

Intro to Scripting

September 6, 2023 8:41 AM

GUI - uses the CLI underneath (but harder to automate)

Scripts are text documents full of commands - quick and easy to develop, run slower than compiled programs

PowerShell help is useful

- 4 types of PowerShell
- PowerShell
- PowerShell (x86)
- PowerShell ISE
- PowerShell ISE (x86)

Always use the 64 bit version

Run as administrator

Help command searches for help topics (ex: "get-help *netadapter" gives you results that have netadapter in it)

Get-Command shows you all commands

Get-help has the alias "help"

PowerShell uses a verb-noun format - meaning that most commands will be "Set-x" or "Get-x"

Ex. Get-EventLog

Get-Help (command-name)

Required parameters are displayed in **<these guys>** or just doesn't have any brackets

Optional parameters are displayed in **[square brackets]**

Get-Help -name <anything you need>

How to use a command:

Help :command-name:

What commands there are:

Help :general term:

Get-Command -noun *event* < shows everything that has *event*

★ Using help:

Help <command> -full

Mandatory parameter : <these>

Option parameter: [these]

Positional parameter: [this] <and this>

Parameter values:

-string

-integer numbers

-datetime
-float

Formatting and Filtering

September 15, 2023 9:08 AM

Connecting Commands:

Pipeline - "|"

- this is used to connect commands to each other
- the pipe allows the first command pass its output to the next commands input

Regular Expression:

★ the escape character for PowerShell is ""

- this will type out the exact data rather than the value of the output of a variable (while using write-out)

PowerShell Objects:

- when PowerShell commands are executed, they are held in memory and create a table

-**Objects:** the "table row" - represents a single process or single service

-**Property:** the "table column" - represents one piece of info about the object, like name or process ID

-**Method:** the "action" - related to a single object and makes that object **do something** - like stop a process or start a service

-**Collection:** the entire set of objects, or what we call the table

★ **Get-Member** or **gm** - looks at each object and constructs a list of their properties and methods.

- this is used for finding more about what you can do with the command

ex. "get-process | get-member"

★ **Get-Help** or **help**

- used for finding out how to use the command

PowerShell Filtering

- used to modify the output of your initial command

Sort-Object - used for sorting an object in ascending or descending order

★ - Get-Process | Sort-Object -property VM -descending

Select-Object - used for selecting properties of the object you want

★ - Get-Process | Select-Object -property Name, ID, VM

- to find the properties that are available: "<command> | Get-Member"

Where-Object - used for narrowing in on what you want to be displayed

★ - Get-Service | Where-Object -filter {\$_.Status -eq 'Running'}

Filtering-Left - trying to narrow your search as fast as possible by doing the largest filters to the left-most part of the command

- this saves processing and network power

Filtering (comparison) - PowerShell uses these comparison operators while using Where-Object

-eq - equality

-ne - not equal

-ge - greater than or equal

-le - less than or equal

-gt - greater than

-lt - less than

-and

-like - uses wildcard characters (? for single character, * for any) - "where -filter {\$_.name -like '?e*'"

-match - finds things that have those characters in it - "where -filter {\$_.name -match 'search'"

-split - add onto the end of a string to make it cast as an array where the new variable is created at the space

(\$computerNames -split " ")

\$_. - this is used before a PowerShell variable

- ex. "\$_.Status"

★ - **placeholder used for piping in multiple inputs**

ex.

PC1

PC2 **\$_.** - each PC will be input into the **\$_.** command

PC3

Formatting:

Format-Table (ft) - this is how PowerShell displays by default

- -AutoSize
- -Property
- -groupBy
- -Wrap

Format-List (fl) - used to display more info that doesn't fit horizontally

Format-Wide (fw) - displays a wide list, can use "-columns" to give more columns ("-columns 5")

these formatting commands can be used in conjunction with:

- Out-Host
- Out-File
- Out-Printer

★ **Out-GridView** - gives it to you in a nice GUI

Always **FORMAT right** - this means that you will

PowerShell Scripts Best Practices

October 4, 2023 8:14 AM

".ps1" extension for PowerShell
\\test.ps1 to run it (have to follow file path)
run by pressing F5

★ [CmdletBinding()]
param (
[Parameter(Mandatory=\$True)] - this makes the parameter ask for a value if there is none
[string]\$computername, - this will ask the user for the value of \$computername
[ValidateSet(2,3)] - this only allows 2 or 3 as the input for \$drivetype
[int]\$drivetype = 3
) - called a parameter block (parameterizing)

documentation is good - use <# text here #> to do a text block

input and output:

