Modul 122
Scripting

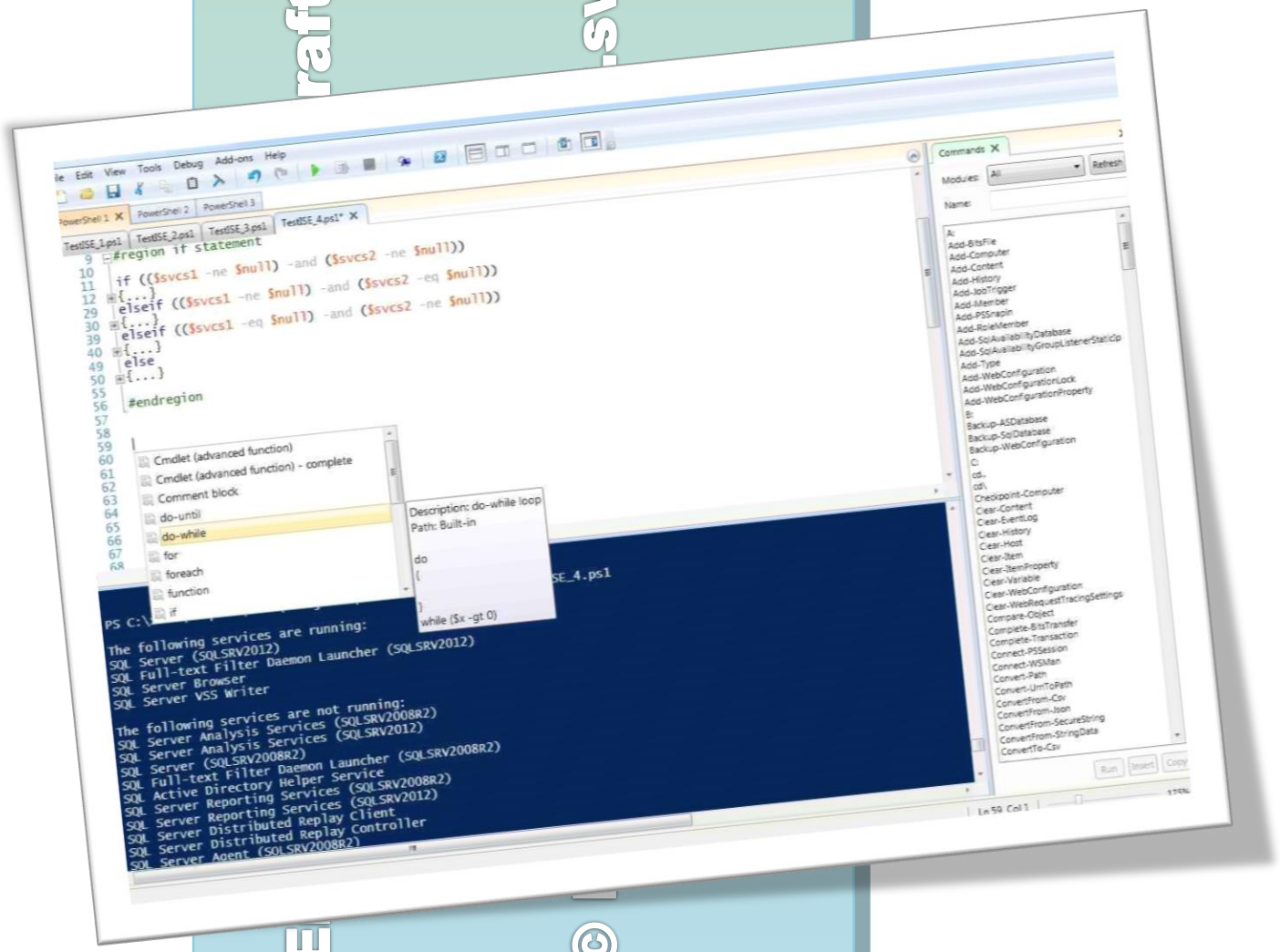
Zusammenfassung: Scripting – Einführung Scripting von R. Burger
Erstellt im November 2015
(Draft Version 0.02, Oktober 2018)

Einleitung

Abgestimmt auf den Unterricht für das Modul 122 finden Sie hier eine Einführung ins Scripting. **Das folgende Script ergänzt den im Unterricht behandelten Stoff und setzt voraus, dass Sie selber Notizen während der Unterrichtszeit geschrieben/erstellt haben.**

Dieses Dokument ist noch in der ersten Entwurfsphase und ist noch nicht zum Verteilen vorgesehen! In den nächsten Wochen/Monaten wird eine überarbeitete Version im Downloadbereich zur Verfügung stehen. Danke für's Verständnis.

Für Fragen und Anregungen stehe ich gerne zur Verfügung.



Inhaltsverzeichnis

Grundlagen

Scripts wie PowerShell sind für einheitliche Automation und Verwaltung des Systems.
Scripts sind Effektiv und Effizient ohne enormen Lernaufwand.
Interaktiv oder automatisch (sheduled Jobs und Workflows)
Je nach Scriptsprache auch Multi-Prozessing und Multitasking fähig.
Verschiedene Scriptsprachen erlauben auch Remoteverwaltung von Systemen (PowerShell, Perl, KiXtart etc.)

Einführung

- Für was sind Scripts gut?
- Wozu Scripts?

Der Unterschied von Scripts und „normalen“ Programmen ist gar nicht so gross, wie man zuerst meinen kann. Beide können im Hintergrund laufen oder auf Interaktion mit dem Benutzer warten. Je nach Programmier- resp. Scriptsprache, können Scripts auch objektorientiert oder einfach strukturiert geschrieben werden. Beide können sehr simpel aufgebaut sein, oder mit einer grafischen Oberfläche erscheinen.
Wo liegt denn jetzt der Unterschied?

Programme müssen kompiliert werden auf die entsprechende Hardware und das Betriebssystem. Scripte laufen auf allen Plattformen, welche unterstützt werden. Es braucht dazu einen Interpreter, welcher auf dem Zielsystem auf die Hardware und Betriebssystem abgestimmt installiert sein muss.

- Möglichkeit zum Automatisieren von Abläufen
- Rechnerunabhängig (PHP, Perl, Java-Script, etc.)
- Etc.…

Hilfesyntax lesen

Syntax zum lesen allgemeiner Scripts
Wie lese ich ein Help im Beispiel von CMD:

```
C:\Users\burgerre>net share /?
Die Syntax dieses Befehls lautet:

NET SHARE
Freigabename
      Freigabename=Laufwerk:Pfad [/GRANT:Benutzer,[READ | CHANGE | FULL]]
                                 [/USERS:Nummer | /UNLIMITED]
                                 [/REMARK:"Text"]
                                 [/CACHE:Manual | Documents | Programs |
BranchCetache | None]
      Freigabename [/USERS:Anzahl | /UNLIMITED]
                   [/REMARK:"Text"]
                   [/CACHE:Manual | Documents | Programs | BranchCache | None]
      {Freigabename | Gerätename | Laufwerk:Pfad} /DELETE
      Freigabename \\Computername /DELETE
```

Net share kann alleine gestartet werden (ohne Parameter)
Alle Optionen mit [ ] sind Optional
Alle Möglichkeiten mit { } sind Pflicht. Beispiel mit dem Parameter / delete ist es zwingend, dass einer der drei Parameter Freigabename | Gerätename | Laufwerk:Pfad gesetzt wird.
Das Pipe | bedeutet ODER.

Was ist eine Shell

Die Shell oft auch Eingabeaufforderung, Konsole oder Terminal genannt ist in der Regel das CLI (Command Line Interface), wo Befehle eingegeben werden können. Vereinzelt kommen auch Shells mittels grafischer Oberfläche, auch GUI (Graphical User Interface) genannt, wie sie vom MAC-OS her bekannt sind (Bsp. Finder).
Die Shell erlaubt dem Benutzer den Zugriff auf das Betriebssystem.

Ein anderer Begriff für die Shell ist auch der Kommandozeileninterpreter. Beim Scripten oder Programmieren von Scripts sprechen wir oft vom Kommandozeileninterpreter.

Hier ein Auszug bekannter Scriptsprachen und Shell's:

Unter Windows:
- CMD
- Command (von DOS bis Windows SE)
- Basic
- Visual Basic (VB)
- Visual Basic for Applications (VBA)
- Windows PowerShell

Unter Unix / Linux:
- Bash-Shell
- Bourne-Shell
- C-Shell
- Korn-Shell
- K-Shell
- X-Shell (nicht zu verwechseln mit dem X-Term!)
- Z-Shell

Systemunabhängige Script-Programmiersprachen:
- AutoIt
- Groovy
- JavaScript
- KiXtart
- Lisp
- Perl
- PHP
- Phyton
- TACL
- Tcl
- XML
- XSLT

Syntax und Semantik

Wir hören oft die Begriffe Syntax und Semantik. Was bedeuten diese zwei Begriffe genau?

Syntax

Ein System von Regeln, nach denen erlaubte Konstruktionen bzw. wohlgeformte Ausdrücke aus einem grundlegenden Zeichenvorrat (dem Alphabet) gebildet werden.
Syntax ist die Definition aller zulässigen Wörter / Programme, die in einer Sprache formuliert werden können

Kurz: Syntax beschreibt die verlangte Zusammenstellung eines Befehls inkl. Klammern und weiteren Zeichen.

