.Status -eq "Stopped" }
service | group Status
service | measure
```

Notes for the previous slide

1. Windows service is a computer program that operates in the background
2. Positional parameter is `-Name` not `-DisplayName`

Set-Service

Note

This cmdlet requires elevated permissions

Example

```
Set-Service -Name "lanmanworkstation" -DisplayName
↪  "LanMan Workstation"
Set-Service -Name W32Time -StartupType Disabled
Set-Service -Name "WinDefend" -Status Running
Set-Service -Name "Dhcp" -Status Stopped
```

Notes for the previous slide

1. Due to the permissions requirement, this cannot be tested on classroom PCs
2. The default name is Workstation
3. The second one disables time synchronisation
4. The third starts Windows Defender Antivirus
5. The fourth stops DHCP

Start-Service, Stop-Service and Restart-Service

These are alternatives to using `Set-Service -Status`

Example

```
Start-Service -Name Appinfo
Get-Service -DisplayName "Windows *" | Restart-Service
$service = Get-Service -Name W32Time
Stop-Service -InputObject $service
```

Notes for the previous slide

1. Again, administrator permissions are needed
2. You can call the cmdlets in a loop, if the input is an array of services or services' names

Table of Contents

Stop-Computer and Restart-Computer

Example

```
Stop-Computer
Stop-Computer -Force
Stop-Computer -ComputerName "server1", "client2"
Restart-Computer
Restart-Computer -Force
Restart-Computer -ComputerName "server1", "localhost"
```

Note

Be careful not to interrupt important processes or other users

Notes for the previous slide

1. `-Force` shuts down even when other users are still logged on
2. `-Computer` may include other computers on the network where the logged on account is a local administrator

Table of Contents

WMI and CIM

- Windows Management Instrumentation (WMI) provides an operating system interface through which instrumented components provide information and notification. WMI is Microsoft's implementation of the Common Information Model (CIM) standards
- The Common Information Model (CIM) is an open standard that defines how managed elements in an IT environment are represented as a common set of objects and relationships between them

Notes for the previous slide

1. In essence, these are tools to interact with the operating system and the hardware on a low level

Get-WmiObject

Example

```
Get-WmiObject -Class Win32_Bios
Get-WmiObject -Class Win32_Bios | select *
Get-WmiObject Win32_Service
gwmi Win32_processor
gwmi Win32_Share -Computer "server1"
gwmi -List
```

Notes for the previous slide

1. The first example gets BIOS info
2. The third line is similar to `Get-Service`
3. `Win32_processor` is for processor info
4. Next shows shares on `server1`
5. `-List` shows all available classes

CIM

Example

```
Get-CimInstance -Class Win32_Bios
Get-CimClass # lists all classes
Get-CimClass Win32_Service
(Get-CimClass Win32_Service).CimClassMethods
```

Note

CIM is preferred over WMI

Notes for the previous slide

1. CIM provides similar functionality but it is a more modern approach
2. `Get-WmiObject` should be replaced by `Get-CimInstance`
3. WMI is still widely used

Table of Contents

Get-WindowsFeature

Example

```
Get-WindowsFeature
Get-WindowsFeature -Name "*AD*"
Get-WindowsFeature | where { $_.InstallState -eq
↪  "Installed" }
```


Notes for the previous slide

1. The cmdlet shows what is currently installed and generally available
2. A role is a set of software programs that let a computer perform a specific function for multiple users or other computers within a network, e.g. DHCP or Active Directory
3. Features are software programs that support the functionality of one or more roles, e.g. tools like the Telnet client

Install-WindowsFeature

Example

```
Install-WindowsFeature -Name AD-Domain-Services
Install-WindowsFeature BitLocker, Web-Ftp-Server
Install-WindowsFeature -Name Hyper-V -Restart
Install-WindowsFeature Hyper-V -Computer "server1"
Install-WindowsFeature -Name Web-Server
↪  -IncludeAllSubFeature -IncludeManagementTools
```


Notes for the previous slide

1. `Name` can be named or positional
2. Arrays are accepted
3. `-Restart` restarts automatically, if it is needed
4. The last example installs IIS with all subfeatures and tools
5. `Add-WindowsFeature` is an alias

Uninstall-WindowsFeature

Example

```
Uninstall-WindowsFeature -Name AD-Domain-Services
Uninstall-WindowsFeature BitLocker, Web-Ftp-Server
Uninstall-WindowsFeature -Name Hyper-V -Restart
Uninstall-WindowsFeature Hyper-V -Computer "server1"
```


Notes for the previous slide

1. This cmdlet works the same way as `Install-WindowsFeature`
2. `Remove-WindowsFeature` is an alias
3. These are server-only cmdlets

Table of Contents

-ComputerName

Many commands have the `-Computer` parameter which runs the command on the target computer over the network

Example

```
Stop-Computer -ComputerName "server1"
gwmi Win32_Share -ComputerName "server1"
Install-WindowsFeature Hyper-V -ComputerName "server1"
```

Notes for the previous slide

1. We have already used the `-Computer` and `-ComputerName` parameters in previous examples

Invoke-Command

Example

```
Invoke-Command -ScriptBlock { ls }
Invoke-Command -ComputerName "server1" -ScriptBlock {
  ls "C:\"
}
Invoke-Command -FilePath "c:\scripts\test.ps1"
↪  -ComputerName "client1"
$targets = Get-Content Machines.txt
$command = [ScriptBlock]{ (Get-Host).Version }
icm -ComputerName $targets -ScriptBlock $command
```

Notes for the previous slide

1. `Invoke-Command` provides much more flexibility
2. The first command is the same as running just `ls`
3. ComputerName may be an array
4. `[ScriptBlock]` is a type we did not cover before, but it can contains commands that `Invoke-Command` takes as input

Sessions

Example

```
$session = New-PSSession "server1"
Invoke-Command -Session $session -ScriptBlock {
  hostname
} # server1

Enter-PSSession $session
hostname # server1
```

Notes for the previous slide

1. After running `Enter-PSSession` all commands are run as if they are run locally on server1

Table of Contents

Get-EventLog

Example

```
Get-EventLog -LogName System

$parameters = @{
  LogName = "Application"
  EntryType = "Error"
  After = (Get-Date).AddDays(-30)
}

Get-EventLog @parameters
```


Notes for the previous slide

1. `LogName` is mandatory
2. The interfaces with the Windows Event Log technology
3. On GUI, you could use the Event Viewer
4. This cmdlet can be used without GUI and for automating
5. Let's remind ourselves how splatting works

Custom Event logs

Example

```
New-EventLog -LogName 'My Custom Log' -Source
↪  'Important Script'