Read-Host "type something" - this writes "type something" to the user and waits for a response | **results are in a string**
if you want to make a dialogue box:

[void][System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic') - this initiates the box

[Microsoft.VisualBasic.Interaction]::Inputbox('Enter computer name','Computer name dialog box','text input')

Write-Host "colours!" -foreground yellow -background black - used for displaying output | -fore and -back are not needed

Write-Warning - gives a warning - has "WARNING:" text

Write-Verbose - gives extra info - has "VERBOSE:" text

Write-Debug - gives debug info - has "DEBUG:" text

Write-Error - gives error message

\$var = "server-r2" - this is how to make a variable

\${My Variable} - this makes a variable that has a space in it (needs curly brackets)

to find what type a variable is:

"Server-R2" | GM or "server-r2".GetType()

single quotes are taken as literal text

double quotes allow variables to read what they contain

backtick skips 1 character

\$var = 'what does \$computer contain?'

what does \$computer contain?

\$var = "what does \$computer contain?"

what does Server-R2 contain?

\$var = "`\$computername contains \$computername"

\$computername contains server-r2

how to turn a string into an int:

[int]\$number = read-host "enter a number"

\$number.GetType()

TypeName: System.Int32

declaring variable types: **best practice for if your variable is going to be a specific type of type**

[int] - integer

[single] - float with 1 decimal places

[double] - float with 2 decimal places
[string] - string of characters
[char] - exactly 1 character
[xml] - xml document
[adsi]

removing variable:
Remove-Variable

declaring multiple objects in a single variable:
\$computers = 'SERVER-R2','SERVER1','localhost'
\$computers
SERVER-R2
SERVER1
localhost

^ this is an array

we can access specific elements of the array by using square brackets - **remember it starts at 0, you can use -1 for last, and -2 as second last**
ex.

\$computers[1]
SERVER1

using \$_ to iterate:

\$computers | ForEach-Object { \$_.ToLower() }
server-r2
client1
localhost

\$computers | ForEach-Object { \$_.ToUpper() }
SERVER-R2
CLIENT1
LOCALHOST

scopes:

- the shell itself is the top-level scope called "global scope"
- a script is called a "script scope"
- the script scope is considered a "child" of the global scope

★ **main PS rule for scope:**

trying to access a scope element

- PS checks to see if it exists within the current scope
- if it doesn't it checks the current scope's parent
- keeps going up until it gets to global scope

for ex.

in scope.ps1
\$x

in a normal PS window

C:\scope.ps1
(output is nothing)

\$x = 4
C:\scope.ps1
(output is:) 4 **this is because PS is going to the global scope (of the normal PS window)**

however:

in scope.ps1
\$x = 10
\$x

normal PS window

C:\scope.ps1

(output is:) 10

\$x = 4

C:\scope.ps1

(output still:) 10 **this is because PS is going to the script scope, NOT the global scope**

when making functions:

you might run into name issues with PS functions having the same name

- prefix the function name with initials > Get-QPNetinfo

```
Function Get-QPNetinfo {           - makes the function  
code here  
}
```

Get-QPNetinfo - calls the function

you can pull another script into the one you're using:

.C:\commands.ps1 - this pulls commands.ps1 into the scope you are using

. .C:\commands.ps1 - this opens the commands.ps1 as its own scope

Loops & Conditional Statements

October 13, 2023 9:12 AM

in an "if" statement:

- the first thing evaluated is the part in (normal brackets)
- if the value is true then the script is run

difference between foreach and foreach-object

- foreach loads everything into ram immediately | faster but uses more ram

```
foreach ($<item> in $<collection>){  
<statement>  
}
```

- foreach-object does everything one by one | slower but uses less ram

```
foreach-object {$_.<variable name>}
```

```
$condition = $true  
if ($condition){  
<code here>  
}
```

^ this will run

```
$condition = $false  
if ($condition){  
<code here>  
}
```

