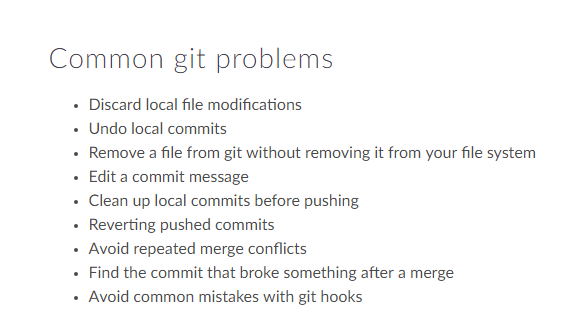
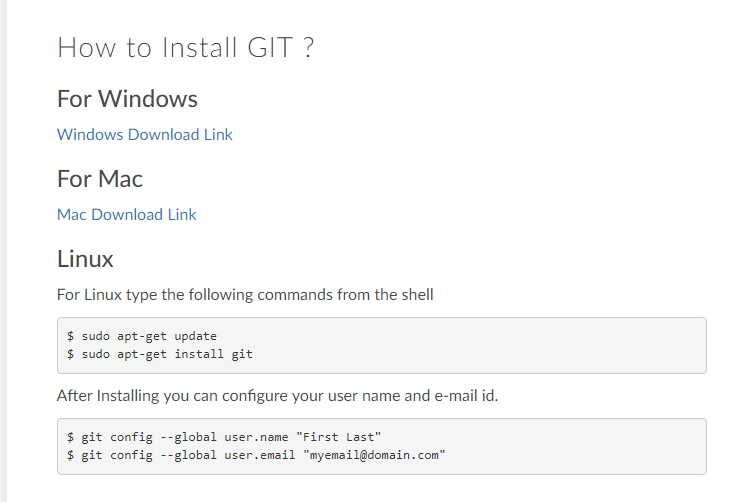
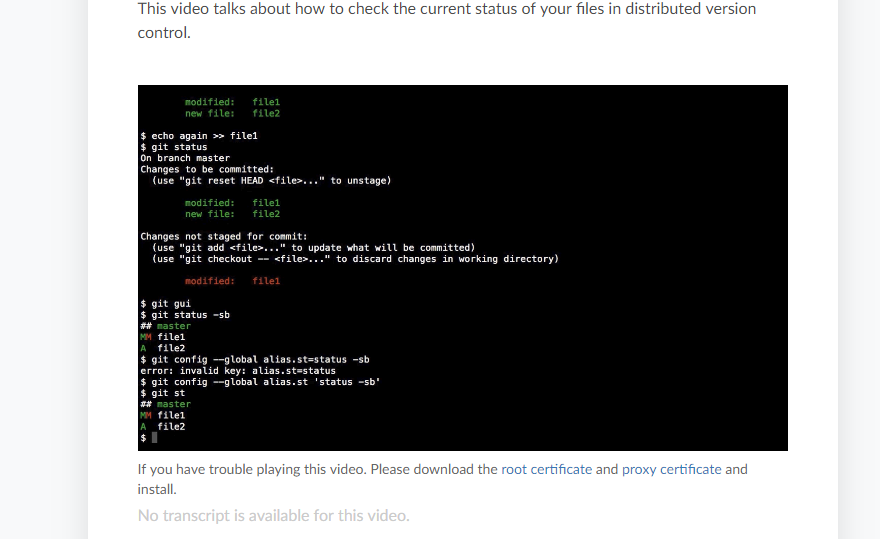
GIT

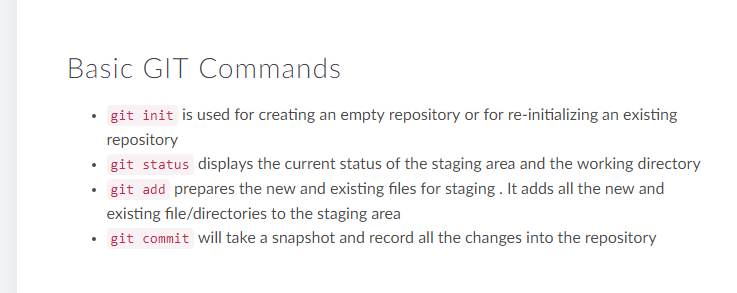
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