$event = @{
  LogName = "My Custom Log"
  Source = "Important Script"
  EventId = 3
  Message = "Test message"
}
Write-EventLog @event
```


Notes for the previous slide

1. While `Get-EventLog` is useful for reading events, we might want to create our own events
2. This requires elevated permissions
3. First, we need to create a log. Both `-LogName` and `-Source` are needed
4. Then we can start logging message with `Write-EventLog`

Table of Contents

What Did We Learn?

- File management
- Processes and services management
- Power management
- WMI and CIM
- Installing and removing features and roles
- Remote commands and sessions
- Reading and writing events

2.6 Active Directory

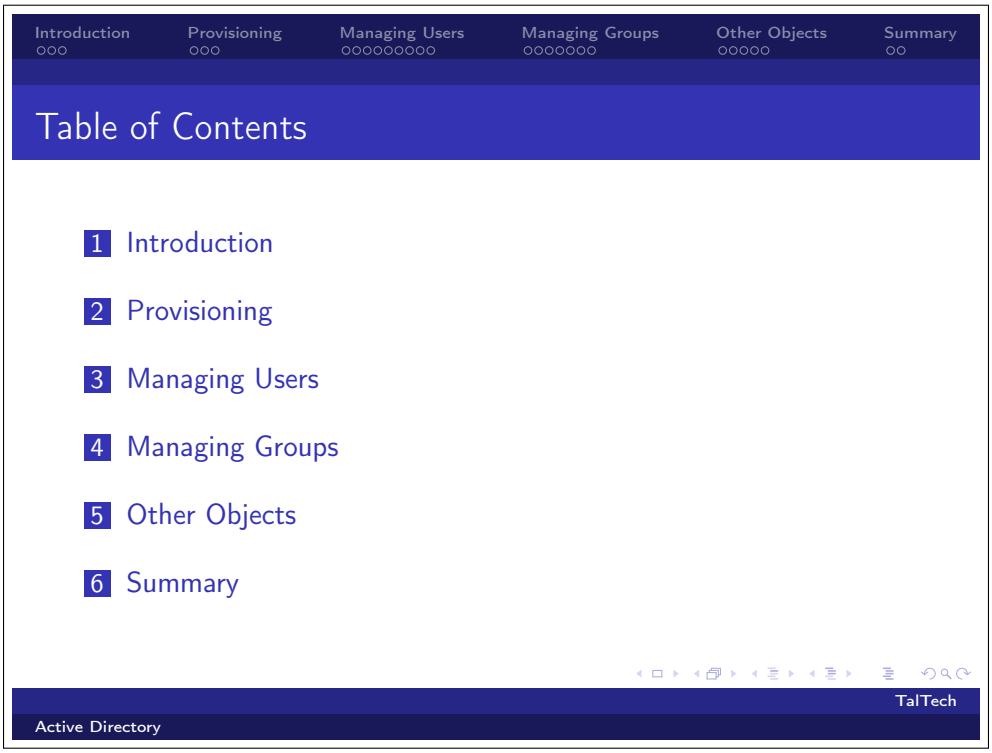


Table of Contents

What is Active Directory?

- Active Directory (AD) is a directory service
- It stores data about items (most importantly users, computers, groups, printers) like a database and replies to queries about this data
- For example, it is used to authenticate users when they log on
- A server hosting Active Directory Domain Services (AD DS) is a domain controller
- All Active Directory PowerShell cmdlets have the `*-AD*` syntax

Notes for the previous slide

1. Implementing direcotry services is virtually inevitable in a corporate environment

AD Cmdlets

Example

```
Get-Command -Module ActiveDirectory
Get-Command -Module ActiveDirectory -Verb Get
```

Notes for the previous slide

1. There are many commands. We will cover only the very basics

Table of Contents

Install AD DS and Create a New Forest

Example

```
Install-WindowsFeature AD-Domain-Services

$password = ConvertTo-SecureString -String
↪  "TalTech123" -AsPlainText -Force

Install-ADDSForest -DomainName pscourse.local
↪  -SafeModeAdministratorPassword $password
```


Notes for the previous slide

1. $password is a secure string object
2. The commands make the machine a domain controller
3. It is unlikely that you will need to go through this initial process in the real life yourself
4. After a restart, you need to log in with your domain administrator account

Browsing the New Domain

Example

```
Get-ADForest
Get-ADDomain
Get-ADDomain -Server pscourse

Get-ADUser -Filter *
Get-ADGroup -Filter *
Get-ADComputer -Filter *
Get-ADObject -Filter *
```


Notes for the previous slide

1. `Get-ADForest` will show info about the forest
2. `Get-ADDomain` will show info about the domain
3. All AD cmdlets have `-Server` parameter which can be used in a multi-domain environment
4. The other cmdlets list objects
5. Some of the output might be too confusing, but some can be discussed in the seminar

Table of Contents

Basic User Account Management

Example

```
New-ADUser -Name John

Get-ADUser John

Remove-ADUser John
```

Notes for the previous slide

1. These are the basic cmdlets to create, browse and remove a user
2. Note the long code in the SID attribute. This is the unique ID that the service uses to identify objects (not the name or anything else)
3. By default, users are created in the `disabled` status
4. Removal needs confirmation

User Account Creation

Example

```
$password = ConvertTo-SecureString -String
 ↪  "ChangeMeNOw" -Force -AsPlainText
New-ADUser -Name "Mary Poppins" -GivenName "Mary"
 ↪  -Surname "Poppins" -SamAccountName "mary.poppins"
 ↪  -AccountPassword $password -Country "UK" -City
 ↪  "London" -StreetAddress "17 Cherry Tree Lane"
 ↪  -EmployeeNumber 123456 -Title "Nanny" -Enabled
 ↪  $true
Get-ADUser mary.poppins | select *
Get-ADUser mary.poppins -Property *
```

Notes for the previous slide

1. There are many parameters to this command
2. Just `select *` will reveal more but not all attributes
3. We need to query all properties to see them

User Account Creation by Splatting

Example

```
$user = @{ Name = "Mary Poppins"
      GivenName = "Mary"
        Surname = "Poppins"
 SamAccountName = "mary.poppins"
AccountPassword = $password
        Country = "UK"
           City = "London"
  StreetAddress = "17 Cherry Tree Lane"
 EmployeeNumber = 123456
          Title = "Nanny"
        Enabled = $true }
```

Notes for the previous slide

1. Splatted arguments are easier to read
2. Now you can run `New-ADUser @user`

User Account Management

Example

```
New-ADUser sherlock.holmes
Set-ADUser -Identity sherlock.holmes -DisplayName
↪  "Sherlock Holmes"
Set-ADUser sherlock.holmes -StreetAddress "221b Baker
↪  St" -City "London" -Country "UK"
Enable-ADAccount sherlock.holmes
```

Notes for the previous slide

1. After the account is created, its attributes can be modified with `Set-ADUser`
2. `Enable-ADAccount` enables the account so it can be used to log on to systems

Querying Data

Example

```
$data = Get-ADUser sherlock.holmes -Property
↪  DisplayName, StreetAddress, City, Country

"User {0} is located at {1}, {2} in {3}" -f
↪  $data.DisplayName, $data.StreetAddress,
↪  $data.City, $data.Country
# User Sherlock Holmes is located at 221b Baker St,
↪  London in UK

"$data" # displays the distinguished name
```

Notes for the previous slide

1. After the account is created, its attributes can be modified with `Set-ADUser`
2. `Enable-ADAccount` enables the account so it can be used to log on to systems

Querying Data

Example

```
Get-ADUser -Filter "name -like '*a*'" # mary.poppins
↪  and Administrator, but not Sherlock
Get-ADUser -Filter "country -eq 'UK'"
Get-ADUser -Filter "country -eq 'UK'" |
↪  Disable-ADAccount
Get-ADUser -Filter "country -eq 'UK'" | Set-ADUser
↪  -StreetAddress "Westminister"
```

Notes for the previous slide

1. Running `Get-ADUser` again would show that the accounts are now disabled

Reset Passwords

Example

```
$user = "sherlock.holmes"
$password = "TemporaryPassword123!"
$secureString = ConvertTo-SecureString -AsPlainText
↪  "$password" -Force
Set-ADAccountPassword -Identity $user -Reset
↪  -NewPassword $secureString
```

Notes for the previous slide

1. The use of variable is optional. It could also be a oneliner
2. Secure strings are objects that contain passwords. They cannot be converted back to plain text
3. It might also be a useful idea to use `Read-Host` to get the username and new password

Unlock User Account

Example

```
Unlock-ADAccount -Identity sherlock.holmes
Unlock-ADAccount mary.poppins
```

Notes for the previous slide

1. If the user tries a wrong password too many times, the account gets locked
2. `-Identity` is a positional parameter

Table of Contents

Creating Groups

Example

```
New-ADGroup -Name "London Accounting" -GroupScope
↪  Global
Set-ADGroup -Identity "London Accounting" -DisplayName
↪  "London Accounting"
Get-ADGroup "london accounting"
```

Remark

Names are case *insensitive*

Notes for the previous slide

1. `GroupScope` matters in a multi domain environment. In a single domain environment, `Global` is always fine

Creating Groups in a Loop

Example