^ this will not

★ Switch statement:

```
$day = 3  
Switch ($day){  
0{$result = 'Sunday'}  
1{$result = 'Monday'}  
2{$result = 'Tuesday'}  
3{$result = 'Wednesday'}  
... etc.  
default{'unknown'}  
}
```

you can also replace the integers with strings

★ Arrays:

```
$roles = @('DHCP','ADDS','SQL','WebServer')
```

For Loops:

```
for (<init>; <condition>; <repeat>) {
```


<code>

}

ex.

```
for($count = 0; $count -le 5; $++){
```

```
$count
```

```
}
```

^ this loop will go 5 times, since it is checking to see if the count is less than or equal to 5

While is a pre-test loop - meaning it checks if the condition is true before running

Do-While and Do-Until are post-test loops - will run once before checking condition to see if its true

"break" will eject you from the current block

Automation

October 18, 2023 8:49 AM

automation includes some form of:

- remote-control
- multitasking
- monitoring

WS-MAN (web services for management)

- operates over HTTP/HTTPS
- uses background service "Windows Remote Management" (WinRM)
- WinRM is installed with PS2 or higher
- **started by default** on windows server 2008 or higher
- also installed on most workstations but **disabled by default**

to convert the output from remote commands to be transferred over the network you need to serialize it

- to do this we convert the information to an XML format
- then when it is at its destination we must deserialize it back into an object
- ^ this means that the information is not real time

requirements:

- windows powershell v2 or higher
- ideally they are both domain joined
- allow WMI rule through firewall (in and out)

to enable WinRM for remoting in on clients:

"Enable-PSRemoting"

could also do it through a GPO (computer config>admin template> windows components)

Enter-PSSession - less resource intensive RDP | passes through your user credentials

PowerShell uses remoting in 2 ways:

one-to-one:

- "Enter-PSSession -computername Server01" - remotes into server01
- "Exit-PSSession" (or close PS window)

one-to-many:

- "Invoke-Command" - sends a command to multiple remote computers **at the same time**
- by default, PS can talk to up to 32 computers at once; if you ask for more it will queue computers up
- if you want to increase the limit for remote PCs, you can use "-throttleLimit"

Invoke-Command -command {<code here>}

- computerName (Get-Content <path to list of computers.txt>)

Get-Content outputs a string which is what -computerName needs;

Get-ADComputer returns objects which -computerName won't know understand

WMI is the older version of CMI, it works on more devices and is designed for older servers.

Get-WMIObject -list | shows all classes that you can use

Get-WMIObject -Namespace root\CIMv2 -list | where name -like "*dis*"

CMI doesn't have a list, but it has:

get-Cimclass -namespace root\CIMv2

if you want to use credentials for remote computers with CIM: (??)

Get-CimInstance via Invoke-Command

invoke-command -scriptblock {Get-CimInstance -ClassName win32_logicaldisk} -computername server-1 -credential DOMAIN\Administrator

Managing Background Jobs:

help *job

- background jobs are commands that run separately in the background

★ - **background jobs don't give you errors which is no good**

- have to decide if you want to run a job in the background **before** you run the command

synchronously / real-time: run a command and you wait for the command to complete

asynchronously / background: run a job in the background and continue to use the shell to do other jobs

★ **TEST YOUR COMMAND BEFORE YOU SET IT TO A BACKGROUND JOB**

how to create a job:

Start-Job -ScriptBlock {<code>}

or

Start-Job -Command ()

-name | allows you to specify the name of the job

-credential | asks for credentials to run command

Get-Job - all this does is shows you the status of the jobs that have run or are running

Get-Job | Format-List - this shows you more information about the job itself (not the data)

★ the property "HasMoreData" being true means you can view the data

remove-job - gets rid of the job in get-job

stop-job - terminates the job

wait-job - forces shell to stop and wait until job is completed, then allows the shell to continue

Receive-Job [<name/ID>] - this is how to view the data that the command has obtained

- if you want to view the results more than one you have to use **-Keep (you have to type it every time you want to view it)**, otherwise its viewable only once (stored in cache)

To make a command act in the background:

add a **-AsJob** parameter

ex: **Get-WMIObject win32_operatingsystem -computerName (get-content computernames.txt) -asjob**

or: **invoke-command -computername localhost -scriptblock {get-process} -asjob -jobname testing1**

Scheduling tasks:

New-JobTrigger - used to schedule a job by creating a trigger that activates a task

New-ScheduledTaskOption - use to set options for the job

Register-ScheduledJob - used to register the job with task scheduler; creates job definition in task scheduler's XML format - creates folder hierarchy to hold results