Semantik

Eine Informationsfolge die Bedeutung dieser Informationsfolge.
Bedeutung der zulässigen Wörter / Programme. Syntaktische falsche Wörter / Programme haben keine Semantik.

Kurz: Semantik beschreibt die Bedeutung eines Wortes resp. Befehls.

Editor

Egal mit welcher Programmier- oder Scriptsprache wir arbeiten, wir brauchen einen Editor. Höhere Programmiersprachen liefern in der Regel eine ganze Entwicklungsumgebung mit integriertem Editor an. Wenn man mit mehreren verschiedenen Scriptsprachen arbeitet, empfiehlt es sich, einen Editor zu wählen, welcher mit allen Scriptsprachen umgeben kann und natürlich, dass man diesen Editor gut kennt.
Es gibt sehr viele Editoren in Internet. Der „richtige“ Editor erleichtert die Arbeit extrem und hilft beim Programmieren und bei der Fehlersuche.
„Mein“ Editor ist der EditPad Pro von JG Edwards.
Dieser Editor ist extrem flexibel und beliebig anpassbar, unterstützt Regular Expressions, kann direkt jedes beliebige Tool zum Prüfen (Debuggen) aufrufen und vieles mehr.

Ein paar Tipps zur Einstellung des EditPad Pro ist im Anhang (Kleine Tipps zum Editor auf Seite 30) zu finden. (noch in Bearbeitung!)

Für Windows PowerShell arbeite ich gerne mit EditPad Pro in Kombination mit dem ISE Editor (Integrated Scripting Environment) von Windows PowerShell

Varianten Scriptsprachen

Folgende Scriptsprachen werden in diesem Dokument kurz vorgestellt:

- CMD
- PowerShell
- Perl
- KIX
- VBA
- PHP

CMD / Batch

Einführung

Wenn wir als erstes das CMD öffnen erscheint folgendes Fenster:

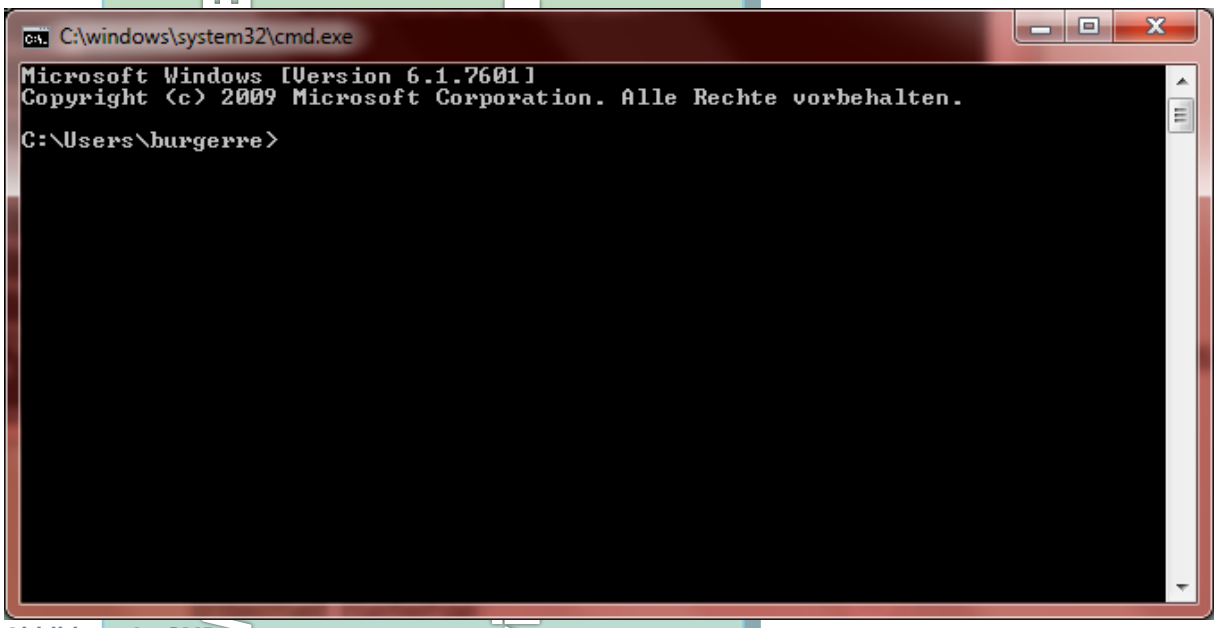


Abbildung 1 - CMD

In der ersten Zeile sehen wir die Version des Betriebssystems resp. vom CMD.
Anschliessend kommt das Prompt.

Das Prompt

Das Prompt ist eine vordefinierte Variabel, welche jederzeit geändert werden kann.
Mit dem Befehl *set* kann ich Variablen setzen, abfragen und auch wieder löschen.

Anzeigen / abfragen:

```
C:\Users\burgerre>set prompt
PROMPT=$P$G
```

Setzen / überschreiben

```
C:\Users\burgerre>set prompt=Mein Prompt:

Mein Prompt:
```

Folgende Möglichkeiten gibt es nebst Klartext und Umgebungsvariablen den Prompt zu definieren:

PROMPT [Text]

Text	Bezeichnet die neue Eingabeaufforderung.

Sie kann aus normalen Zeichen und folgenden Sonderzeichen bestehen:

$A	& (Kaufmännisches Und)
$B	| (Verkettungszeichen)
$C	((Klammer auf)
$D	Aktuelles Datum

$E	Escapezeichen (ASCII-Code 27)
$F	) (Klammer zu)
$G	> (Größer-als-Zeichen)
$H	Rückschritt (löscht vorangehendes Zeichen)
$L	< (Kleiner-als-Zeichen)
$N	Aktuelles Laufwerk
$P	Aktuelles Laufwerk und Pfad
$Q	= (Gleichheitszeichen)
$S	(Leerzeichen)
$T	Aktuelle Zeit
$V	Windows Versionsnummer
$_	Wagenrücklauf und Zeilenvorschub
$$	$ (Dollarzeichen)

Wenn die Befehlserweiterungen aktiviert sind, unterstützt der PROMPT-Befehl die folgenden zusätzlichen Formatierungszeichen:

$+	Keine oder mehr Pluszeichen (+), abhängig von der Anzahl der gespeicherten PUSHD-Verzeichnisse, wobei ein Zeichen pro Ebene ausgegeben wird.
$M	Zeigt den Remotenamen, der dem aktuellen Laufwerkbuchstaben zugeordnet ist, an oder nichts, wenn es sich bei dem aktuellen Laufwerkbuchstaben um kein Netzlaufwerk handelt.

Pfade

Wir unterscheiden grundsätzlich zwischen absolutem Pfad (Path) und relativem Pfad.

Absoluter Pfad

Absolute Pfade sind von überall her klar definiert und eindeutig.
Beispiele:

```
C:\windows\system32
\\10.10.0.1\austausch\inf2c
http://www.sbb.ch/freizeit-ferien/
```

Relativer Pfad

Relative Pfade sind vom aktuellen Standpunkt her gesehen und daher nicht immer eindeutig.
Beispiele:

```
\windows\system32 (kann auch auf Laufwerk d: sein oder auf Server
\\10.10.0.1\austausch\)
../../images/logo.jpg
```

Interne / externe Befehle

Wir unterscheiden zwischen internen und externen Befehlen oder Kommandos wie folgt:

Internet Befehle:

Befehle, welche im CMD intern sind.
cls, dir, del, net, etc.

Externe Befehle:

Programme die aufgerufen werden (abgelegt als exe, com, bat, etc.)
Ping, nslookup, route, format, tree, chcp, chkdsk etc.

Kleine Scripts

Starten wir mal mit ein paar Beispielen um uns an das Scripting mit CMD heran zu tasten. Später in diesem Script befassen wir uns noch mit der Windows PowerShell, KiXtart und Perl. Weitere Scriptsprachen folgen.

Starten wir mit CMD:

Variablen

Egal mit welcher Programmiersprache wir arbeiten, Variablen sind unumgängliche Helferlein für noch so kleine Scripte. Wir kennen dies aus der Mathematik, wo wir oft x oder i als Variable einsetzten, oder in der Geometrie: $a^2 + b^2 = c^2$ etc.

Hier brauchen wir die Variablen auch als Platzhalter, welche im Verlauf des Programmes verändert werden können oder gar müssen.

Das Hochzählen von Variablen ist unter anderem bei Schleifen interessant.

```
set var1=0
set /a var1=%var1%+1
```

Spezielle Variablen:

Name	Replacement Value Used
%CD%	The current directory, not ending in a slash character if it is not in the root directory of the current drive
%TIME%	The system time in HH:MM:SS.mm format.
%DATE%	The system date in a format specific to localization.
%RANDOM%	A generated pseudo-random number between 0 and 32767.
%ERRORLEVEL%	The error level returned by the last executed command, or by the last called batch script.
%CMDEXTVERSION%	The version number of the Command Processor Extensions currently used by cmd.exe.
%CMDCMDLINE%	The content of the command line used when the current cmd.exe was started.