```
$locations = "London", "New York", "Tallinn"
$departments = "Accounting", "Sales", "Warehouse"

foreach ($l in $locations) {
  foreach ($d in $departments) {
    New-ADGroup -Name "$l $d" -GroupScope Global
  }
}
```


Notes for the previous slide

1. The script quickly creates 9 groups
2. Already existing groups will give errors

Adding Members

Example

```
Add-ADGroupMember -Identity "london accounting"
↪  -Members "administrator"
Add-ADGroupMember -Identity "london accounting"
↪  -Members "sherlock.holmes", "mary.poppins"

Get-ADGroupMember "london accounting"
```


Notes for the previous slide

1. `-Identity` and `-Members` are positional parameters

Remove Members

Example

```
Remove-ADGroupMember "london accounting"
↪  "administrator"

Add-ADGroupMember -Identity "london accounting"
↪  -Members "administrator"
Remove-ADGroupMember "london accounting"
↪  "administrator" -Confirm:$false
```


Notes for the previous slide

1. `-Confirm:$false` suppresses the confirmation dialogue. This applies to many commands

Principal Membership

Example

```
Get-ADPrincipalGroupMembership -Identity
↪  sherlock.holmes
Get-ADPrincipalGroupMembership sherlock.holmes
Get-ADPrincipalGroupMembership | select
↪  -ExpandProperty name
```


Notes for the previous slide

1. All users are members of Domain Users
2. `select -ExpandProperty` outputs an array with the names

Nested Groups and -Recurse

Example

```
New-ADGroup Accounting -GroupScope Global
Add-ADGroupMember Accounting (Get-ADGroup -Filter
↪  "name -like '* Accounting'")
New-ADUser teele.raja
Add-ADGroupMember "tallinn accounting" teele.raja

Get-ADGroupMember accounting # displays nested groups
Get-ADGroupMember accounting -Recursive # displays
↪  nested members
```


Notes for the previous slide

1. First, we create a new group Accounting
2. Then, we add all regional accounting groups into it
3. Remember positional parameters `-Identity` and `-Members`

Table of Contents

Computer Accounts

Example

```
Get-ADComputer -Filter *

Get-ADComputer -Filter * -Property *
```

Notes for the previous slide

1. All servers and computers create their own object
2. Listing all properties reveals details, e.g. even when the computer last logged on
3. Computer accounts are created usually on the client side, i.e. when the devices are added to the domain

Contacts

Example

```
New-ADObject -Type Contact john.h.watson -DisplayName
↪  "John H. Watson"
Get-ADObject -Filter "name -eq 'john.h.watson'"
↪  -Properties *
Remove-ADObject (Get-ADObject -Filter "name -eq
↪  'john.h.watson'")
```

Notes for the previous slide

1. Contacts are most useful for adding extranal partnets by creating objects with e-mail addresses, but that needs Exchange installed in the environment
2. Unlike user accounts, these cannot be used for logging in
3. Contacts cannot be removed by just name

Organisational Units

An organizational unit (OU) is a container within a Microsoft Active Directory domain which can hold users, groups and computers

Example

```
Get-ADOrganizationalUnit -Filter *
New-ADOrganizationalUnit -Name "Europe"
Get-ADOrganizationalUnit
↪  "OU=Europe,DC=pscourse,DC=local"
```

Note

`Get-ADOrganizationalUnit "OU=Users,DC=pscourse,DC=local"` will not work

Notes for the previous slide

1. OUs should not be confused with groups

Organisational Units

Example

```
New-ADAccount -Path "OU=Europe,DC=pscourse,DC=local"
↪  oliver.twist
New-ADObject -Path "OU=Europe,DC=pscourse,DC=local"
↪  -Type Contact hercule.poirot
```

Notes for the previous slide

1. Use `-Path` to set the OU when creating objects

Table of Contents

What Did We Learn?

- What is Active Directory?
- Installing first domain controller in a domain
- Creating, managing and removing
  - users
  - groups
- Computer accounts
- Contacts
- Organisational units

2.7 Hyper-V

Hyper-V
PowerShell Course

TalTech

Table of Contents

Introduction Creating VMs VM States Virtual Hardware Switches Checkpoints PS Direct Summary

Table of Contents

1 Introduction
2 Creating VMs
3 VM States
4 Virtual Hardware
5 Switches
6 Checkpoints
7 PS Direct
8 Summary

TalTech
Hyper-V

Virtualisation

- Virtualisation refers to the act of creating a virtual (rather than actual) version of something
- Hardware virtualisation hides the physical characteristics of a computing platform from the users. The software that controls virtualisation is called a "hypervisor"

Notes for the previous slide

1. It's possible to virtualise, for example, applications, computer hardware platforms, storage devices, and network resources

Hyper-V

Microsoft Hyper-V is a hypervisor. It can create virtual machines (VMs) on x86-64 systems running Windows

Example

```
Get-WindowsFeature | ? { $_.name -match "hyper-v" }
Install-WindowsFeature Hyper-V, Hyper-V-PowerShell

Get-Command -Module hyper-v
Get-VM
```


Notes for the previous slide

1. Hyper-V comes free with Windows
2. Nesting VMs might be a bit tricky and is only supported on Intel processors
3. `Get-Command` will show all Hyper-V related PowerShell cmdlets
4. `Get-VM` will show nothing, if there are no VMs

Table of Contents

New-VM

Example

```
New-VM
Get-VM
Remove-VM "New Virtual Machine"

New-VM -Name "TestVm1"
vm TestVm1 | select *
```

Notes for the previous slide

1. Creating a VM with default settings is as easy as running `New-VM`
2. Let's create a test VM `TestVm1`
3. A blank VM has no operating system
4. Default VM is without a virtual hard disk either. You could be explicit with `-NoVHD`
5. Remember that `Get-` is optional

New-VM

Example

```
(Get-VM TestVm1).MemoryStartup/1M # 1024
New-VM "TestVm2" -MemoryStartupBytes 512MB -NewVHDPath
↪  "C:\VHD\testvm2.vhdx" -NewVHDSizeBytes 5GB
(Get-VM TestVm2).MemoryStartup/1MB
(ls "C:\VHD\testvm2.vhdx").Length/1MB # 4
Get-VM TestVm2 | Select-Object VMId | Get-VHD
```

Notes for the previous slide

1. Default RAM is 1 GB
2. VHD means virtual hard disk
3. You can create a VHD on the fly with `-NewVHDPath`
4. `-NewVHDSizeBytes` takes byte count as input but `5GB` will expand to 5368709120
5. The VHDX file will be dynamically expanding and is only 4 MB initially
6. `VMId` is its unique identifier
7. `Get-VM TestVm1 | Select-Object VMId | Get-VHD` gives no output

New-VM

Example

```
New-VHD -Path "C:\vhd\testvm3.vhdx" -SizeBytes 3GB
New-VM "TestVm3" -VHDPath "C:\VHD\testvm3.vhdx"
```

Remark

.vhdx files can be be mounted and browsed with `Mount-DiskImage`

Notes for the previous slide

1. You can also create a VHD separately or use an existing VHD, such as from a previous VM

Table of Contents

Start-VM, Suspend-VM and Stop-VM

Example

```
Start-VM -Name TestVm1
Get-VM | where { $_.State -eq 'Off' } | Start-VM
Get-VM | where { $_.State -eq 'Running' } | Stop-VM
vm TestVm1 | Suspend-VM
(vm TestVm1).State # Paused
vm TestVm2 | Stop-VM -Save # frees memory
(vm TestVm1).State # Saved
```


Notes for the previous slide

1. Both piping from `Get-VM` and running cmdlets directly are options
2. Suspending saves the state but does not free RAM
3. Stopping frees RAM

Table of Contents

DVD Drive

Example

```
Set-VMDvdDrive -VMName TestVm1 -Path
↪  "C:\ISOs\ubuntu-18.10-desktop-amd64.iso"
Set-VMDvdDrive -VMName TestVm1 -ControllerNumber 0
↪  -ControllerLocation 1 -Path
↪  "C:\ISOs\ubuntu-18.10-desktop-amd64.iso"