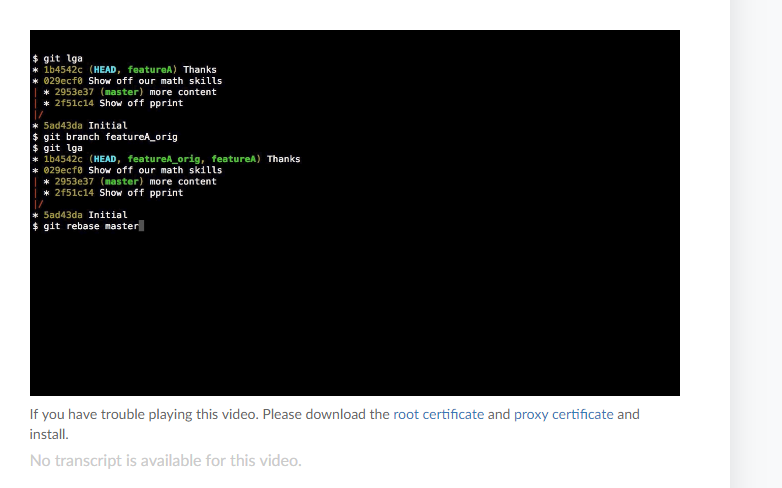


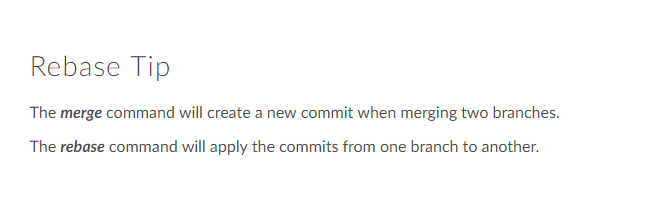


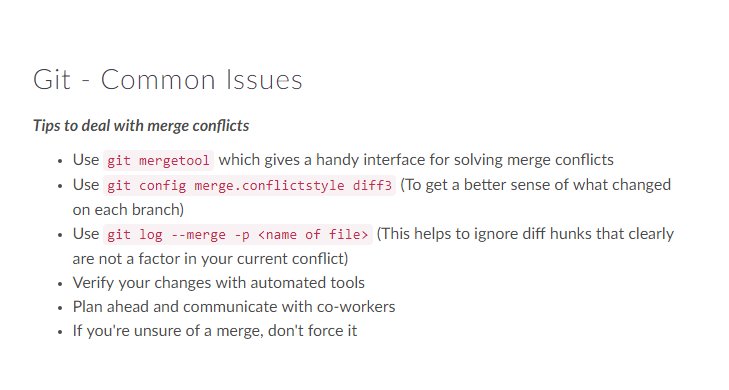




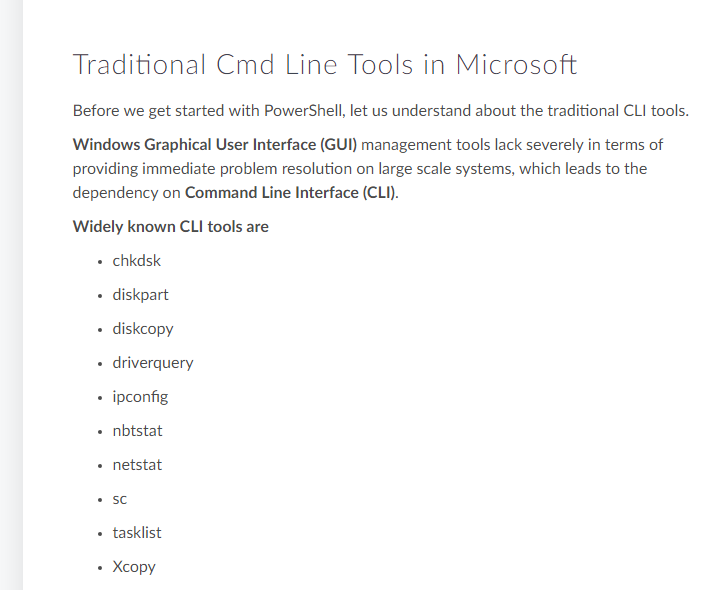








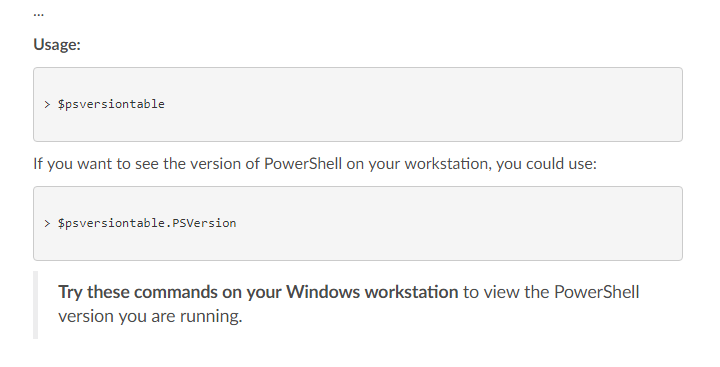
Power Shell











PSVersionTable is a built-in variable that contains versions of PowerShell and related components on your workstation. It includes information such as:

* CLRVersion
* SerializationVersion
* BuildVersion
* PSCompatibleVersions

...

**Usage:**

> $psversiontable

If you want to see the version of PowerShell on your workstation, you could use:

> $psversiontable.PSVersion

**Try these commands on your Windows workstation** to view the PowerShell version you are running.

Open Source Version of PowerShell

The latest version of PowerShell 6.0 is available on [GitHub.com](https://github.com/) as Open Source Project. This supports both Windows and Non-Windows Platforms including:

* Windows 10/Server 2016
* Windows 8.1/Server 2012 R2
* Ubuntu 16.04
* Ubuntu 14.04
* CentOS 7
* openSUSE 42.1
* Oracle Linux 7
* Red Hat Enterprise Linux (RHEL) 7 and
* Arch Linux
* macOS 10.11
* Docker

PowerShell Source for Linux

* For a **Ubuntu system**, source files can be downloaded and installed as a Debian package (DEB) file from GitHub.
* For **CentOS, openSUSE, Oracle Linux and RHEL**, it can be downloaded and deployed as an RPM Package Manager files with an RPM extension.
* **Arch Linux versions** are accessible from the Arch User Repository, rather than GitHub.
* For **Container**, a Docker version of PowerShell can be leveraged.

PowerShell commands allow manipulation of objects and are used to complete the standard automation tasks.

**Command Format**

They follow a standard **VERB-NOUN** naming format.

* The first word in the cmdlet represents **what needs to be done** such as get, set, start, and remove.
* The second word identifies **which object it should act upon** such as service, and process.

For example, the command **get-service** lists all System Services.

Shell Function Commands

* Functions are a **Named grouping of Windows PowerShell commands and expressions.**
* Functions can have parameters and switches.
* They can take .NET objects as input and return .NET objects as output, just like cmdlets.
* They can be reused during that session.

**Command to view all PowerShell built-in Functions:**

>get-command -Type Function

**Try it on your Workstation and [Katacoda Playground](https://katacoda.com/courses/ubuntu/playground) to view the Output**

function get-timeresult {

param ([int]$a,[int]$b)

$c=$a\*$b

Write-Output $c

}

**To Run this function:**

get-timeresult -a 6 -b 8

**The Output will be 48**

Try this simple function on your PowerShell Console.