Ausgabe von Variablen

```
Echo Heute ist der %date% - Aktuelle Uhrzeit: %time%
```

Abfragen

If und if else sind die wichtigsten Abfragen im CMD.

Ist heute ein Arbeitstag? Dann:
    Ich wünsche ich dir einen schönen Tag
Sonst:
    Ich wünsche dir ein schönes Weekend

Ist entspricht dem Befehlt IF und sonst dem Befehl ELSE.
Mit diesem Konstrukt können wir schon sehr viele Aufgaben lösen.

Nachfolgendes Beispiel fragt ab, ob die erste Tastatureingabe gleich ist wie zweite Tastatureingabe (Wird zum Beispiel gebraucht um Mailadressen oder Passwörter zu bestätigen). Ist die Eingabe gleich, wird dies bestätigt, sonst wird gemeldet, dass dies nicht gleich ist.

Beispiel:

```
@echo off
REM ********************************************************
REM Autor: Reto Burger
REM Create: 5.Sep.2015 20:54
REM Script: Abfragen einer Variablen und vergleichen
REM Version 1
REM ********************************************************

echo Bitte um Eingabe der Variable 1:
REM Tastatureingabe wird der Variablen var1 zugewiesen
set /p var1=

REM Variante 2 zur Tastatureingabe mit Textaufforderung:
set /p var2= "Bitte um Eingabe der Variable2: "

echo die Variable1 ist %var1%
echo die Variable2 ist %var2%

if "%var1%"=="%var2%" (
      echo Variable 1 "%var1%" ist gleich Variable2 "%var2%"
) ELSE (
      echo Variable 1 "%var1%" ist NICHT gleich Variable2 "%var2%"
      )

pause
REM Programm wird geschlossen
exit
```

Batch-Operatoren und Umleitungen

Zeichen	Beschreibung
&	Befehlsverkettung: mehrere Batch-Befehle in einer Zeile können hintereinander ausgeführt werden Befehl1 & Befehl2
&&	Bedingte Befehlsverkettung: der zweite Befehl wird nur ausgeführt, wenn der erste Befehl erfolgreich war
|	Pipe Befehlsverkettung mittels "pipe": Der zweite Befehl bekommt die Ausgabe des ersten Befehls als Eingabe
||	Bedingte Befehlsverkettung: der zweite Befehl wird nur ausgeführt, wenn der erste Befehl fehlschlug
<	Umleitung der Eingabe (Bsp.: Befehl mit Information aus Datei.)
>	Umleitung der (Standard-)Ausgabe zu einem anderen Ziel. Wenn das Ziel eine Datei ist, wird diese neu angelegt (falls die Datei schon existiert, wird sie zuvor gelöscht) Es kann aber auch nach NUL (Die Ausgabe verschwindet) oder CON(Bildschirm) umgeleitet werden Bsp: dir DieseDateiExistiertNicht.txt >stdout.txt 2>stderror.txt type EineDateiDieNichtExistiert.txt > UmgeleiteteAusgabe.txt 2>&1
>>	Umleitung der (Standard-)Ausgabe mit Anhängen des Textes (falls die Datei schon existiert; sonst wird die Datei wie bei > angelegt)

Beispiele

```
C: & cd \windows & dir
```

Die zwei & verknüpfen die folgenden drei Befehle:

C: wechselt zum Laufwerk c:
Cd \windows wechselt in das Verzeichnis c:\winsows
Dir listet alle Ordner und Dateien im Verzeichnis c:\windows auf

```
@echo off
dir c:\temp\*.* ¦ find „test“ > Antwort.txt
```

Verzeichnis c:\temp wird durchsucht und zwar alle Dateien.
Dieses Ergebnis wird gefiltert und alle Dateien, welche im Dateinamen test beinhalten (Achtung Gross-Kleinschreibund wird unterschieden > case-sensitive). Alles was gefunden wird, wird nun in die Datei Antwort.txt geschrieben. Ist noch keine Datei namens Antwort.txt vorhanden, wird diese neu erstellt, ansonsten überschrieben.

Zwei Dateien in eine Datei zusammen fügen kann ich zum einen mit dem normalen Copy Befehl

```
copy Datei_1.txt + Datei_2.txt Datei.txt
```

oder

```
type Datei_1.txt > Datei.txt 2> error.txt
type Datei_2.txt >> Datei.txt 2>> error.txt
```

das ganze jetzt auf eine Zeile

```
type Datei_1.txt > Datei.txt 2> error.txt & type Datei_2.txt >> Datei.txt 2>>
error.txt
```

Mit **2>** resp. **2>>** habe ich die Möglichkeit, Fehlermeldungen in eine Datei (Bsp. error.txt) umzuleiten. Das macht vor allem dann Sinn, wenn ich grössere Scripts habe und nicht nach jeder Kommandozeile nach Fehler suchen möchte.

Switch-Anweisung

Mit case…
Im CMD leider nicht möglich.

Schleifen

Es gibt verschiedene Arten von Schleifen:

- For
- While
- Do while

Wie unterscheiden sich diese unterschiedlichen Schleifen:

For	Die Zählschleife, eine Sonderform der vorprüfenden Schleife
While	Die vorprüfende oder kopfgesteuerte Schleife
Do while	Die nachprüfende oder fußgesteuerte Schleife

Im CMD gibt es nur die for-schleife.

For-Schleife

Wir kennen unter CMD mehrere Einsatzmöglichkeiten von Schleifen:

Die „normale“ Schleife ermöglicht zum Beispiel das durchsuchen von Daten, Verzeichnissen etc.

```
REM Suche nach der Datei temp.log unter c:\Windows\ inkl. Unterverzeichnissen.
for /F "tokens=*" %%f in ('dir /S /b C:\Windows\temp.log') do (
        echo "%%f".
)
```

Die Zählschleife ist erlaubt etwas eine gewisse Anzahl mal zu wiederholen.

```
REM Zähle von 0 bis 20 in zweierschritten (also nur gerade Zahlen)
for /L %%N IN (0, 2, 20) DO echo Zahl: %%N
```

setlocal

Erweiterungen

```
SETLOCAL {EnableDelayedExpansion | DisableDelayedExpansion}  {EnableExtensions |
DisableExtensions}
```

EnableExtensions	Aktiviert die Befehlswerweiterung
DisableExtensions	Deaktiviert die Befehlswerweiterung
EnableDelayedExpansion	Aktiviert die verzögerte Erweiterung von Umgebungsvariablen. Diese Argumente haben Vorrang gegenüber den Optionen "CMD /V:ON" oder "/V:OFF". Hier kann anstelle des % ein ! gesetzt werden vor und nach dem Variablennamen.
DisableDelayedExpansion	Aktiviert die verzögerte Erweiterung von Umgebungsvariablen. Diese Argumente haben Vorrang gegenüber den Optionen "CMD /V:ON" oder "/V:OFF".

Mit **endlocal** wird das setlocal zurückgesetzt und somit alle weiteren Erweiterungen deaktiviert.

Variablen beschneiden

Variablen könne aus- oder beschnitten werden mittels ~ (Tildezeichen)

```
@echo off
setlocal enableextensions enabledelayedexpansion

set tageszeit=%time%
REM Echo %tageszeit%

set hh=!tageszeit:~0,2!
set mm=!tageszeit:~3,2!

Echo Die aktuelle Stunde ist: !hh!
Echo Die aktuelle Minute ist: !mm!
endlocal
```

Regular Expression / REGEX

Einführung

Regular Expressions scheint zu Beginn recht unleserlich, hat man aber ein paar Stunden investiert, zeichnet es sich als unverzichtbare Erweiterung in fast allen Programmiersprachen ab.
Nebst Implementierungen in vielen Programmier- und Scriptsprachen verfügen auch viele Editoren über die Funktionen der Regulären Ausdrücke (Regex) oft in der Funktion zum Suchen oder Suchen und Ersetzen in Dokumenten. Solche Regular Expressions sind viel mächtiger als das einfache Suchen und Ersetzten mit „normalen“ Wildcards.

Ein **Cheat-Sheet** findet sich im Anhang des Dokumentes.