Get-VMDvdDrive "TestVm1"

Set-VMDvdDrive -VMName TestVm1 -Path $null
```

Notes for the previous slide

1. This is usually vital for installing a new OS
2. If you need mutiple ISOs attached at the same time, be mindful of `ControllerLocation` and `ControllerNumber`
3. `$null` without `ControllerLocation` and `ControllerNumber` will eject all

CPU

Example

```
Get-VM TestVm1 | Set-VM -ProcessorCount 2

Get-VMProcessor TestVm1

Set-VMProcessor TestVm1 -Count 4 -Reserve 10 -Maximum
↪  75

Get-VMProcessor TestVm1 | select *
```

Notes for the previous slide

1. There does not need to be a direct correlation between virtual processors and logical processors
2. Finding the best count is challenging and takes testing
3. `-Reserve` reserves 10% of the logical processor resources and `-Maximum` limits the usage to 75%
4. `Set-VMProcessor` provides more parameters for granual configuration

RAM

Example

```
Get-VMMemory -VMName TestVm1

Set-VMMemory TestVm1 -DynamicMemoryEnabled $true
↪  -MinimumBytes 64MB -StartupBytes 256MB
↪  -MaximumBytes 2GB

Get-VMMemory -VMName TestVm1 | select *
```

Notes for the previous slide

1. Dynamic memory means using as much of the host RAM as is needed
2. Minimum amount will still reseve host's memory
3. Maximum amount will not be exceeded

VHDs

Example

```
New-VHD -Path "C:\vhd\testvm1_2.vhdx" -SizeBytes 3GB

Add-VMHardDiskDrive -VMName TestVm1 -Path
↪  "C:\vhd\testvm1_2.vhdx"

Get-VMHardDiskDrive -VMName TestVm1 # note the number
↪  and location

Remove-VMHardDiskDrive -VMName TestVm1 -ControllerType
↪  SCSI -ControllerNumber 0 -ControllerLocation 5
```

Notes for the previous slide

1. A VM may have several controllers which have several locations

Table of Contents

Switch Types

- External
- Internal
- Private

Notes for the previous slide

1. External allows connection with external network, including the Internet
2. Internal allows connection with the host PC and other VMs on the same host
3. Private switch allows connection to other VMs on the same host only

New-VMSwitch

Example

```
Get-NetAdapter # shows existing (physical) adapters
New-VMSwitch -Name "External Switch 1" -NetAdapterName
↪  Ethernet
New-VMSwitch -Name "Internal Switch 1" -SwitchType
↪  Internal
New-VMSwitch -Name "Private Switch 1" -SwitchType
↪  Private

Get-VMSwitch # lists 3 virtual switches
```


Notes for the previous slide

1. Adapters can be
2. The created virtual adapter will be external, if `-NetAdapterName` or `-NetAdapterInterfaceDescription` is specified

Connecting and Disconnecting Adapters

Example

```
Connect-VMNetworkAdapter -VMName "TestVm1" -SwitchName
↪  "Private Switch 1"
Add-VMNetworkAdapter -VMName TestVm1 -Name "Another
↪  Adapter"
Connect-VMNetworkAdapter -VMName "TestVm1" -SwitchName
↪  "External Switch 1" -Name "Another Adapter"

Get-VMNetworkAdapter TestVm1

Disconnect-VMNetworkAdapter -VMName "TestVm1"
```


Notes for the previous slide

1. VMs have adapters, which can be connected to switches
2. Each VM by default has a network adapter `Network Adapter`
3. `Conntect-VMNetworkAdapter` without the `-Name` parameter connects all adapters to the named switch
4. `Disconnect-VMNetworkAdapter` does not take the switch name as input
5. It disconnects all switches, if you don't specify `-Name`

Table of Contents

Creating Checkpoints

Example

```
Checkpoint-VM -Name TestVm1 -SnapshotName Checkpoint1

Get-VMCheckpoint -VMName TestVm1
```

Notes for the previous slide

1. Checkpoints are also called snapshots by other vendors
2. checkpoints help administrators roll back changes in the case of problems by taking point-in-time images of VMs

Returning to a Checkpoint

Example

```
Get-VMCheckpoint -VMName TestVm1

Restore-VMCheckpoint -Name Checkpoint1 -VMName TestVm1
↪  -Confirm:$false

Remove-VMCheckpoint -VMName TestVm1 -Name Checkpoint1

Get-VMCheckpoint * | Remove-VMCheckpoint
```

Notes for the previous slide

1. By default, it will restore the memory state to what it was when the checkpoint was taken
2. Taking a checkpoint after a previous checkpoint will create a chain
3. Checkpoints make the VMs slower over time and should not be used long term

Table of Contents

PowerShell Direct Sessions

Example

```
Invoke-Command -VMName TestVm1 -ScriptBlock { hostname
↪ }
Enter-PSSession -VMName TestVm1 -Credential
↪ "pscourse\administrator"
Exit-PSSession
```

Remark

If the VM is on the network and the name matches its hostname, both `-ComputerName` and `-VMName` will work, but through a different channel

Notes for the previous slide

1. It is also possible to connect via a network connection

Copying Files

Example

```
$s = New-PSSession -VMName TestVm1

Copy-Item -ToSession $s -Path "C:\host\data.txt"
 ↪  -Destination "C:\guest\"
Copy-Item -FromSession $s -Path "C:\guest\data.txt"
 ↪  -Destination "C:\host\"