Working with Alias Commands

* To find all the Alias Commands

>Get-Command -Type Alias

* To find an equivalent Powershell cmdlet for "dir"

>get-alias dir

* To discover all the aliases defined for the "'get-childitem"" cmdlet

>get-alias –definition get-childitem

**Try these commands on your PowerShell console to see them working!**

##### Most Commonly used Cmdlets

In PowerShell **get-command** and **get-Help** cmdlets are frequently used.

* **get-command** - lists all command in the PowerShell.
* **get-help** - displays information about various PowerShell commands including cmdlets, scripts, functions, etc.

PowerShell has thousands of commands, and we will be covering just the popular ones in this course.

Use PowerShell help to learn about the command and its usage.

Using get-command

**get-command** can be used in two ways

* To list all the command based on "Verb" like, get, and set.

get-command get-\*

* To list all commands based on "Noun" like Process, and Services.

**Example**:

get-command \*-service

##### PowerShell Help Command



**Learning PowerShell commands** is very easy when you learn how to use PowerShell help!

**For example, if you want to manage a Process, what is the command for it and how to use it?**

**Step 1:** First, you will need to find all Cmdlet's related to Process.

- get-command \*-Process

On viewing the listed Cmdlet's, we can filter the option to **get-process**.

**Step 2:** Find out how to use this Cmdlet using PowerShell help.

get-help get-process

This would display a whole lot of information about the Cmdlet including Syntax, Aliases, etc.

Types of Help in PowerShell

PowerShell provides four types of Help,

* **Full** help for get-process

>get-help get-Process -full

* **Detailed** help for get-process

>get-help get-Process -Detailed

* **Examples** of get-process

>get-help get-Process -Examples

* **Online** help for get-process

>get-help get-Process -Online

**Try these commands on your Windows, Ubuntu playgrounds and analyze the results.**

##### View all PS help commands

**Command to view all "help" commands**

get-command \*-help

**Output will be**

* get-help
* update-help
* save-help

**To run update-help and save-help, Administrative Privilege is required**.

##### Format Commands

Format Commands in PowerShell are used to Change the Output View based on the specified properties.

**Related Cmdlets:**

* Format-Custom
* Format-List
* Format-SecureBootUEFI
* Format-Table
* Format-Wide

**Usage:**

>Get-Process | Format-Table ProcessName, id

>Get-Process | Format-list ProcessName, id

**Try these commands on your Windows and Ubuntu PowerShell consoles to view the Output.**

PowerShell Module Cmdlets

* Get-Module
* Import-Module
* New-Module
* Remove-Module

**Steps to Import Active Directory Module in Windows 7:**

* Download and Install a suitable RSAT version for Windows 7.
* Run the following command "Import-Module ActiveDirectory".

Administrative Privilege is required for performing this.

##### Variables Example

In the command below, **$path** stores the path of "C:\Windows\System32". **get-ChildItem**command displays the list of Item present in the given path.

>$path = "C:\Windows\System32"

>Get-ChildItem -Path $path

**Try these commands on your PowerShell Console.**

##### Read-Host Cmdlet

The **Read-Host** cmdlet interactively prompts the user to enter a value for the **Variable**.

For example, the command below prompts the user to enter a name.

>$Name=Read-Host "Please enter user name"

**Try this command on your Windows workstation PowerShell Consoles.**

If we need to mask the data entered at the prompt, the **-assecurestring parameter** can be used.

**Example to provide the Password securely:**

>$Password=Read-Host -assecurestring "Please enter your password"

On using the -assecurestring parameter, when the user starts entering the password, it will masked with "\*".

**Try this command on your Windows workstation PowerShell Consoles.**

Variable

**PowerShell** contains **Pre-Defined** variables also

* **Get-variable** - To list out all the variables
* **Remove-variable** - To remove the variable which is created

Try these commands on your Windows Workstation PowerShell console.

##### Write-Host Cmdlet

The **Write-Host cmdlet** can be used to write\displays messages on the Windows PowerShell console.