Beispiele

Suche alle Wörter, welche mit k enden:

```
K$
```

Suche alle Wörter ausschliesslich aus den Zeichen a bis f bestehen

```
^[a-f]*$
```

Suche alle Wörter, welche eine Zeichenfolge genauso nochmals wiederholt

```
(...).*\1
```

Suche Begriffe ohne Doppelbuchstaben mit Ausnahme wenn das Wort „ef“ enthält.

```
^(?!.*(.)\1)|ef
```

Beispiel mit PowerShell und Regex:

Suche die Version 2. und ersetze mit 3.

```
"Einführung in PowerShell 2.0" -replace "\d\.", "3."
```

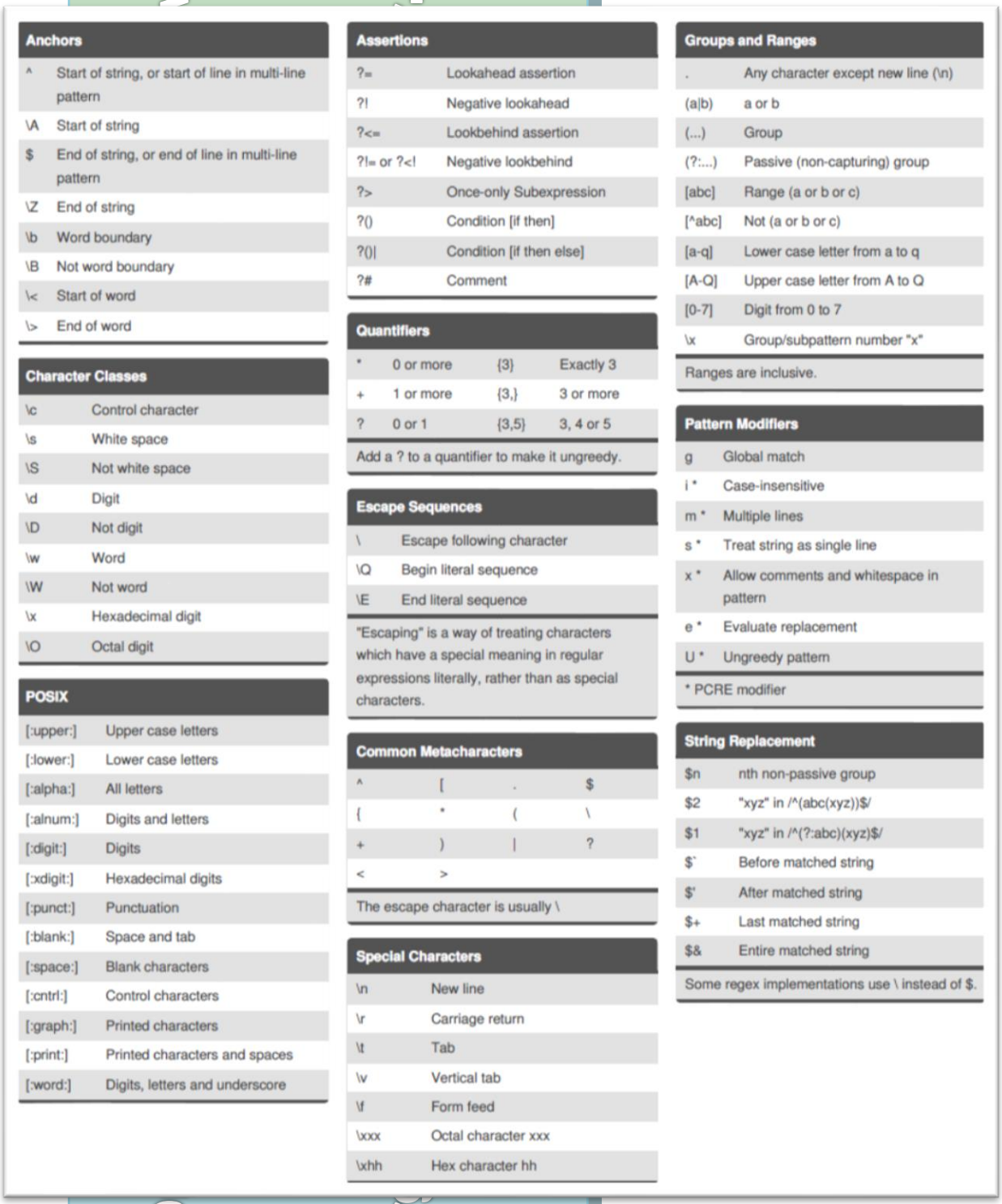
In obigem Beispiel wird nicht nur ein Textmuster gesucht, sondern gleich noch ersetzt. Dieser Aufruf gibt anders als -match keinen Boolschen Wert zurück, der über das Zutreffen des Musters informiert, sondern die geänderte Zeichenkette. Im obigen Beispiel wird aus "2.0" ein "3.0".

Kapitel wird noch erweitert!

Übungen (mit EditPad)

Hier kann eine Testversion von EditPad geladen werden:
http://download.jgsoft.com/editpad/SetupEditPadProDemo.exe

Regular Expressions - Cheat Sheet



Anchors

^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

POSIX

[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions

?=	Lookahead assertion
?!	Negative lookahead
?<=	Lookbehind assertion
?!= or ?<!	Negative lookbehind
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences

\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters

^	[	.	$
{	*	(	\
+	)	|	?
<	>		

The escape character is usually \

Special Characters

\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed
\xxx	Octal character xxx
\xhh	Hex character hh

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers

g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

* PCRE modifier

String Replacement

$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string

Some regex implementations use \ instead of $.

Abbildung 2 - Regular Expressions - Cheat Sheet

Windows PowerShell PS

Einführung

Über Windows PowerShell könnte man ein eigenes grosses Buch schreiben. Verschiedene gute Literatur ist auch auf dem Markt erhältlich. Hier soll nur eine Einführung von Windows PowerShell gezeigt werden. Für all jene, die Lust auf mehr Wissen mit der Windows PowerShell bekommen und damit ausführlich programmieren wollen, empfehle ich entsprechende Literatur oder Schulungen zu besuchen.

Auch wenn es Windows PowerShell heisst, schreiben wir nachfolgend einfach PowerShell.

PowerShell ist ein fester Bestandteil vom Betriebssystem Windows und verschiedene Aktionen können nur noch mittels PowerShell getätigt werden. PowerShell ist daher auch ein Management Tool für alle neuen Windows Versionen (Server und Clients).
Das PowerShell Management wird seitens Microsoft vor der GUI entwickelt. Das sieht man oft bei Programmen wie MS Exchange und MS SQL etc. Die PowerShell kann auf API zugreifen wie .NET und COM. PowerShell hat Schnittstellen zu WMI, CIM und ADSI. Auch Drittherstellen wird die Möglichkeit geboten eine Schnittstelle zu schreiben für die PowerShell.

PowerShell ist eine DLL und arbeitet mit dem HostPorgramm PowerShell.exe und verarbeitet Objekte (keinen Text!).

Eine weitere Stärke von der Windows PowerShell ist die Unterstützung von Regular Expressions.

Versionen von Windows PowerShell

Anzeigen der aktuellen Windows PowerShell-Version mit:

```
$PSVersionTable
```

Zeigt:

```
Name                           Value
----                           -----
PSVersion                      5.0.10586.0
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.10586.0
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

Oder

```
$PSVersionTable.PSVersion
```

Zeigt:

```
Major  Minor  Build  Revision
-----  -----  -----  --------
5      0      10586  0
```

Bevor wir starten

PowerShell hat Sicherheitseinstellungen definiert. Diese können mit folgendem Befehl angeschaut werden:

```
Get-ExecutionPolicy -list
```

Zeigt:

```
        Scope ExecutionPolicy
        ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process       Undefined
  CurrentUser       Undefined
 LocalMachine       Undefined
```

Beschreibung der Policy / Regel

Policy	Beschreibung
Restricted (Eingeschränkt)	Ist die Standardeinstellung Es können keine Skripts ausgeführt werden. Windows PowerShell kann nur im interaktiven Modus genutzt werden. Es werden keine Konfigurationsdateien geladen und keine Scripts ausgeführt (Standard)
AllSigned (Vollständig signiert)	Nur von einem vertrauenswürdigen Autor erstellte Skripts können ausgeführt werden. Signierte Scripts und Konfigurationsdateien von einem vertrauenswürdigen Herausgeber werden ausgeführt. Auch lokal erstellte Scripts müssen signiert sein.
RemoteSigned (Remote signiert)	Heruntergeladene Skripts müssen von einem vertrauenswürdigen Autor signiert werden, bevor sie ausgeführt werden können. Aus dem Internet heruntergeladenen Scripts und Konfigurationsdateien müssen von einem vertrauenswürdigen Herausgeber signiert sein.
Unrestricted (Uneingeschränkt)	Es gibt überhaupt keine Einschränkungen. Alle Windows PowerShell-Skripts können ausgeführt werden. Alle Konfigurationsdateien und alle Scripts werden ausgeführt. Bei nicht signierten Scripts aus dem Internet muss man jede Ausführung am Prompt bestätigen
Bypass	Keinerlei Blockade, keine Warnungen oder Prompts. (Achtung!!)
Undefined (Undefiniert)	Entfernt die gerade zugewiesene Richtlinie (nur für lokal zugewiesene Richtlinien, nicht für GPO-applizierte)

Beschreibung der Scope / Anwendungsbereich

Scope	Beschreibung
MachinePolicy	Ausführungsrichtlinien sind via Gruppenrichtlinie definiert
UserPolicy	Ausführungsrichtlinien sind via Gruppenrichtlinie definiert
Process	Die Ausführungsrichtlinie wirkt sich nur auf die aktuelle Windows Powershell-Prozess.
CurrentUser	Die Ausführungsrichtlinie wirkt sich nur auf den aktuellen Benutzer.
LocalMachine	Ist die Standardeinstellung Die Ausführungsrichtlinie wirkt sich auf alle Benutzer des Computers.

Zeigt die Einstellungen für den aktuellen Benutzer:

```
Get-ExecutionPolicy -Scope CurrentUser
```

Mit folgendem Befehl können wir das Ausführen von PowerShell Scripten frei schalten:

```
# Setze die Policy für den aktuellen User
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