Remove-PSSession $s
```


Notes for the previous slide

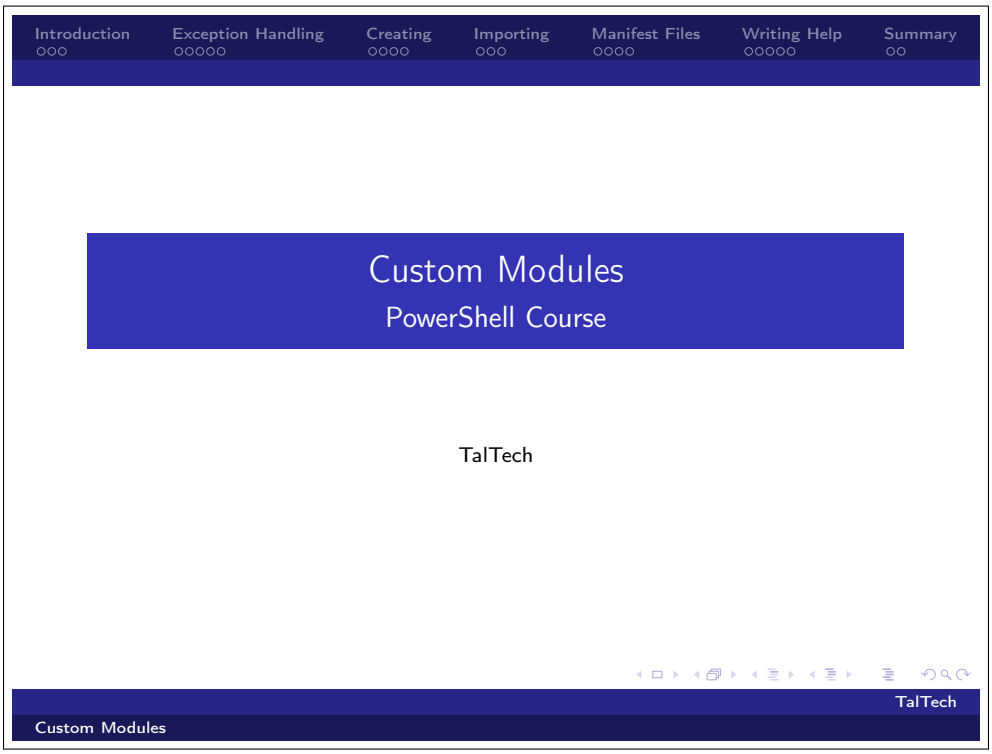
1. Again, this would also work without a network, but it only works with Windows VMs
2. Session should be removed once done

Table of Contents

What Did We Learn?

- What is Hyper-V?
- Creating VMs
- Changing states
- Managing virtual hardware
- Managing virtual adapters and switches
- Using checkpoints
- PowerShell Direct sessions

2.8 Custom Modules



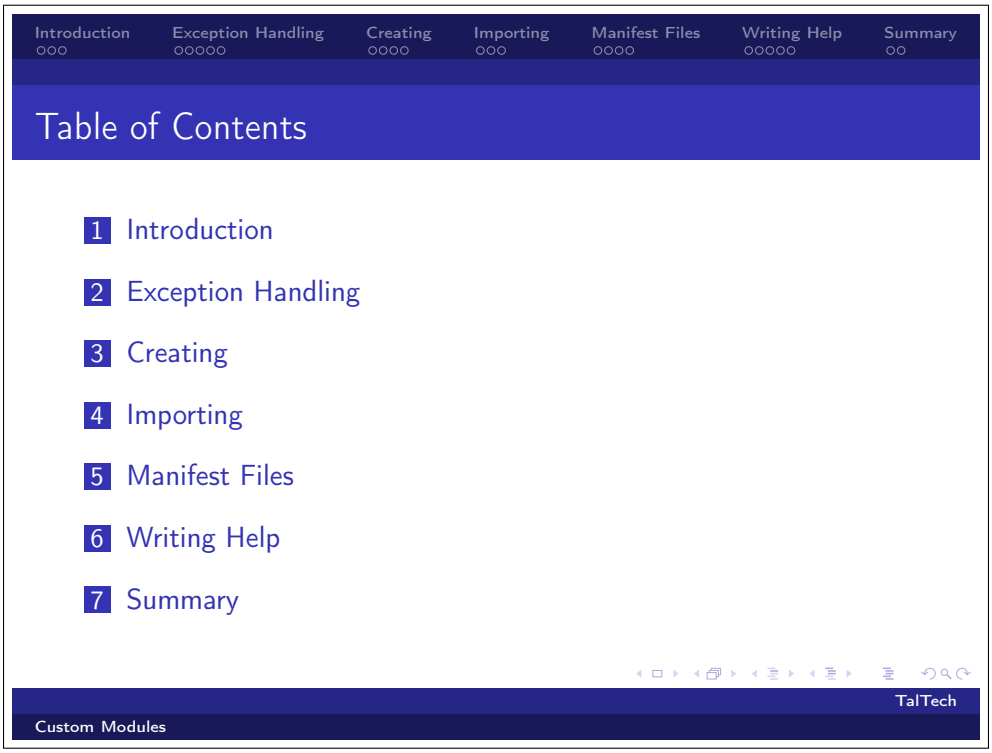


Table of Contents

What Are Custom Modules and Why Create Them?

- Modules are PowerShell scripts that define functions or variables and are saved with the `.psm1` extension
- They are useful for enabling the same set of variables and custom commands quickly across different computers

Notes for the previous slide

1. Creating your first PowerShell module is as easy as creating an empty `.psm1` file
2. Using the `.psm1` extension means that it's possible to use the `Module` cmdlets such as `Import-Module` on it

Placing Modules

- The created `.psm1` files must be placed into a path that is in the `$env:PSModulePath` variable
- By default, the variable includes (run `$env:PSModulePath -split ";"` to see)

```
"C:\Users\%username%\Documents\WindowsPowerShell\
↪  Modules"
"C:\Program Files\WindowsPowerShell\Modules"
"C:\Windows\system32\WindowsPowerShell\v1.0\
↪  Modules"
```

Notes for the previous slide

1. The files could be placed to these locations via various methods, such as PowerShell, GPO or being prepopulated in the OS image

Table of Contents

Why Do We Need Exception Handling?

In more complex scripts, errors are to be expected. Exception handling allows us to gracefully continue running the script and log meaningful error message

Notes for the previous slide

1. Before we talk about custom modules, let's learn the concept of exception handling
2. Exceptions are errors

Try/Catch

Example

```
try { Get-Content Non-existentFile.txt }
catch { "There was an error" }
# the exception is not caught

try {
  Get-Content -ErrorAction Stop Non-existentFile.txt
}
catch { "There was an error" }
# reports the error gracefully
```


Notes for the previous slide

1. It's important to distinguish between non-terminating and terminating errors
2. By default, syntax errors and running out of memory are terminating
3. `-ErrorAction Stop` would make running the cmdlet terminating
4. Then `catch` will catch it

Try/Catch

Example

```
$ErrorActionPreference = "Stop"

try {
  Get-Content Non-existentFile.txt
}
catch {
  "There was an error `"{0}`" It occurred with item
  ↪  `"{1}`"" -f $_.Exception.Message,
  ↪  $_.Exception.ItemName
}
```


Notes for the previous slide

1. For convenience, you could set the variable `ErrorActionPreference` and it would apply to all cmdlets
2. `$_` holds detailed info about the error in the `catch` block

Finally

Example

```
$ErrorActionPreference = "Stop"
try {
  Get-Content Non-existentFile.txt
}
catch { "Error" }
finally {
  "An attempt was made at {0}" -f (Get-Date) |
  ↪  Out-File Log.txt
}
```


Notes for the previous slide

1. The `finally` block is run in any case
2. You could check in `finally` if the attempt was successful

Table of Contents

Creating a Module

Create a `MathModule.psm1` file with the content of

Example

```
function Get-Factorial {
    $counter = $args[0]
    $factorial = 1
    while ($counter) {
      $factorial *= $counter--
    }
    $factorial
}
```

(continues on the next slide)

Notes for the previous slide

1. This is an example we used in an earlier seminar

(continues from the previous slide)

Example

```
function Get-Fibonacci ($n) {
  $x, $y, $z = 0, 1, 1
  while ($z -le $n) {
    $z
    $z = $x + $y
    $x = $y
    $y = $z
  }
  }
```

(continues on the next slide)

Notes for the previous slide

1. This function returns Fibonacci numbers up to the number given as parameter

(continues from the previous slide)

Example

```
Export-ModuleMember -Function "*-Factorial"
```

Place the file to `"%userprofile%\Documents\WindowsPowerShell\Modules\MathModule"`

Notes for the previous slide

1. Create missing paths as needed

Table of Contents

Import-Module

Example

```
Get-Module -ListAvailable MathModule
Import-Module -Name MathModule
```


Notes for the previous slide

1. Without `-ListAvailable` you only see already loaded modules
2. Note how the `ModuleType` is `Script`
3. You could also not move the file to a path in `$env:PSModulePath` and specify the full path to the `.psm1` file as the `-Name`
4. `ipmo` is an alias to `Import-Module`

Example

```
Get-Factorial 6 # 720
Get-Fibonacci 150 # command not found
```


Notes for the previous slide

1. We only exported `*-Factorial`
2. Remove the one exporting line or add an additional line about `Get-Fibonacci` to the file in order to have the other function too
3. It's best to be explicit about the functions you make available, because sometimes you might want to have functions that you use in the module but should not be made available to module users
4. Modules are an easy way to get commonly used custom functions to other computers

Table of Contents

Why Create One?

A manifest file can contain various data about the module. It is a `.psd1` file that contains a hash table with keys and values. There are about 30 valid manifest elements. See info about all by running `help New-ModuleManifest -Full`. Some examples are

- Author
- Company
- Version
- Description
- Help file location
- Functions, variables, aliases to export

Notes for the previous slide

1. Manifests are important for bigger modules

New-ModuleManifest

Example

```
New-ModuleManifest -Author "Instructor"`
-Path "$env:USERPROFILE\Documents\WindowsPowerShell\
↪  Modules\MathModule\MathModule.psd1"`
-CompanyName "TalTech"`
-ModuleVersion "1.0.0"`
-Description "Some math tools"
```

Notes for the previous slide

1. It is possible to create and modify the file manually, but it is more convenient with the `New-ModuleManifest` cmdlet
2. Note how a backtick allows us to break lines. Splatting would be an alternative

Accessing the Attributes

Example