Register-ScheduledJob -Name DailyProcList -ScriptBlock {Get-Process} -Trigger (New-JobTrigger -Daily -At 2am) -ScheduledJobOption (New-ScheduledJobOption -WakeToRun -RunElevated)

Project #1 Planning

October 20, 2023

10:02 AM

menu system — MATT
with submenus

AD OU

- create
- remove

AD Groups

- create
- remove

} MATT

User Management

- Manage new or current user
 - create new user
 - delete existing user
 - move existing user
- View locked accounts (and when)
- View user logon times
- View password crack attempts (2 P/W fails)

R&D — MATT

Create VMs — 10 VMs

all connected to isolated private network

ICMP protocol setup — enabled through firewall

Single user creates/modifies their own VMs

VMs internet connected

System Backup — QUANG
Must be as a background

- Schedule backup process
- Execute Backups immediately
- Manage all backup jobs
 - View jobs
 - restore
 - delete

run backup now

- 1 name of backup policy: (\$Policy)
- 2 What do you want to backup: (\$Filespec)
- 3 where do you want to backup to: (\$BackupLocation)
- 4 Save & run command — check if var exist
- 5 exit without running

run backup later

- 1 name of backup policy: (\$Policy)
- 2 what do you want to backup: (\$Filespec)
- 3 where do you want to backup to: (\$BackupLocation)
- 4 when do you want to do the backup? (daily?, weekly?, @what time?)
- 5 Save & run command — check if vars exist
- 6 exit without running

} Pseudocode

Manage backups

- 1 View backup jobs — this can populate dropdown box?
- 2 restore to a backup — dropdown(?) of backup options
- 3 delete a backup job — this, and ask if they're sure
- 4 go back

restore to a backup

backup options here

Cancel

restore

→ prompt user "are you sure?"
it will go back to how it was before

go back to manage

delete a backup

backup options here

cancel

!!delete!!

→ prompt user "are you sure you want
to delete your backup?"

go back to
manage

TASK 2

automate at least 2 additional jobs

garfie wants us to do Azure PowerShell

Configuring cisco switches / routers

↳ create a cutsheet for cisco configs

domain wide task manager

Cisco Config goal: — QUANG

it's assumed IP/SSH has been configured

- ask user for SSH credentials
↳ IP, username, password

- ask for secret password

- show general config → Spanning tree, IP, IP routing, what type of device (L2, L3, Router), OSPF
- show a list of ports on the device → switchport mode (access, trunk, no), port security, Portfast, BPDU guard | IP address, description
- show VLANs → where they're assigned & how (access, trunk), native VLAN, ip for VLAN if IP routing

being able to configure

general config

- ↳ spanning tree
- ospf
- ACL
- logging synchronous

ports

- ↳ switchport mode / encapsulation
- assign VLANs / ranges?
- IP address
- description
- port security
- Portfast
- BPDU guard

VLANs

- ↳ create/remove from database (maybe in general config?)

VLANs

- ↳ create/remove from database (maybe in general config?)
- ↳ IPs on VLAN 1
- IP helper addresses
- auto disable VLAN 1
- no shut vlans

how it'll actually look

- check boxes for what you want to configure > next
 - ↳ stage one by one (general, VLAN, port)
 - ↳ more menu to configure from there (for now just a switch)

Quiz notes more complete than they were before

Wednesday, November 15, 2023 8:13 AM

13 questions

- 6 points** - Make sure you can do things as a background job / asynchronously
- *Do this so we can continue using our computer without having to wait for the job to complete*
- Make sure you can view the job specifications
- Check if the job is running / failed / done
 - Name, ID, Parent / Child
- Get-Job**
- *shows the jobs that have been run in this session of PowerShell*
- Receive-Job**
- *gives you the data that the job has pulled (needs job ID or name)*
 - *make sure to use -Keep each time you receive it or the data will be erased (as its stored in cache)*
- Start-Job**
- *executes a command in the background*
 - *used with either -command() or -ScriptBlock{}*
- AsJob**
- *method used on some commands such as Get-WmiObject*
 - *"help * -parameter -AsJob"*
- Remove-Job**
- *this will remove a job from the Get-Job list*
- Wait-Job**
- *Runs a script as a job, but only allows you to continue when the job is done*