Erste kleine Übungen

```
# Zeigt das Homeverzeichnis von PowerShell
$PSHOME
```

```
# Was im CMD das Echo ist in der PowerShell das Write-Host
Write-Host "Hello World!"
```

```
# Was im CMD das Echo ist in der PowerShell das Write-Host
# `n macht einen Zeilenumbruch
Write-Host "Hello World! `nHeute ist ein toller Tag!"
```

```
# Abfrage der Tastatur (Tastatureingabe)
Read-Host "Hi, wie geht es dir"
```

Eingaben via Tastatureingabe einer Variablen zuordnen:

```
# Definiere in der Variablen „Variable1“ via Tastatur einen Text oder Zahl
# Die Variable ist sichtbar in allen Bereichen der Script-Datei (mit $Script:)
$Script:Variable1 = Read-Host "Gebe mir einen Text ein:"
Write-Host "Der eingegebene Text ist: $Variable1"
```

Anstelle von Dir /s etc. kann mittels Get-ChildItem nach Dateien gesucht werden. Hier ein paar Beispiele:

```
#Suche alle PDF-Dateien vom aktuellen Verzeichnis und allen Unterverzeichnissen
Get-ChildItem -Recurse | where {$_.extension -eq ".pdf"}
```

```
#Suche alle docx-Dateien mit Datum von 2015 vom aktuellen Verzeichnis und allen
Unterverzeichnissen
Get-ChildItem -Recurse | where {$_.extension -eq ".docx" -and
$_.LastWriteTime.Year -gt 2015}
```

```
#Suche alle Text- und Worddateien (altes und neues Format)
#ohne ".pdf (macht hier keinen Sinn!!) und suche nach dem Inhalt Modul 122
#PS: Nach -Pattern werden Regular Expressions eingetragen!
Get-ChildItem *.txt, *.doc, *.docx -exclude *.pdf -recurse | Select-String -
Pattern "Modul 122"
```

```
#Suche alle Dateien mit der Erweiterung .txt und benenne sie wie folgt um: Suind
Leerzeichen im Dateinamen, ersetze diese mit einem Underline „_“.
Get-ChildItem *.txt  | Rename-Item -NewName {$_.Name -replace ' ','_'}
```

Arbeiten mit Variablen

Variablen beginnen mit einem $
Bei Variablen spielt die Gross- Kleinschreibung keine Rolle, ist also nicht case-sensitiv (also case-insensitive).

Man kann einer Variablen einen Wert zuweisen, indem man ihren Namen, einen Zuweisungsoperator und einen Ausdruck kombiniert.

```
$v = "Meine Variable"
```

oder

```
$x = 1 + 1
```

Mit der PowerShell ist es möglich, Variablen einen ganz bestimmten Typ zuzuweisen. Dabei wird der gewünschte Typ vor das Dollarzeichen der Variablenbezeichnung gesetzt. Zum Beispiel:

```
[int]$x = 5
```

Alle Dateitypen aus dem .NET-Framework sind bekannt, wie:

Zahlen: [int], [long], [double], [decimal], [float], [single], [byte]
Zeichen: [string], [char]
Speziell noch: [XML], [ADSI]

Anzeigen von Umgebungsvariablen:
Das geht nicht mit set wie unter cmd. Set ist ein Alias für Set-Variable. Um Umgebungsvariablen abzufragen verwenden Sie:

```
Get-ChildItem env:
```

Möchte ich eine bestimmte Umgebungsvariable, wie z.B. Computername abfragen gebe ich folgendes ein:

```
Get-ChildItem env:computername
```

Oder

```
$env:COMPUTERNAME
```

Scope

Weiter kann man beim Anlegen der Variablen auch die Sichtbarkeit definieren. Der genannte Scope kann festgelegt werden. Dabei wird bestimmt, in welchen Bereichen des gesamten Codes die Variable gültig ist. Folgende Optionen stehen zur Verfügung:

global: sichtbar in allen Bereichen
script: sichtbar in allen Bereichen der Script-Datei
local: sichtbar nur im aktuellen Bereich und darin eingebetteter Bereich (z.B. Funktionen)
private: sichtbar nur im aktuellen Bereich (z.B. innerhalb einer Funktion)

Soll beispielsweise eine Variable überall sichtbar sein, wird folgende Zeile programmiert:

```
$global:x = 100
```

Wird der Scope nicht explizit angegeben, gilt die Sichtbarkeit local.

Zuweisungsoperatoren

Die PowerShell kennt folgende Zuweisungsoperatoren

Operator	Beschreibung
=	Sets the value of a variable to the specified value.
+=	Increases the value of a variable by the specified value, or appends the specified value to the existing value.
-=	Decreases the value of a variable by the specified value.
*=	Multiplies the value of a variable by the specified value, or appends the specified value to the existing value.
/=	Divides the value of a variable by the specified value.
%=	Divides the value of a variable by the specified value and then assigns the remainder (modulus) to the variable.
++	Increases the value of a variable, assignable property, or array element by 1.
--	Decreases the value of a variable, assignable property, or array element by 1.

Vergleichsoperatoren

eq (equals, =)
ne (not equals, != / <>)
gt (greater than, >)
ge (greater or equal, >=)
lt (lower than, <)
le (lower or equal, <=)

Bedingungen

Nachfolgend finden Sie die Konstrukte der Bedingungen wie: if / elseif / else, switch, while, und for:

Dot-Sourcing.

Möchte ich eine Variable auch nach Beendigung einer Funktion oder eines Scripts nutzen können, starte ich die Funktion oder das Script mit einem . (Punkt).
Wenn ein Skript im aktuellen Bereich dot-sourced ist sind alle Funktionen, Aliase und Variablen, die im Skript erstellt werden im aktuellen Bereich zur Verfügung.

Beispiele

```
. xxxx
```

Die IF Bedingung

```
if(condition) {…}
elseif(condition) {…}
else {…}
```

Beispiele

```
# Zwei Variablen eingeben und vergleichen.
# Sind Variablen gleich > Melde Variablen sind gleich
# Sind Variablen UNgleich > Melde Variablen sind UNgleich

$Var1 = Read-Host "Gebe mir einen Text 1:"
$Var2 = Read-Host "Gebe mir einen Text 2:"

if ($var1 -eq $var2) {Write-Host "Die Variablen sind gleich"}
else {Write-Host "Die Variablen sind UNgleich"}

Write-Host "$var1 und $var2"
```

Die switch Bedingung

```
switch(expression) {
value1 {…}
value2 {…}
default {…}
}
```

Die while Bedingung

```
while(expression) {
… }
```

Die for Bedingung

```
for([initializer]; [condition]; [iterator]) {
…}
```

Beispiel

```
Folgende Schleife wird 10x durchlaufen:
for($i = 1; $i -le 10; $i += 1)
```

Die foreach Bedingung

```
foreach(identifier in collection) {
…}
```

Funktionen und Filter

Wie aus vielen anderen Programmiersprachen bekannt, erlaubt es auch die Windows PowerShell, mehrere Statements zu einer Funktion zusammen zu fassen.

Eine Funktion vereinfacht die Wiederverwendung von Code und hilft ein Script zu strukturieren.
Wenn auch die Funktionen in PowerShell jenen von VB-Script und PHP ähnlich sind, gibt es gravierende Unterschiede.

Unabhängig davon, ob eine Funktion nur ein einzelnes Statement oder eine komplexe Abfolge von Anweisungen umfasst, lässt sich der in ihr enthaltene Code einfach durch den Aufruf des Funktionsnamens inklusive der eventuell erforderlichen Parameter ausführen.
Das vermeidet redundante Script-Blöcke und erleichtert die Pflege des Codes.

Unterschied zwischen Funktionen und Filter

Folgt!!

Arbeiten mit Funktionen

Aufruf einer Funktion:

```
function global:Get-BenutzerEingabe
```

Auch hier kann der Scope angegeben werden.

```
function [<scope:>]<name> [([type]$parameter1[,[type]$parameter2])]{

	param([type]$parameter1 [,[type]$parameter2])
	dynamicparam {<statement list>}

	begin {<statement list>}
	process {<statement list>}
	end {<statement list>}

}
```

filter name {
param($parameter1, $parameter2, …)
}

Beispiele mit Funktionen:

```
# Einstieg in Funktionen

# Bereich Funktionen

Function Ausgab1{
	Write-Host "Das ist die Ausgabe 1"
}

Function Ausgab2{
    Write-Host "Das ist die zweite Ausgabe ;-)"
}

# Bereich Main

$Varianten = Read-Host "Welche Funktion soll gestartet werden? [1] Ausgabe 1 oder
[2] Ausgabe 2?"
```

```
if($Varianten -eq "1"){
    . Ausgab1
}elseif($Varianten -eq "2"){
    . Ausgab2
}else{
            Write-Host "Ungültige Auswahl"
}
```