```
Get-Module -ListAvailable MathModule

Get-Module -ListAvailable MathModule | select *
```

Notes for the previous slide

1. Open a new PowerShell session to clear previous data
2. Note how the `ModuleType` is now `Manifest`

Table of Contents

Example

```
Get-Help Get-Factorial # a very blank help
```


Notes for the previous slide

1. By default, the help page is very minimalistic
2. PowerShell cannot assume what the function is intended for

Example

```
function Get-Fibonacci {
  <#
    .SYNOPSIS
      Returns numbers in the Fibonacci sequence
    .EXAMPLE
      Get-Fibonacci 400
    .DESCRIPTION
      The function takes an integer as input and
↪   returns an array with Fibonacci numbers as output.
↪   The last number in the output, will be less or
↪   equal to the input value.
  #>
  Param([parameter(Mandatory=$true)][int]$Max)
```


Notes for the previous slide

1. The code continues on the next slide
2. Script help can be created with a special multiline comment within the function definition
3. Synopsis should be very short
4. There are many more attributes that can be definied

Example

```
  $x, $y, $z = 0, 1, 1
  while ($z -le $Max) {
    $z; $z = $x + $y
    $x = $y; $y = $z
  }
}
```


Notes for the previous slide

1. This ends the function definition with help

Example