Scheduled Jobs

- New-ScheduledTaskTrigger**
- *This command lets you set when the schedule will run*
 - *-Weekly or -Daily lets you set how often*
 - *-At is required for when it runs (what time)*
- New-ScheduledTaskAction**
- *This command lets you set what will happen when triggered*
 - *-Execute lets you run an application (for running script "PowerShell")*
 - *-Argument let you select what arguments you want in the action (script would have commands here)*
- Register-ScheduledTask**
- *This is what actually sets the task as scheduled*
 - *-TaskName is where you set what the task will be called*
 - *-TaskPath is where you set where the task will be found (in task scheduler)*
 - *-Action this is where you put the variable that is assigned to New-ScheduledTaskAction*
 - *-Trigger this is where you put the variable that is assigned to New-ScheduledTaskTrigger*

- 12 points** - two scripts, one can call another (dot sourcing, calling function)
- . <script path>**
- *This calls the script, but does not import its variables and functions*
- . . <script path>**
- *Dot sourcing essentially "imports" the called script's functions and variables into your current script scope*

Parameter Block - one script needs to have this

WMI = Windows Management Instrumentation | CIM = Common Information Model

GM / Help is still needed

- Need to be able to do a loop
- Until told to quit

- Break will quit a loop

```
WHILE LOOP
$var = 0 #declaring a variable before

While($var -le 5){ #start of the while loop | also condition for doing the loop
$var ++ #increases $var by 1
"$var = $var" #displays the value of $var
}
```

```
DO WHILE LOOP
$var = 0 #declaring a variable before

★ DO{ #start of do while loop
$var ++ #increases $var by 1
"$var = $var" #displays the value of $var

}while ($var -le 5) #condition that is checked for loop continuation
```

- What is the difference between synchronous vs asynchronous for a job
- *synchronous locks you out of the terminal until the job is complete*
 - *asynchronous runs the job in the background and lets you continue using the terminal*

- What is the difference between serialized / deserialized (invoke command thing)
- *serialization is what you do to do data that is going over the network (like when using invoke command)*
- It converts the information into an XML file (which you can't do anything with)*
- *deserialization is what you do when the information comes back to the device running the command*
- Converts the information back into an object, (THIS INFORMATION IS NOT REAL TIME)*

Can you use Garfield's favourite, the help system???

- (?) find cmdlet that can view your disk objects
- *Get-Disk*

- (?) find a cmdlet that can change your IP address
- *Get-NetIPAddress*

```
Function Function-Name{ # this is how to make a function
<code>
}
```

```
★ [CmdletBinding()]
param (
[Parameter(Mandatory=$True)]
[string]$computername,
[ValidateSet(2,3)]
[int]$drivetype = 3
)
```

- this makes the parameter ask for a value if there is none
- this will ask the user for the value of \$computername
- this only allows 2 or 3 as the input for \$drivetype

- called a parameter block (parameterizing)

Bash - Loops and Conditions

Wednesday, November 22, 2023

8:21 AM

General syntax for if:

```
if command-list1
then
    command-list2
fi
```

Or

```
if command-list1; then command-list2; fi
```

OR

Command1 && command2 - does the same as an if and then

Command1 || command2 - runs command2 if command1 is a success

Failed code uses an error code of 1-255, successful code returns 0

0 = True, 1-255 = False

Exit *n* - will set the return value

"test" and "["/>" - are essentially the same

- Evaluates conditions then exits with a status based on result

Test option filename - this is the syntax

ex. test -t 1; echo \$? - 0 means no redirection of stdout

OR

[-r "\$FILE"]; echo \$? - 0 means file exists and readable

NAME=John

Test "\$NAME" = Bill (IDK WHAT THIS DOES???)

Testing integer Expressions:

Same as powershell

-eq, -ne, -lt, -le, -gt, -ge

! = not

-a = and

-o = or

Else statement:

```
if command-list1
Then
    Command-list2
Else
    Command-list3
Fi
```


Elif statement:

If command-list1

Then

 Command-list2

Elif command-list3

Then

 Command-list4

Else

 Command-list5

Fi

Indent for readability

Case Statement: basically a switch

Case word in

Pattern1) command-list

;;

Pattern2) command-list

;;

Pattern) command-list

;;

Esac

FOR loop:

for NAME in LIST

Do

 Command-list

Done

Or

For NAME

Do

 Command-list

Done

WHILE loop:

While command-list1

Do

- command-list2 is run as long as command-list1 has an exit code of 0 (no errors)