Erweiterung unseres Beispiels
Wenn ich eine ungültige Eingabe machen, soll nochmals abgefragt werden nach Funktion 1 oder 2

```
#Übungsbeispiel mit Funktionen

# Bereich Funktionenen

Function Ausgabe1{
    Write-Host "Das ist die Ausgabe 1"
}

Function Ausgabe2{
    Write-Host "Das ist die zweite Ausgabe ;-)"
}
# Bereich Main

function Abfrage{
$Varianten = Read-Host "Welche Funktion soll gestartet werden? [1] Ausgabe 1 oder
[2] Ausgabe 2?"

if($Varianten -eq "1"){
    . ausgabe1
}elseif($Varianten -eq "2"){
    . ausgabe2
}else{
    Write-Host "Deine eingabe ist ungültig!!!"
    Abfrage
}
}
Abfrage
```

```
# Beispiel mit Parameterübergaben

Function ReturnString ($x)
{
    $x = $x +1
    return $x
}

Write-Host "Resultat:"
ReturnString -x 1
```

```
#Beispiel 1 einfache Addition

Function ReturnString ($x,$y)
{
    $ergebnis = [int]$x + [int]$y
```

```
    return $ergebnis
}


$x = Read-Host "erste Zahl"
$y = Read-Host "zweite Zahl"

Write-Host "Resultat:"
ReturnString -x $x -y $y
```

```
# Beispiel 2 mit Parameterübergaben

Function ReturnString2 ($x,$x2)
{
    $z = $x + " " + $x2
    return $z
}

Write-Host "Resultat:"
ReturnString2 -x "hello" -x2 "world"
```

```
# Beispiel 3 mit Parameterübergaben und umbenennen (Grossschreibung)

Function ReturnString3 ($x,$x2)
{
    $x = $x.Replace("h", "H")
    $x2 = $x2.Replace("w", "W")
    $z = $x + " " + $x2
    return $z
}

Write-Host "Resultat:"
ReturnString3 -x "hello" -x2 "world"
```

Array's in PowerShell

Auch PowerShell kennt wie fast jede moderne Scriptsprache den Datentyp Array.
Der Standardtyp von PowerShell für Arrays ist „Variant".
Ich gehe davon aus, dass Sie das Prinzip von Array kennen.

Werte einem Array zuweisen:

```
$Arbeitstage = @(„Montag“,“Dienstag“,“Mittwoch“,“Donnerstag“,“Freitag“)
```

Array erweitern:

```
$Arbeitstage = $ Arbeitstage + „Samstag“
```

oder

```
$Arbeitstage += „Samstag“
```

Arrays verbinden:

```
$Arbeitstage += $Freitage
```

Prüfe, ob eine Variable ein Array ist:

```
(Get-Process) -is [array]
```

Arrays ausgeben

Einzelne Inhalte eines Array ausgeben (erstes beginnt mit 0!)

```
$Arbeitstage[0,2,4]
```

Oder einen Bereich ausgeben:

```
$Arbeitstage[0..2]
```

Das ganze Array ausgeben:

```
Write-Host $Arbeitstage
```

Um auf das letzte Element im Array zuzugreifen:

```
$Arbeitstage[-1]
```

Antwort: Freitag

Elemente in Arrays finden

Ist Mittwoch im Array innerhalb der Liste zwischen 1 und 3?

```
$Arbeitstage[1..3] -contains "Mittwoch"
```

In unserem Fall ist die Antwort: true

Der Parameter –contains berücksichtigt noch zusätzlich Gross- Kleinschreibung!

Möchte ich mit Wildcards arbeiten und im Bereich zwischen 1 bis 3 nach Werten suchen welche mit tag enden, gebe ich folgendes ein:

```
$Arbeitstage[1..3] -like "*tag"
```

Antwort: Dienstag Donnerstag

Arrays sortieren

Sortiere den Inhalt eine Arrays alphabetisch nur für den Output:

```
$Arbeitstage | sort
```

Sortiere den Inhalt eine Arrays alphabetisch, so dass es im Array neu geordnet ist:

```
$Arbeitstage = $Arbeitstage | sort
```

Arrays und Elemente löschen

Ganzes Array löschen:

```
$Arbeitstage = $null
```

Um ein einzelnes Element zu löschen, muss man mit Filter arbeiten:

```
$Arbeitstage = $Arbeitstage | where {$_ -ne "Mittwoch"}
```

Perl

Kapitel noch in Vorbereitung > folgt.

KIX

Kapitel noch in Vorbereitung > folgt.

VBA

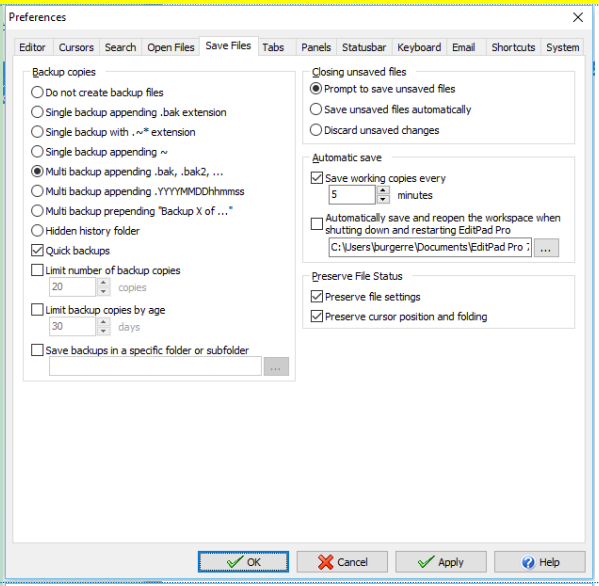
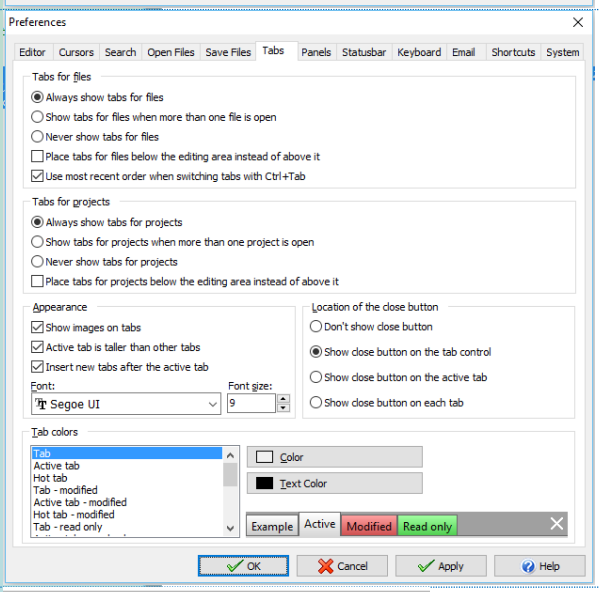
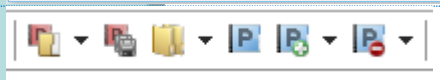
Kapitel noch in Vorbereitung > folgt.

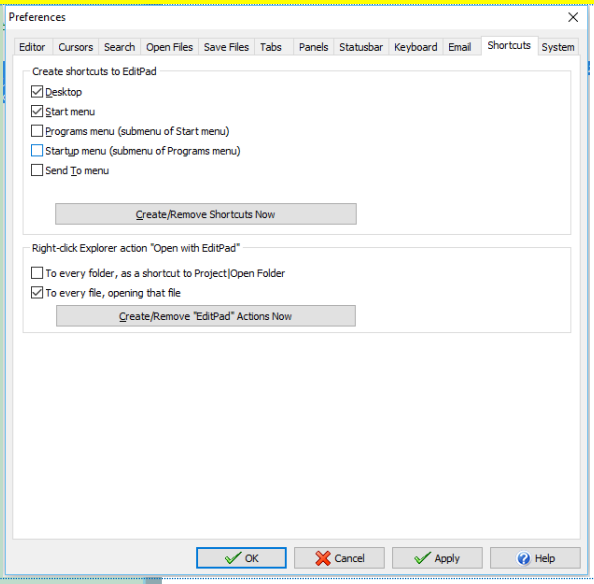
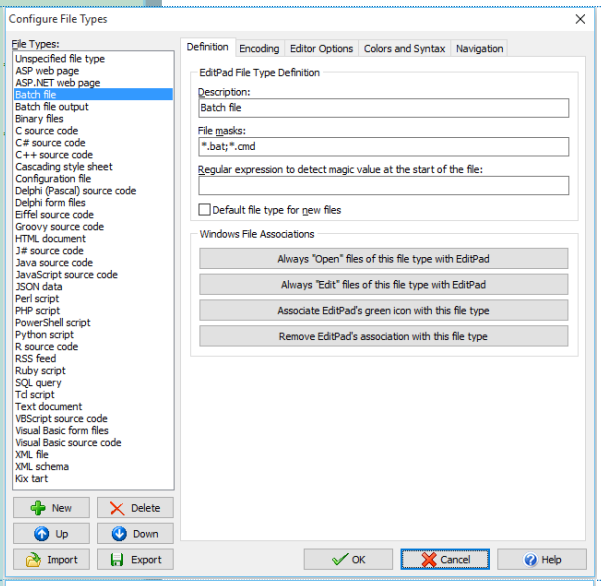
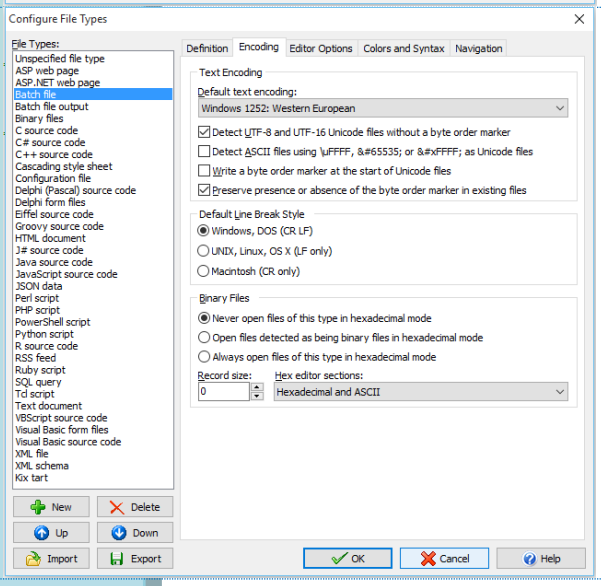
PHP