**Example**

>$numberofcommands=(get-command).count

>Write-Host "$numberofcommands available in this System"

**foregroundcolor** and **backgroundcolor** are two optional parameters that can be used with **Write-Host**, to specify a different text and text background colors.

**Example**:

Write-Host "This is red text on a yellow background" -foregroundcolor red -backgroundcolor yellow

Try these commands on your Windows Workstation PowerShell Consoles.

##### Write-Host

Separator parameter in Write-Host specifies the string that can be used to separate the objects.

**Example**

>Write-Host (5,10,15,20) -Separator ", +5= "

This command will list the given numbers in separate lines.

Try this command on your Windows workstation PowerShell Consoles.

Pipeline in PowerShell

Pipes (|) in PowerShell allow passing the output of one cmdlet as an input to another cmdlet. This helps in performing a series of actions on selected objects easily using a single command.

**Now let's look at an example to kill a specific process.**

* Open a notepad on your system. Now, execute the command below. This should list details of this new process.

>get-process -Name notepad

* Now let us kill/stop this process.

>get-process -Name notepad | Stop-Process

The above command will select the process notepad and stop it.

**Try these commands on your workstation for further clarity.**

##### Operators in PowerShell

**Operators are used to perform mathematical and filtering task**

* **-ne** (not equal to)
* **-lt** (less than)
* **-le** (less than or equal to)
* **-gt** (greater than)
* **-ge** (greater than or equal to)
* **-like** (like—a wildcard comparison)
* **-notlike** (not like—a wildcard comparison)
* **-contains** (contains the specified value)
* **-notcontains** (doesn't contain the specified value)

Now, let's look at some commands to see how to use Pipes with other Cmdlets to perform certain common operations.

**Where-Object** cmdlet can be used to filter Objects in the list based on a given condition.

**Example:** To filter the process which is consuming more than 20 CPU(S).

Get-Process | Where-Object -FilterScript{$\_.cpu –gt 20}

To see this command working, try it out on Windows and Ubuntu Powershell Console.

Following command can be used to get a list of services, find the ones that are in stopped status and start them.

>get-service bits,winrm | where-object {$\_.status -eq 'stopped'} | start-service

Try this on your Windows Workstation PowerShell console.

The Cmdlet **ForEach-Object** is used to perform an action on each instance of an object. Syntactically, **ForEach-Object** is very similar to Where-Object.

**Example:** To display the free space of each Partition in a Hard Disk (in Bytes)

> Get-WMIObject Win32\_LogicalDisk | ForEach-Object {$\_.FreeSpace}

##### To Select-Object

**The Select-Object cmdlet displays all the properties of particular Object**.

**Example**:

>Get-Process -Name chrome | Select-Object -Property \*

Try this command on your PowerShell console to view the output.

find the hard disk free space in **Gigabytes**:

> Get-WMIObject Win32\_LogicalDisk | ForEach-Object {$\_.FreeSpace/1024/1024/1024}

Try this on your Windows Workstation PowerShell Console.

##### Pipeline - To Find Specific Properties

The **Select-Object** cmdlet chooses specified properties of a set of objects or an object. The **Property parameter** can be used to mention what needs to be displayed.

**Example:** To display Process name, Id and CPU seconds.

>Get-Process | Select-Object -Property ProcessName, Id, Cpu

To view the output, try this on your Windows and Ubuntu PowerShell Consoles.

##### Pipeline - To Sort Objects

Sort-Object cmdlet sorts the set of Objects according to a particular Property.

**Example:** To Sort the Process based on Memory.

Get-Process | Sort-Object -Property ProcessName

To view the output, try this on your Windows and Ubuntu PowerShell Consoles.

##### Integrated Scripting Environment

**Integrated Scripting Environment (ISE)** is regarded as a host application for Windows PowerShell and it is **GUI**.

ISE can be used to run commands and write, test, and debug scripts from a single interface.