 Command-list2

Done

UNTIL loop:

Until command-list1

Do

- command-list2 is run as long as command-list1 has an exit code NOT EQUAL to 0

 Command-list2

Done

IO redirection

For NUMBER in 1 2 3 4

Do

 Echo \$NUMBER

#CASE EXAMPLE

#!/bin/bash

echo -n "Enter the name of a country: "

read COUNTRY

echo -n "The official language of \$COUNTRY is "

case \$COUNTRY in

 Lithuania)

 echo -n "Lithuanian"

;;

 Romania | Moldova)

 echo -n "Romanian"

;;

 Italy | "San Marino" | Switzerland | "Vatican City")

 echo -n "Italian"

;;

*)

 echo -n "unknown"

;;

esac

Done > myfile.txt

While read LINE

Do

Echo \$LINE

Done < myfile.txt

BREAK / CONTINUE

Break - kill loop

Continue - restart loop

Bash Intro

Wednesday, November 15, 2023 8:37 AM

Gedit / Nano / VIM for text editing

- need to set the permissions to execute on the bash file
- **sudo chmod u+x <script_name>**

Common to add **.sh** to the end of a script - not required

Implicit execution:

`./myscript.sh` - this will run the script if it has execute permissions

`Otherscript.sh` - same thing

Explicit execution:

`Bash myscript.sh`

`Bash -x myscript.sh`

Hashspling (**#!**)

- Ensures the correct shell type is used to interpret the script
- First two characters must be **"#!"** then the absolute path to script

★ **#!/bin/bash** - *THIS IS THE FIRST THING IN THE SCRIPT OF A BASH SCRIPT*

Comments are still **#<comment>**

Debugging:

Running the script with **"-x"** or **"-v"** will give an output

Setting **"set -x"** and or **"set -v"** will enable debugging mode

-v is **Verbose** - crazy stuff

-x is **Xtrace** - prints commands before executing

Special Characters - also called metacharacters

~ = home directory

/ = root directory OR hides all metacharacters

"" = hides most metacharacters (except **\$**)

" = hides all metacharacters

In bash **all variables are string**. (except an array)

Variables need to start with a letter or underscore - must also be in ALL CAPS

"set" - reveals variables and their values

Variable usage in Bash

`MYVAR="Test Subject"`

- **must have no space between the variable and the value**

`Echo $MYVAR`

- the output of this would be **"Test Subject"**

`Echo -n "Enter Value: " ; read MYVAR`

- asks the user to enter a value and assigns it to MYVAR

`Echo $MYVAR`

- the output of this would be **whatever the user just put in**

`MYVAR=123`

- this is still in **string** form

`Let RESULT=$MYVAR+5`

- **"let"** makes the variable act as an **integer** rather than a string

Echo \$RESULT - output of this would be "128"

Function Creation: - functions use "return" rather than "exit"

```
function <Name>{  
local SUM - this makes the scope local rather than global (like if you didn't put local)  
let SUM=$1+$2  
echo $SUM  
}
```

BASH Automation

Wednesday, November 22, 2023

8:58 AM

Automated Tasks utilities:

Cron

At

Batch

Cron - found at /etc/crontab

Used for task scheduling - uses a combination of time and day for frequency

Syntax in /etc/crontab:

Minute hour day month dayofweek command

* - all valid

"-" - range of values

/ - step value

- comments



Crontab -e - for modification (don't change the file directly)

Crontab -l - to view tasks

/etc/cron.allow - if it exists it acts as an implicit deny, ignores deny - with these users allowed

/etc/cron.deny - if ONLY this exists, it denies these users

Only put the username on a single line for allow or deny

"at" is used to schedule a one-time task at a specific time

-l : view list of scheduled jobs (reg users can see their own)

-c : view system environment at scheduling time

-d : delete a job

-f : run scheduled job from shell script1

-r <ID-Number> : deletes the job

To exit you need to press **[CTRL+D]**

Ex.

At 5:00pm FRI

at> <command here>

[CTRL+D]

"batch" is used to schedule a one-time task for whenever the systems load average drops below 0.8

Same as the at command

Performance Monitoring

iostat - CPU monitoring

vmstat - Network monitoring

free - RAM monitoring

df - File System monitoring

Service monitoring:

"ps ax"

Systemctl show -p ActiveState

Email notices:

Mail -s "subject line" username < datafile.txt

Command | mail -s "subject line" username