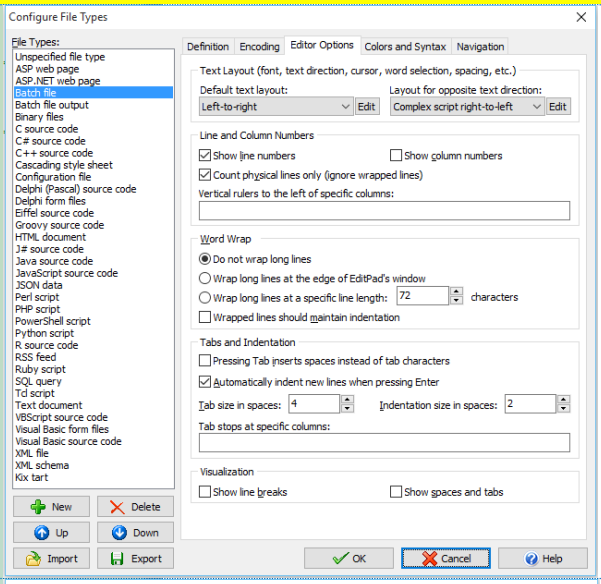
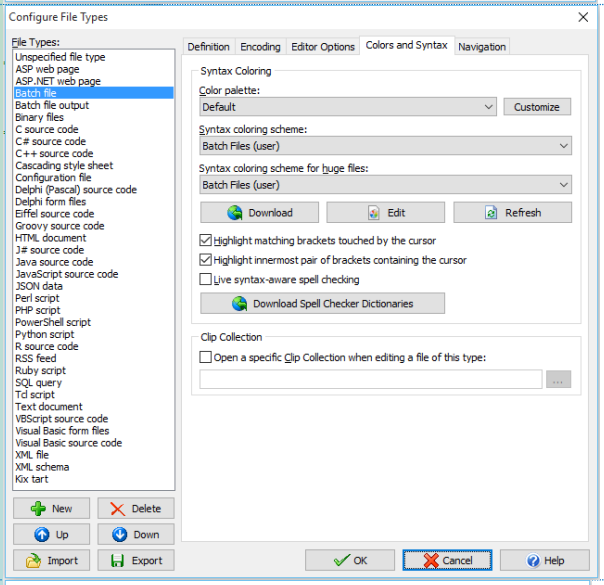
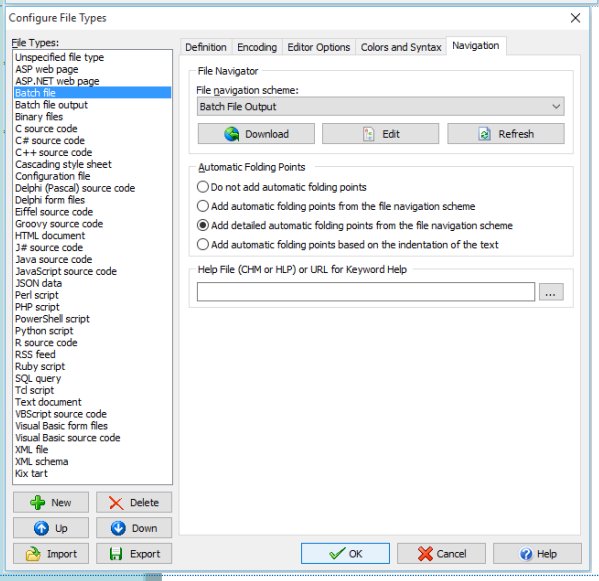
Kapitel noch in Vorbereitung > folgt.

Anhang

Kleine Tipps zum Editor

Grundeinstellungen	Bild
Anstelle einer Versionsverwaltung kann eingestellt werden, dass alle jedes Mal beim Speichern die Vorversion als *.bakX gespeichert wird. X wird dann beliebig hochgezählt.	
Um das Projekt-Tab immer zu sehen, empfiehlt es sich unter Tabs for projects die Option „Always show tabs for projects“ zu aktivieren. So kann man mehrere Dateien einem Projekt zuordnen.	
Menübar für die Projektzuordnungen und zum Sperren von Projekten.	

Grundeinstellungen	Bild
Wo soll der Editor aufgerufen werden kann im oberen Feld definiert werden. Das Content Menü (rechte Maustaste) kann im unteren Bereich aktiviert oder wieder deaktiviert werden.	
Filetype Einstellungen Beschreibung folgt….!	
	

Grundeinstellungen	Bild
	
	
	
Tools Zum direkten starten und testen von Scripts	

Weitere gute Tools von JG-Soft:

- AceText
- PowerGrep
- Regex Magic
- Regex Buddi
- DeployMaster
- HelpScribble

Übersicht aller CMD-Befehle

Geben Sie HELP "Befehlsname" ein, um weitere Informationen zu einem bestimmten Befehl anzuzeigen. Hier ein Auszug unter Windows 7:

Befehl	Beschreibung
ASSOC	Zeigt Dateierweiterungszuordnungen an bzw. ändert sie.
ATTRIB	Zeigt Dateiattribute an bzw. ändert sie.
BREAK	Schaltet die erweiterte Überprüfung für STRG+C ein bzw. aus.
BOOTCFG	Legt Eigenschaften zur Steuerung des Startladenvorganges in der Startdatenbank fest.
CACLS	Zeigt Datei-ACLs (Access Control List) an bzw. ändert sie.
CALL	Ruft eine Batchdatei von einer anderen Batchdatei aus auf.
CD	Zeigt den Namen des aktuellen Verzeichnisses an bzw. ändert diesen.
CHCP	Zeigt die aktive Codepagenummer an bzw. legt sie fest.
CHDIR	Zeigt den Namen des aktuellen Verzeichnisses an bzw. ändert es.
CHKDSK	Überprüft einen Datenträger und zeigt einen Statusbericht an.
CHKNTFS	Zeigt die Überprüfung des Datenträgers beim Start an bzw. verändert sie.
CLS	Löscht den Bildschirminhalt.
CMD	Startet eine neue Instanz des Windows-Befehlsinterpreters.
COLOR	Legt die Hintergrund- und Vordergrundfarben für die Konsole fest.
COMP	Vergleicht den Inhalt zweier Dateien oder Sätze von Dateien.
COMPACT	Zeigt die Komprimierung von Dateien auf NTFS-Partitionen an bzw. ändert sie.
CONVERT	Konvertiert FAT-Volumes in NTFS. Das aktuelle Laufwerk kann nicht konvertiert werden.
COPY	Kopiert eine oder mehrere Dateien an eine andere Stelle.
DATE	Zeigt das Datum an bzw. legt es fest.
DEL	Löscht eine oder mehrere Dateien.
DIR	Listet die Dateien und Unterverzeichnisse eines Verzeichnisses auf.
DISKCOMP	Vergleicht den Inhalt von zwei Disketten.
DISKCOPY	Kopiert den Inhalt von einer Diskette auf eine andere Diskette.
DISKPART	Zeigt Eigenschaften von Festplattenpartitionen an bzw. konfiguriert sie.
DOSKEY	Bearbeitet Befehlseingaben, erinnert Windows-Befehle und erstellt Macros.
DRIVERQUERY	Zeigt den aktuellen Gerätetreiberstatus und die Eigenschaften an.
ECHO	Zeigt Meldungen an bzw. schaltet die Befehlsanzeige ein oder aus.