**Features**

* Multiline editing
* IntelliSense
* Visual debugging
* Tab completion
* Syntax coloring
* Selective execution
* Context-sensitive help

Simple scripts can be tested in any PowerShell console.

ISE interface can be accessed the same way as Native PowerShell console.

**Different ways to start PowerShell**:

* From the RUN command, type: PowerShell ISE
* From cmd.exe, type: PowerShell ISE
* Using **GUI: Start > All Programs > Accessories > Windows PowerShell > Windows PowerShell ISE (Windows7 and Later versions)**

##### Script Using IF - Elseif

Here is a simple application of If-Elseif block to find the Greater Number.

> if(20 -gt 10)

{

Write-Host "20 is Greater"

}elseif(10 -gt 20)

{

Write-Host "10 is Greater"

}

Output is **20 is Greater**. Try it out on Windows PowerShell ISE Console.

##### Script Using Do-While

Like in any programming language, **Do-While** loop in PowerShell performs an action as long as a given condition is satisfied. Here is a script that illustrates the usage of Do While loop.

> do{

$N1=$N1+1

Write-Host "The Current Variable is $N1"

}While($N1 -lt 5)

Try it out on your Windows PowerShell Console to view the Output.

##### Script Using For-Each

In this example, **For-Each** loop counts the total number of users given in the "$name" variable and list the user names.

> $names = "Amy","Bob","Candice","Dick","Eunice","Frank"

> $count = 0

> ForEach ($singlename in $names) {

$count += 1

Write-Host "$singlename"

}Write-Host "The total number of names is $count."

Try it out on your Windows **PowerShell ISE Console** to view the Output.

**To run PowerShell script files PowerShell Execution Policy must be set as "Remotesigned"**.

**Commands to set Execution Policy**

Set-ExecutionPolicy RemoteSigned -Force

**PowerShell Script File**

* It contains a bunch of PowerShell commands. These files should be saved with ".PS1" extension.
* PS script files can be run on both Native PowerShell or ISE.

To run this command, Administrative Privilege is required.

PS Script File creation

**To find the greater Number**:

* Type following commands in the notepad

$a = Read-host "A"

$b = Read-host "B"

if ($a -gt $b)

{

write-host "A"

} elseif ($b -gt $a)

{

write-host "B"

}

* Save as "\*.PS1" file in a current working directory and execute

**If the file name is *script.ps1* then type as**

> .\script.ps1

* PowerShell console prompts for A and B
* If **A: 100 and B: 200** then
* Output will be **B**

Error Handling in PowerShell

It is required to understand the types of errors that can occur during the execution PowerShell commands and Scripts.

**Types of Error**

* **Terminating Error** - A serious error during execution that halts the command or script completely.

**Examples** - Non-existent of cmdlets, syntax errors which prevent a cmdlet from running, or other fatal error.

* **Non-Terminating Error** - A non-serious error that allows execution to continue despite the failure.

**Example** - Operational errors such file not found and permissions problems.

Error Action Preference can be used for Non-Terminating errors.

##### Error Action Preference

For non-terminating errors, there is the choice to tell PowerShell how to manage these situations.

**Error Action Preference allows us to specify the desired behavior for a non-terminating error**

**Choices for Error Action Preference**

* **SilentlyContinue** – Here, Error messages are suppressed, and then execution goes on.
* **Stop** – Forces execution to stop, behaving like a terminating error.
* **Continue** - The default option. Errors will display, and execution will continue.
* **Inquire** – Urge the user for input to check whether to proceed or not.
* **Ignore** – (new in v3) – the error is ignored and not logged to the error stream.

##### EAP

**Example 1** - For Script

**To Set the preference at the script scope to Stop, place the following command at the top of the script file**

$ErrorActionPreference = "Stop"

**Example 2** - For Cmdlets

**To set the preference at the cmdlet level to Inquire, error action switch can be added**

get-childitem "D:\Folder1" -ErrorAction "Inquire"

Try this command on your Windows Workstation to view an effect of "Error Action".