```
help Get-Fibonacci -Full
```


Notes for the previous slide

1. Now this command displays a help page with what we defined before
2. This becomes very powerful when you one day find yourself developing functions for others

Table of Contents

What Did We Learn?

- What modules are
- Exception handling
- Creating modules
- Using modules
- Manifest files
- Creating custom function help pages

Appendix 3 Lab Assignments

3.1 Introduction

1. Name 5 facts that are unique or almost unique to PowerShell
2. Name 5 use cases where PowerShell would be the right tool to use
3. Name 3 differences between PowerShell versions
4. Which PowerShell version is pre-installed with the client and server Windows operating system that is currently most popular?
5. Open the Integrated Scripting Environment, create a new script that prints the PowerShell version and the string ”Hello World!”
6. Save the script as a file and run the file from the console

Post answers to Moodle. The script and its output should be captured as screenshots and attached to the submission. Code should be commented, i.e. comments explain what the individual commands on each line do

3.2 Basics 1

1. Explain how variables are defined and which variable names are illegal. Bring examples. Show happens when an illegal name syntax is used in a script
2. Define and print 3 variables of each class that the seminar covered. The hashtables and PSCustomObjects must contain values of different different classes. Try to create the examples base on real life objects or events
3. Demonstrate the use of each learned arithmetic operators with 3 examples
4. Do the same for comparison operators

5. Do the same for assignment operators

Post answers to Moodle. The script and its output should be captured as screenshots and attached to the submission. Code should be commented, i.e. comments explain what the individual commands on each line do

3.3 Basics 2

1. Create and demonstrate a function that does basic arithmetic calculations. The input parameters are two floats and the arithmetic operator as a string. The output should be the calculation with the answer, such as `3 * 4 = 12`

2. Create and demonstrate a function that takes a date object as input. The output should wish a happy birthday with the age, e.g. `Happy 21st Birthday!` in case the birthday is the current day and print the remaining number of days until the birthday otherwise

3. Create and demonstrate a function that generates Fibonacci numbers to the count specified as an input parameter loaded with `Read-Host`

4. Make sure at least one of the functions created above uses named parameters and demonstrate splatting with it

Post answers to Moodle. The script and its output should be captured as screenshots and attached to the submission. Code should be commented, i.e. comments explain what the individual commands on each line do

3.4 Basics 3

1. Demonstrate the use of filtering and formatting on the output of `Get-ChildItem` with 7 different commands

2. Explain the differences between `Write-Output` and `Write-Host`

3. Create a function which downloads the JSON object from the URL `https://jsonplaceholder.typicode.com/todos` and swaps all completed items to not completed and vice-versa

4. Demonstrate the use of the `-f` operator for strings including padding, number formatting and time formatting in 10 different examples

5. Create and demonstrate a function that takes any string as input and then prints its characters in reverse order using a loop

Post answers to Moodle. The script and its output should be captured as screenshots and attached to the submission. Code should be commented, i.e. comments explain what the individual commands on each line do

3.5 Windows OS Management

1. Create a function that lists the .doc or .docx files that are bigger than 300 kB in the input folder and its subfolders. The output should be a JSON object that also contains the files' full path in capital letters

2. Create a command that outputs current running processes or services into a CSV file with at least 6 columns. Choose the columns yourself but justify the decisions

3. Create a script that extracts basic hardware info (e.g. CPU, RAM, HDD, motherboard, multimedia devices) with CIM and is able to output the data in both CSV and JSON formats

4. Create a function that adds a new log to Windows Event Log Management. Create a function which writes events to the log compiled from at least two input parameters of the function

Post answers to Moodle. The script and its output should be captured as screenshots and attached to the submission. Code should be commented, i.e. comments explain what the individual commands on each line do

3.6 Active Directory

1. Create a function that takes a username and new password as input and resets user's password

2. Create contacts from a JSON file

3. Create users from a CSV file

4. Create a new group `Regional Admins` and place the created users into it

5. Place the new group into `Domain Admins`

6. Create users in a loop from a CSV file that also includes group memberships. Create groups, if they do not already exist

Post answers to Moodle. The script and its output should be captured as screenshots and attached to the submission. Code should be commented, i.e. comments explain what the individual commands on each line do

3.7 Hyper-V

1. Create a virtual machine with 2 CPUs, 2 GB of RAM and two virtual hard disks (3 GB and 1GB).
2. Download any Windows operating system trial version ISO, mount it to the virtual DVD drive and install the operating system
3. Demonstrate the use of checkpoints
4. Demonstrate the use of PowerShell Direct (both `Invoke-Command` and `Copy-Item` with either `-ToSession` or `-FromSession`)

Post answers to Moodle. The script and its output should be captured as screenshots and attached to the submission. Code should be commented, i.e. comments explain what the individual commands on each line do

3.8 Custom Modules

1. Create a custom module about Active Directory or Hyper-V commands where you link together or hard code some tasks and variables. It must include:
    - At least 4 functions out of which not all are exported
    - Variables
    - A demonstration of meaningful exception handling
    - A manifest file
    - Help section

Post answers to Moodle. The script and its output should be captured as screenshots and attached to the submission. Code should be commented, i.e. comments explain what the individual commands on each line do

Appendix 4 Knowledge Tests

4.1 Introduction

1. Which of the following are Microsoft recommended editors for writing PowerShell?
   (a) Notepad
   (b) Notepad++
   (c) Microsoft Visual Studio Code
   (d) Integrated Scripting Environment (ISE)
2. PowerShell is pre-installed on
   (a) Only Windows 7 and beyond
   (b) Only Windows Vista and beyond
   (c) Only Windows 2000 and beyond
   (d) Only Windows Server 2003 and beyond
   (e) Only Windows Server 2008R2 and beyond
   (f) Only Windows Server 2016 and beyond
3. PowerShell is open source and can be used on Linux since
   (a) Version 1.0
   (b) Version 2.0
   (c) Version 3.0
   (d) Version 4.0
   (e) Version 5.1
   (f) Version 6.0

4. What is the syntax for PowerShell native cmdlets? Drag and drop the following in the correct order:

   (a) Dash (−)

   (b) Verb

   (c) Adjective

   (d) Plus (+)

   (e) Adverb

   (f) Noun

   (g) Slash (/)

   (h) Double quotation marks (")

5. How are decimals and array elements separated in PowerShell input?

   (a) `.` separates decimals and `,` separates array elements

   (b) `,` separates decimals and `.` separates array elements

   (c) Depends on system language settings

6. Parameters can be

   (a) Nicknamed/so-called

   (b) Named

   (c) Positional

   (d) Reverse

   (e) Unnamed

7. Piping means

   (a) Writing code that follows convention

   (b) Using only PowerShell native cmdlets

   (c) Defining custom functions

   (d) Passing the output of one command to the next

   (e) Modifying code without changing functionality

8. Comments are made by using

(a) Starting with `#` for single line comments

(b) Starting with `//` for single line comments

(c) Starting with `%` for single line comments

(d) Starting with `<#` and ending with `#>` for multi line comments

(e) Starting with `/*` and ending with `*/` for multi line comments

(f) Starting with `*/` and ending with `/*` for multi line comments

9. Variables are identified by the preceding

(a) `$`

(b) `_`

(c) `!`

(d) `?`

(e) `=`

10. PowerShell scripts have file extension

(a) `.power`

(b) `.ps1`

(c) `.pss`

(d) `.ps`

(e) `.script`

4.2 Basics 1

1. Running `$x` would

(a) Set `$x` to value zero / unset the variable

(b) Print the value of `$x`

(c) Do nothing

(d) Return an error

2. Which are *not* correctly formed variables

(a) `$important_variable`

(b) `$important variable`

(c) `importantVariable`

(d) `${important variable}`

3. Setting `$var` to value `"TalTech"` and running `'$var'` will

(a) Return an error

(b) Print `TalTech`

(c) Print `$var`

4. Which of the following will have `[bool]` value `True`

(a) `[bool]"False"`

(b) `[bool]-100`

(c) `[bool]"0"`

(d) `[bool]3`

5. Which are valid arrays

(a) `@(1,2,3)`

(b) `40..10`

(c) `@()`

(d) `1, 4, "TalTech"`

(e) `1.2.4`

6. Hashtables store

(a) Lists of numbers only

(b) Lists of any kind of elements

(c) CSV files

(d) Key/value pairs

(e) Hash function outputs

7. Attributes in PSCustomObjects may be

(a) Integers

(b) Other PSCustomObjects

(c) Hashtables

(d) Strings

(e) Time spans

8. What is the output of `5%2`?

   (a) 1

   (b) 2

   (c) 3

   (d) 4

   (e) 5

9. Which of the following will return `True`?

   (a) `'a' -eq "a"`

   (b) `6 -lt 6`

   (c) `5 -lt 8`

   (d) `4 -ge 5`

   (e) `3 -le 3`

10. Running `$array="a","b","c"` and `$array-="c"` results in

    (a) `$array` contains two elements — `"a"` and `"b"`

    (b) `$array` is empty

    (c) An error is displayed

4.3 Basics 2

1. Which are conditional statements in PowerShell?

   (a) If

   (b) Switch

   (c) When

   (d) Then

(e) Else

(f) Ifelse

(g) Elseif

2. Loop `for ($i=2; $i -lt 1; $i++) { "" }` will

   (a) Not run at all

   (b) Return an error

   (c) Run once

   (d) Run twice

   (e) Run indefinitely

3. Loop `ForEach-Object ($x in @(1..2)) { "" }` will

   (a) Not run at all

   (b) Return an error

   (c) Run once

   (d) Run twice

   (e) Run indefinitely

4. The result of running `Write-Host "test" | Out-File "test"` is

   (a) An error

   (b) An string `"test"` in file `test`

   (c) An empty file `test`

5. The result of running `Get-Date | tee -FilePath time -Variable time` is

   (a) Current date is printed in the console

   (b) Current date is saved to file `time`

   (c) Current date is saved to variable `time`

