```
C:\Users\<YourUsername>\AppData\Local\Google\Chrome\User Data\Default\Login Data

C:\Users\<YourUsername>\AppData\Local\Microsoft\Edge\User Data\Default\Login Data

C:\Users\John\AppData\Roaming\Mozilla\Firefox\Profiles\x1xyz.default-release\
                              Logins.json, key4.db
```

To check if directory exists or not

```
using System;
using System.IO;

class Program
{
   static void Main()
   {
      string folderPath = @"C:\Users\YourUsername\Documents\TestFolder";

      if (Directory.Exists(folderPath))
      {
         Console.WriteLine("Folder exists.");
      }
      else
      {
         Console.WriteLine("Folder does not exist.");
      }
   }
}
```

To search for particular files in the directory

```
using System;
using System.IO;

class Program
```

```
{
    static void Main()
    {
        string filePath = @"C:\MyFolder\example.txt";

        if (File.Exists(filePath))
        {
            Console.WriteLine("File exists.");
        }
        else
        {
            Console.WriteLine("File does not exist.");
        }
    }
}
```

To get all files from directory

```
string folderPath = @"C:\MyFolder";
string searchPattern = "example*.txt";  // Can also use "*.log", "data?.csv", etc.

string[] files = Directory.GetFiles(folderPath, searchPattern);

if (files.Length > 0)
{
    Console.WriteLine("Matching files found:");
    foreach (var file in files)
    {
        Console.WriteLine(file);
    }
}
else
```

```
{
    Console.WriteLine("No matching files found.");
}
```

Run powershell command in c#

```
using System;
using System.Diagnostics;

class Program
{
    static void Main()
    {
        // The PowerShell command you want to run
        string command = "Get-LocalUser | Select-Object Name, Enabled, PasswordRequired";

        // Set up the process info
        ProcessStartInfo psi = new ProcessStartInfo();
        psi.FileName = "powershell.exe";
        psi.Arguments = $"-Command \"{command}\"";
        psi.RedirectStandardOutput = true;
        psi.RedirectStandardError = true;
        psi.UseShellExecute = false;
        psi.CreateNoWindow = true;

        // Start the process
        Process process = Process.Start(psi);

        // Read the output
        string output = process.StandardOutput.ReadToEnd();
        string error = process.StandardError.ReadToEnd();
        process.WaitForExit();

        // Display the result
        Console.WriteLine("Output:\n" + output);
        if (!string.IsNullOrWhiteSpace(error))
            Console.WriteLine("Error:\n" + error);
    }
}
```

2nd method require installation

```
using System;
using System.Management.Automation;
using System.Collections.ObjectModel;

class Program
{
    static void Main()
    {
        // Create a PowerShell instance
        using (PowerShell ps = PowerShell.Create())
        {
            // Add your PowerShell command
            ps.AddScript("Get-LocalUser | Select-Object Name, Enabled, PasswordRequired");

            // Execute the command
            Collection<PSObject> results = ps.Invoke();

            // Display the results
            foreach (var result in results)
            {
Console.WriteLine(result.Members["Name"].Value);
Console.WriteLine(result.Members["PasswordRequired"].Value);            }

            // Display any errors
            if (ps.Streams.Error.Count > 0)
            {
                Console.WriteLine("Errors:");
                foreach (var error in ps.Streams.Error)
                {
                    Console.WriteLine(error.ToString());
                }
            }
        }
    }
}
```

Get the main ip address

```
using System;
using System.Linq;
using System.Net;
using System.Net.NetworkInformation;
using System.Net.Sockets;

class Program
{
    static void Main()
    {
        string mainIp = GetLocalIPv4();
        Console.WriteLine("Main IP Address: " + mainIp);
    }
    static string GetLocalIPv4()
    {
        foreach (NetworkInterface ni in NetworkInterface.GetAllNetworkInterfaces())
        {
            if (ni.OperationalStatus != OperationalStatus.Up)
                continue;

            if (ni.NetworkInterfaceType == NetworkInterfaceType.Loopback ||
                ni.Description.ToLower().Contains("virtual") ||
                ni.Description.ToLower().Contains("vmware") ||
                ni.Description.ToLower().Contains("virtualbox") ||
                ni.Description.ToLower().Contains("tunnel"))
                continue;

            IPInterfaceProperties ipProps = ni.GetIPProperties();

            foreach (UnicastIPAddressInformation ip in ipProps.UnicastAddresses)
            {
                if (ip.Address.AddressFamily == AddressFamily.InterNetwork)
                {
                    return ip.Address.ToString();
                }
            }
        }

        return "Not found";
    }
}
```

(Get-NetIPAddress -AddressFamily IPv4 -InterfaceAlias "Wi-Fi").IPAddress

```
Get-NetIPAddress -AddressFamily IPv4 | Where-Object { $_.PrefixOrigin -eq "Dhcp" -and
$_.IPAddress -notlike "169.*" } | Select-Object -ExpandProperty IPAddress
```

```
ipconfig | findstr /i "IPv4"
```

```
(Get-NetAdapter -Name "Wi-Fi").MacAddress
```

Get Ip and mac

```
using System;
using System.Linq;
using System.Net.NetworkInformation;
using System.Net.Sockets;

class Program
{
    static void Main()
    {
        string mainIP = GetMainIPv4Address();
        Console.WriteLine("Main IPv4 Address: " + mainIP);
    }

    static string GetMainIPv4Address()
    {
        foreach (NetworkInterface ni in NetworkInterface.GetAllNetworkInterfaces())
        {
            if (ni.OperationalStatus != OperationalStatus.Up)
                continue;

            if (!ni.Supports(NetworkInterfaceComponent.IPv4))
                continue;
            string desc = ni.Description.ToLower();
            if (desc.Contains("virtual") || desc.Contains("vmware") || desc.Contains("loopback") ||
desc.Contains("tunnel") || desc.Contains("pseudo"))
                continue;

            var ipProps = ni.GetIPProperties();
            var ip = ipProps.UnicastAddresses
                .FirstOrDefault(x => x.Address.AddressFamily == AddressFamily.InterNetwork);

            if (ip != null)
                return ip.Address.ToString();
        }

        return "Not Found";
    }
}
```

……

```
using System;
using System.Net.NetworkInformation;

class Program
{
    static void Main()
    {
        string mac = GetMainMacAddress();
        Console.WriteLine("Main MAC Address: " + mac);
    }

    static string GetMainMacAddress()
    {
        foreach (NetworkInterface ni in NetworkInterface.GetAllNetworkInterfaces())
        {
            if (ni.OperationalStatus != OperationalStatus.Up)
                continue;

            string desc = ni.Description.ToLower();
            if (desc.Contains("virtual") || desc.Contains("vmware") || desc.Contains("loopback") ||
desc.Contains("tunnel") || desc.Contains("pseudo"))
                continue;

            var macBytes = ni.GetPhysicalAddress().GetAddressBytes();
            if (macBytes.Length == 0) continue;

            return string.Join("-", macBytes.Select(b => b.ToString("X2")));
        }

        return "Not Found";
    }
}
```