Befehl	Beschreibung
ENDLOCAL	Beendet den lokalen Gültigkeitsbereich von Umgebungsänderungen in einer Batchdatei.
ERASE	Löscht eine oder mehrere Dateien.
EXIT	Beendet das Programm CMD.EXE (Befehlsinterpreter).
FC	Vergleicht zwei oder mehr Sätze von Dateien und zeigt die Unterschiede an.
FIND	Sucht eine Zeichenkette in einer oder mehreren Datei(en).
FINDSTR	Sucht Zeichenketten in Dateien.
FOR	Führt einen angegebenen Befehl für jede Datei in einem Dateiensatz aus.
FORMAT	Formatiert einen Datenträger für die Verwendung mit Windows.
FSUTIL	Zeigt die Dateisystemeigenschaften an bzw konfiguriert sie.
FTYPE	Zeigt die Dateitypen an, die bei den Zuordnungen für die entsprechenden Dateierweiterungen verwendet werden bzw. ändert sie.
GOTO	Setzt den Windows-Befehlsinterpreter auf eine markierte Zeile in einem Batchprogramm.
GPRESULT	Zeigt Gruppenrichtlinieninformationen für Computer oder Benutzer an.
GRAFTABL	Ermöglicht Windows, Sonderzeichen im Grafikmodus anzuzeigen.
HELP	Zeigt Hilfeinformationen zu Windows-Befehlen an.
ICACLS	Anzeigen, Ändern, Sichern oder Wiederherstellen von ACLs für Dateien und Verzeichnisse.
IF	Verarbeitet Ausdrücke in einer Batchdatei abhängig von Bedingungen.
LABEL	Erstellt, ändert oder löscht die Bezeichnung eines Volumes.
MD	Erstellt ein Verzeichnis.
MKDIR	Erstellt ein Verzeichnis.
MKLINK	Erstellt symbolische Links und feste Links.
MODE	Konfiguriert ein Systemgerät.
MORE	Zeigt Ausgabe auf dem Bildschirm seitenweise an.
MOVE	Verschiebt ein oder mehrere Dateien von einem Verzeichnis in ein anderes.
OPENFILES	Zeigt Dateien, die von Remotebenutzern zur Dateifreigabe geöffnet wurden an.
PATH	Legt den Suchpfad für ausführbare Dateien fest o. zeigt ihn an.
PAUSE	Hält die Ausführung einer Batchdatei an und zeigt e. Meldung an.
POPD	Wechselt zu dem Verzeichnis, das durch PUSHD gespeichert wurde.
PRINT	Druckt eine Textdatei.
PROMPT	Ändert die Eingabeaufforderung.
PUSHD	Speichert das aktuelle Verzeichnis, und wechselt zu einem anderen Verzeichnis.
RD	Entfernt ein Verzeichnis.
RECOVER	Stellt lesbare Daten von einem beschädigten Datenträger wieder her.
REM	Leitet Kommentare in einer Batchdatei bzw. CONFIG.SYS ein.
REN	Benennt eine Datei bzw. Dateien um.
RENAME	Bennent eine Datei bzw. Dateien um.
REPLACE	Ersetzt Dateien.
RMDIR	Löscht ein Verzeichnis.
ROBOCOPY	Erweitertes Dienstprogramm zum Kopieren von Dateien und Verzeichnisstrukturen
SET	Setzt oder löscht die Umgebungsvariablen bzw. zeigt sie an.
SETLOCAL	Startet die Begrenzung des Gültigkeitsbereiches von Umgebungsänderungen in einer Batchdatei.
SC	Zeigt Dienste (=Hintergrundprozess) an bzw. konfiguriert sie.

Befehl	Beschreibung
SCHTASKS	Erstellt Zeitpläne für auf dem Computer auszuführende Befehle und Programme.
SHIFT	Verändert die Position ersetzbarer Parameter in Batchdateien.
SHUTDOWN	Ermöglicht lokales oder ferngesteuertes Herunterfahren des Computers.
SORT	Sortiert die Eingabe.
START	Startet ein eigenes Fenster, um ein bestimmtes Programm oder einen Befehl auszuführen.
SUBST	Ordnet einen Pfad einem Laufwerkbuchstaben zu.
SYSTEMINFO	Zeigt computerspezifische Eigenschaften und Konfigurationen an.
TASKLIST	Zeigt alle zurzeit laufenden Aufgaben inklusive der Dienste an.
TASKKILL	Bricht einen laufenden Prozess oder eine Anwendung ab oder beendet ihn bzw. sie.
TIME	Zeigt die Systemzeit an bzw. legt sie fest.
TITLE	Bestimmt den Fenstertitel des Eingabeaufforderungsfensters.
TREE	Zeigt die Ordnerstruktur eines Laufwerks oder Pfads grafisch an.
TYPE	Zeigt den Inhalt einer Textdatei an.
VER	Zeigt die Windows-Version an.
VERIFY	Legt fest, ob das ordnungsgemäße Schreiben von Dateien auf den Datenträger überprüft werden soll.
VOL	Zeigt die Volumebezeichnung und die Seriennummer des Datenträgers an.
XCOPY	Kopiert Dateien und Verzeichnisstrukturen.
WMIC	Zeigt WMI-Informationen in der interaktiven Befehlsshell an.

Weitere Informationen finden Sie in der Befehlszeilenreferenz der Onlinehilfe.

CMD – Befehle für‘s Netzwerk

Befehl	Parameter	Info
Ping		
	-a	Zeigt Namensauflösung an
	-n 1	Macht zum Host nur ein Ping (nur eine Echoanforderung)
	-w 5	Zeitlimit in Millisekunden für die einzelne Antwort
tracert (traceroute)		
	-d	Unterdrückt Namensauflösung
ipconfig		
	/all	Zeigt alle Einstellungen der lokalen Netzwerkeinstellungen aller Adapter an
	/flushdns	Lokaler DNS-Cache löschen
arp		
	-a	ARP-Tabelle anschauen
	-s	Neuen Eintrag setzen
	-d	Statischen Eintrag löschen
route print		Zeigt die Routingtabelle an
netstat		
	-r	Wie route print
telnet		
nslookup		Für Abfragen verschiedener DNS-Einstellungen

Tabelle 1: CMD Netzwerkbefehle

Tabelle noch im Aufbau

Glossar

Abkürzung	Ausgeschrieben	Beschreibung
CMD	Command	Windows Eingabeaufforderung
DHCP	Dynamic Host Configuration Protokoll	Zuweisung der Netzwerkkonfiguration an Clients durch einen Server
DNS	Domain Name System	Beantwortung von Anfragen zur Namensauflösung
FQDN	Fully Qualified Domain Name	Bsp. www.reto-burger.ch
FTP	File Transfer Protokoll	Ermöglicht den Datenaustausch über das Internet zwischen Client und Server
FTPS	File Transfer Protokoll Sesure	FTP über SSL oder FTP over TLS
HTTP	HyperText Transfer Protocol	Protokoll zur Datenübertragung an den Webbrowser
HTTPS	HyperText Transfer Protocol Secure	http mit SSL zur sicheren Datenübertragung des http
ICMP	Internet Control Message Protocol	Zum Austausch von Informations- und Fehlermeldungen. Bekannter Befehl: ping
OSI-Modell	Open Systems Interconnection Model	Referenzmodell für Netzwerkprotokolle als Schichtenarchitektur dargestellt mit 7 Schichten / 7 Layer
SFTP	SSH File Transfer Protocol	FTP über SSH
SMTP	Simple Mail Transfer Protokoll	Protokoll zum Austausch vom E-Mails in Computernetzwerken
SNMP	Simple Network Management Protocol	Protokoll zur Überwachung, Fernsteuerung und Fehler-benachrichtigung von Netzwerkkomponenten
SSL	Secure Sockets Layer	Ist die alte Bezeichnung von TLS
TLD	Top Level Domain	Bsp.: .com, .ch, .info, .net, .org. u.v.m.
TLS	Transport Layer Security	weitläufiger bekannt unter der Vorgängerbezeichnung Secure Sockets Layer (SSL). Bezeichnung TLS seit Version 3.0 hybrides Verschlüsselungs-protokoll zur sicheren Daten-übertragung im Internet
UNC	Uniform Naming Convention	(auch Universal Naming Convention) Bsp.: \\server1\freigabe oder \\10.10.0.1\austausch\dokus-free
URI	Uniform Resource Identifier	einheitlicher Bezeichner für Ressourcen Bsp.: http:
URL	Uniform Resource Locator	Bsp.: http://www.reto-burger.ch
WINS	Windows Internet Naming Service	Eine Umsetzung des Netzwerkprotokolls NetBIOS over TCP/IP durch Microsoft

Tabelle 2: Glossar

Verzeichnisse / Index

Abbildungsverzeichnis

Tabellenverzeichnis

Index

Autor:

Reto Burger, Eidg. dipl. Informatik Ingenieur HTL / FH, dipl. Berufsfachschullehrer mit Jahrgang 1968 aus Sempach (Schweiz).

Reto Burger ist Prüfungsexperte, Validator für Abschlussprüfungen in der Zentralschweiz, Lehrer und Dozent an verschiedenen Technikerschulen, Gewerbe- und Berufsschulen und an Hochschulen in der deutschsprachigen Schweiz.

Weiter ist Burger für die Modulentwicklung verschiedener ÜK's Verantwortlich und auch als Autor tätig. Burger war mehrere Jahre in der Kurskommission des VFI's engagiert und ist selber Lehrmeister.
Er hat für I-CH bei mehreren Netzwerk- und Systemtechnik–Module im Aufbau aktiv mitgearbeitet und war verantwortlich für verschiedene Module für die Ausbildung des Berufes gesamtschweizerisch.

Autor von:

- IP-Rechnen kann so einfach sein!
- Netzwerkgrundlagen
- Regular Expressions
- Rund um das Backup

Rechte

Garantie

Alle in diesem Stoff enthaltenen Berechnungen, Daten und Fakten wurden nach bestem Wissen erstellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschliessen. Aus diesem Grund sind die im vorliegenden Stoff enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Referenzen

Im Internet

www.burger.swiss
www.reto-burger.ch
www.bsi.de
usw.

In Literatur

Danke
Wenn Ihnen die Dokumentation gefallen hat, dürfen Sie das gerne mit einem kleinen Betrag via PayPal zeigen.

Reto Burger, Eidg. dipl. Informatik Ingenieur HTL / FH,
dipl. Berufsfachschullehrer mit Jahrgang 1968 aus Sempach (Schweiz).