   (d) An error is returned

6. Which cmdlets or operators can write output to a file

   (a) `Out-File`

(b) `Write-Output`

(c) `Write-Host`

(d) `clip`

(e) `Tee-Object`

(f) > and >>

7. Which are correct function definitions

   (a) `function myFunction {}`

   (b) `def function myFunction {}`

   (c) `def myFunction() {}`

   (d) `myFunction {}`

   (e) `new myFunction {}`

8. Which may be correct calls of a function

   (a) `myFunction -Parameter1 value1`

   (b) `myFunction value1`

   (c) `myFunction -Parameter1 value1 -Parameter2 value2`

   (d) `myFunction -Parameter1 value1 value2`

   (e) `myFunction value1 value2`

9. You run `$input = Read-Host "Enter a value"`. The type of `$input` is

   (a) Always a string

   (b) Dependant on what the given input is

   (c) Always boolean

   (d) Always an integer

10. Which is the correct way to do parameter splatting?

    (a) `Write-Host $parameters`

    (b) `Write-Host !parameters`

    (c) `Write-Host @parameters`

    (d) `Write-Host ?parameters`

    (e) `Write-Host _parameters`

4.4 Basics 3

1. Which are correct syntaxes to extract attributes?

   (a) `[TimeSpan]"1:2:3" | Select-Object Minutes`

   (b) `[TimeSpan]"1:2:3" | Minutes`

   (c) `[TimeSpan]"1:2:3" | show Minutes`

   (d) `[TimeSpan]"1:2:3" | select minutes`

   (e) `([TimeSpan]"1:2:3").Minutes`

2. Which command will show stopped services?

   (a) `Get-Service | Where-Object { $Status -eq "Stopped" }`

   (b) `Get-Service | Where-Object { $!.Status = "Stopped" }`

   (c) `Get-Service | Where-Object { $this.Status=="Stopped" }`

   (d) `Get-Service | Where-Object { $_.Status -eq "Stopped" }`

3. How many lines will running `(Compare-Object 1,2 2,3).Count` output?

   (a) 0

   (b) 1

   (c) 2

   (d) 3

   (e) 4

4. Which of the following return `True`?

   (a) `"TalTech" -match "t"`

   (b) `"TalTech" -match "b"`

   (c) `"TalTech" -match "l.e"`

   (d) `"TalTech" -match "l_e"`

5. Which commands create loops in PowerShell?

   (a) `%`

   (b) `Until`

(c) `foreach`

(d) `ForEach-Object`

(e) `circulate`

(f) `while`

6. Which are correct for accessing attributes in a hashtable?

   (a) `$hashtable.attribute`

   (b) `$hashtable["attribute"]`

   (c) `$hashtable/attribute`

   (d) `$hashtable!attribute`

   (e) `$hashtable | select attribute`

7. Which are correct for accessing attributes in a PSCustomObject?

   (a) `$object.attribute`

   (b) `$object[attribute]`

   (c) `$object/attribute`

   (d) `$object!attribute`

   (e) `$object | select attribute`

8. Which cmdlet will let you read a CSV file into a PowerShell object?

   (a) `Get-Csv`

   (b) `Get-Content` together with `ConvertFrom-Csv`

   (c) `Import-Csv`

   (d) `Read-File` together with `Parse-Csv`

9. JSON is

   (a) Supported by PowerShell

   (b) Human-readable

   (c) Machine-readable

   (d) Less verbose than XML

10. Which are useful parameters for the `Get-Help` cmdlet?

(a) `-Examples`

(b) `-Full`

(c) `-AutoUse`

(d) `-Online`

4.5 Windows OS Management

1. After running `"some text" | Out-File file.txt`, the encoding of the file will be

   (a) ASCII

   (b) UTF-8

   (c) UTF-8 BOM

   (d) UTF-16

2. Which are valid cmdlets to manage services?

   (a) `Kill-Service`

   (b) `Run-Service`

   (c) `Reboot-Service`

   (d) `Restart-Service`

   (e) `Set-Service`

3. Which `Stop-Computer` parameter stops the computer even when users are logged in?

   (a) `-Ignore`

   (b) `-Force`

   (c) `-Override`

   (d) `-Super`

   (e) `-Logout`

4. Which cmdlets can show detailed CPU and BIOS info?

   (a) `Get-WmiObject`

   (b) `Get-CimInstance`

(c) `Get-ChildItem`

(d) `Get-Host`

(e) `Get-ComputerInfo`

5. Drag and drop the parts to make a command that shows all installed features

(a) `Get-WindowsFeature`

(b) `$_.InstallState`

(c) `"Installed"`

(d) `-eq`

(e) `}`

(f) `|`

(g) `?`

(h) `{`

6. The relation between `Invoke-Command` and `Enter-PSSession` is that

(a) `Invoke-Command` always needs to precede `Enter-PSSession`

(b) `Invoke-Command` works only locally, `Enter-PSSession` also remotely

(c) `Invoke-Command` works for single commands, `Enter-PSSession` is more convenient for multiple commands

(d) `Enter-PSSession` always needs to precede `Invoke-Command`

7. Which is correct for retrieving application error events?

(a) `Get-EventLog -LogName "Application" -EntryType "Error"`

(b) `Get-EventLog -EntryType "Application" -LogName "Error"`

(c) `Get-EventViewer -LogName Application -EntryType Error`

(d) `Get-EventViewer -EntryType Application -LogName Error`

(e) `Get-Event -LogName "Application" -EntryType "Error"`

(f) `Get-Event -EntryType "Application" -LogName "Error"`

8. Running `Get-ChildItem` with `-Recurse` will

(a) also show parent directories

(b) also show child items

(c) show more detailed info about the items in the directory

(d) monitor the directory for changes

9. Which commands stop the DHCP service

(a) `Halt-Service DHCP`

(b) `ShutDown-Service DHCP`

(c) `Set-Service -Name DHCP -Running $False`

(d) `Stop-Service DHCP`

(e) `Set-Service -Name DHCP -Status Stopped`

10. `Invoke-Command` takes the input commands as which object class?

(a) `[String]`

(b) `[PowerShell]`

(c) `[ToRun]`

(d) `[ScriptBlock]`

(e) `[Commands]`

4.6 Active Directory

1. Which of the following is true about Active Directory?

(a) It's a directory service

(b) It's developed by Microsoft

(c) It helps to authenticate users

(d) It helps to authorise users

(e) There's a PowerShell module for it

2. Drag and drop items to organise Active Directory containers/objects starting from the largest

(a) Domain

(b) Users, groups and contacts

(c) Organisational Unit

(d) Forest

(e) Subdomain

3. Which command properly creates a `SecureString` object?

(a) `ConvertTo-SecureString -String "secret" -Force`

(b) `ConvertTo-SecureString -String "secret" -AsPlainText -Confirm:$false`

(c) `ConvertTo-SecureString -String "secret" -AsPlainText -Secure`

(d) `ConvertTo-SecureString -String "secret" -AsPlainText -Force`

(e) `ConvertTo-SecureString -String "secret" -AsPlainText -Unsecure`

4. How do services identify Active Directory user accounts?

(a) By display name

(b) By username

(c) By e-mail

(d) By SID

5. Which is correct for searching Estonian users in Active Directory?

(a) `Get-ADUser -Filter "country == 'Estonia'"`

(b) `Get-ADUser -Filter "country -eq 'Estonia'"`

(c) `Get-ADUser -Filter "$country = 'Estonia'"`

(d) `Get-ADUser -Filter "$country -eq 'Estonia'"`

(e) `Get-ADUser -Filter "$_.country = 'Estonia'"`

6. Drag and drop to create a command which removes User1 from Group1

(a) `User 1`

(b) `Group 1`

(c) `Remove-ADGroupMember`

(d) `Set-ADGroupMember`

(e) `-Remove`

7. In a single domain environment, it is recommended to use group scope

   (a) Universal

   (b) Domain local

   (c) Global

8. Which of these may be Active Directory objects?

   (a) Licences

   (b) Users

   (c) Computers

   (d) Applications

   (e) Printers

   (f) Contacts

   (g) Groups

9. Running "`$(Get-ADUser user1)`" will display

   (a) Username

   (b) Display name

   (c) SID

   (d) Distinguished name

10. By default, newly created Active Directory user accounts are

    (a) Disabled

    (b) Enabled

    (c) Locked

    (d) Hidden

    (e) Protected

4.7 Hyper-V

1. Running `New-VM` without any parameters will

   (a) Return an error

   (b) Create a VM named `New Virtual Machine`

   (c) Prompt for the name

2. Which statements are true?

   (a) Hypervisor controls virtualisation

   (b) Microsoft Hyper-V is a hypervisor

   (c) It is possible to virtualise applications, hardware, storage and network

3. Which parameters for `New-VM` will create a VHD of 5 gigabytes?

   (a) `-NewVHDSizeBytes 5GB`

   (b) `-NewVHDSize 5GB`

   (c) `-NewVHDSizeBytes 5368709120`

   (d) `-NewVHDSizeBytes 5 GB`

4. Which commands will free host's memory?

   (a) `Stop-VM`

   (b) `Stop-VM -Save`

   (c) `Suspend-VM`

5. Which hardware attributes can Hyper-V manage for a VM?

   (a) CPU

   (b) RAM

   (c) VHDs

   (d) Screen count

   (e) Network adapters

   (f) Power Supply Unit (PSU) capacity

6. Adapters linked to internal switches can

(a) Communicate with the Internet

(b) Communicate with the host

(c) Communicate with other VMs on the host

7. Adapters linked to private switches can

(a) Communicate with the Internet

(b) Communicate with the host

(c) Communicate with other VMs on the host

8. Which is syntax is correct?

(a) `Snapshot-VM -Name Vm1 -CheckpointName beforeUpgrade`

(b) `Checkpoint-VM -Name Vm1 -SnapshotName beforeUpgrade`

(c) `Snapshot-VM -Name Vm1 -SnapshotName beforeUpgrade`

(d) `Checkpoint-VM -Name Vm1 -CheckpointName beforeUpgrade`

9. `Vm1` is a virtual machine that does not have a network connection with the host. Which command or commands would establish a session?

(a) `Enter-PSSession -VMName TestVm1`

(b) `Enter-VMSession -Name TestVm1`

(c) `Enter-PSSession -Name TestVm1`

(d) `Enter-VMSession -VMName TestVm1`

10. By default, new VMs are created

(a) without a VHD

(b) with a VHD of 5 GB

(c) with a VHD of 16 GB

(d) with a VHD of 20 GB

4.8 Custom Modules

1. In order to make cmdlets work properly in `try{}` block, the cmdlets should have

(a) `-Try $True`

(b) `-Catch $True`

(c) `-ErrorAction Stop`

(d) `-Error Stop`

2. Code in the `finally{}` block will be run

(a) never

(b) always

(c) only if there was an error

(d) only if called from the `catch{}` block

3. The first positional argument in a custom function can be accessed with

(a) `$args[0]`

(b) `$arguments[0]`

(c) `$positional[0]`

(d) `$args[1]`

(e) `$arguments[1]`

(f) `$positional[1]`

4. What is the extension for PowerShell module files?

(a) `.psModule`

(b) `.module`

(c) `.module1`

(d) `.psm1`

5. Functions, variables and aliases in the module can be made available with

(a) `Show-ModuleAsset`

(b) `Export-ModuleAsset`

(c) `Export-ModuleMember`

(d) `Show-ModuleMember`

(e) `Reveal-ModuleAsset`

6. Valid `New-ModuleManifest` parameters are

(a) `CompanyName`

(b) `Author`

(c) `ModuleVersion`

(d) `Todo`

(e) `Description`

7. Without defining help section in a custom function, calling its help page will

   (a) result in an error

   (b) return nothing

   (c) show minimal info

   (d) show fully detailed info

8. Which command will initialise custom module files?

   (a) `Load-Module`

   (b) `Use-Module`

   (c) `Import-Module`

   (d) `Start-Module`

9. Which variable contains folder locations from where custom modules are automatically searched for?

   (a) `$env:PSModulePath`

   (b) `$env:Modules`

   (c) `$env:ModulePath`

   (d) `$env:PSModules`

10. Which help section is meant to give a short, one sentence, summary of what the command is for?

    (a) `.OVERVIEW`

    (b) `.SYNOPSIS`

    (c) `.USEFULNESS`

    (d) `.PURPOSE`