Last write time

```
using System;
using System.Runtime.InteropServices;

class Program
{
    const int KEY_READ = 0x20019;

    [StructLayout(LayoutKind.Sequential)]
    struct FILETIME
    {
        public uint dwLowDateTime;
        public uint dwHighDateTime;
    }

    [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    static extern int RegOpenKeyEx(
        UIntPtr hKey,
        string subKey,
        uint options,
        int samDesired,
        out IntPtr phkResult);

    [DllImport("advapi32.dll", SetLastError = true)]
    static extern int RegCloseKey(IntPtr hKey);

    [DllImport("advapi32.dll", SetLastError = true, CharSet = CharSet.Unicode)]
    static extern int RegQueryInfoKey(
        IntPtr hKey,
        System.Text.StringBuilder lpClass,
        ref uint lpcClass,
        IntPtr lpReserved,
        out uint lpcSubKeys,
        out uint lpcMaxSubKeyLen,
        out uint lpcMaxClassLen,
        out uint lpcValues,
        out uint lpcMaxValueNameLen,
        out uint lpcMaxValueLen,
        out uint lpcbSecurityDescriptor,
        out FILETIME lpftLastWriteTime);

    static void Main()
    {
        string keyPath =
@"SYSTEM\CurrentControlSet\Enum\USBSTOR\Disk&Ven_hp&Prod_v210w&Rev_1100";
        UIntPtr hKeyRoot = (UIntPtr)0x80000002u; // HKEY_LOCAL_MACHINE

        IntPtr hKey;
        int result = RegOpenKeyEx(hKeyRoot, keyPath, 0, KEY_READ, out hKey);
        if (result != 0)
        {
            Console.WriteLine("Failed to open key. Error code: " + result);
            return;
        }

        try
        {
```

```
            FILETIME ft;
            uint classLen = 0;
            uint subKeys, maxSubKeyLen, maxClassLen, values, maxValueNameLen,
maxValueLen, securityDescriptor;

            result = RegQueryInfoKey(hKey, null, ref classLen, IntPtr.Zero,
                out subKeys, out maxSubKeyLen, out maxClassLen,
                out values, out maxValueNameLen, out maxValueLen,
                out securityDescriptor, out ft);

            if (result == 0)
            {
                long fileTime = ((long)ft.dwHighDateTime << 32) + ft.dwLowDateTime;
                DateTime lastWrite =
DateTime.FromFileTimeUtc(fileTime).ToLocalTime();
                Console.WriteLine("Last write time: " + lastWrite);
            }
            else
            {
                Console.WriteLine("Failed to query key. Error code: " + result);
            }
        }
        finally
        {
            RegCloseKey(hKey);
        }
    }
}
```

Last write time of second level

```
using System;
using System.Runtime.InteropServices;
using Microsoft.Win32;

class Program
{
    const int KEY_READ = 0x20019;

    [StructLayout(LayoutKind.Sequential)]
    struct FILETIME
    {
        public uint dwLowDateTime;
        public uint dwHighDateTime;
    }

    [DllImport("advapi32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    static extern int RegOpenKeyEx(
        UIntPtr hKey,
        string subKey,
        uint options,
        int samDesired,
        out IntPtr phkResult);

    [DllImport("advapi32.dll", SetLastError = true)]
    static extern int RegCloseKey(IntPtr hKey);
```

```
    [DllImport("advapi32.dll", SetLastError = true, CharSet = CharSet.Unicode)]
    static extern int RegQueryInfoKey(
        IntPtr hKey,
        System.Text.StringBuilder lpClass,
        ref uint lpcClass,
        IntPtr lpReserved,
        out uint lpcSubKeys,
        out uint lpcMaxSubKeyLen,
        out uint lpcMaxClassLen,
        out uint lpcValues,
        out uint lpcMaxValueNameLen,
        out uint lpcMaxValueLen,
        out uint lpcbSecurityDescriptor,
        out FILETIME lpftLastWriteTime);

    static DateTime? GetKeyLastWriteTime(string fullPath)
    {
        UIntPtr hKeyRoot = (UIntPtr)0x80000002u; // HKEY_LOCAL_MACHINE
        IntPtr hKey;
        int result = RegOpenKeyEx(hKeyRoot, fullPath, 0, KEY_READ, out hKey);

        if (result != 0)
            return null;

        try
        {
            FILETIME ft;
            uint classLen = 0;
            uint subKeys, maxSubKeyLen, maxClassLen, values, maxValueNameLen,
maxValueLen, securityDescriptor;

            result = RegQueryInfoKey(hKey, null, ref classLen, IntPtr.Zero,
                out subKeys, out maxSubKeyLen, out maxClassLen,
                out values, out maxValueNameLen, out maxValueLen,
                out securityDescriptor, out ft);

            if (result == 0)
            {
                long fileTime = ((long)ft.dwHighDateTime << 32) + ft.dwLowDateTime;
                return DateTime.FromFileTimeUtc(fileTime).ToLocalTime();
            }
        }
        finally
        {
            RegCloseKey(hKey);
        }

        return null;
    }

    static void Main()
    {
        string rootPath = @"SYSTEM\CurrentControlSet\Enum\USBSTOR";

        using (RegistryKey baseKey = Registry.LocalMachine.OpenSubKey(rootPath))
        {
            if (baseKey == null)
```

```
            {
                Console.WriteLine("Failed to open base USBSTOR key.");
                return;
            }

            foreach (string firstLevelSubKeyName in baseKey.GetSubKeyNames())
            {
                using (RegistryKey firstLevelKey =
baseKey.OpenSubKey(firstLevelSubKeyName))
                {
                    if (firstLevelKey == null)
                        continue;

                    foreach (string secondLevelSubKeyName in
firstLevelKey.GetSubKeyNames())
                    {
                        string fullSubKeyPath =
$@"{rootPath}\{firstLevelSubKeyName}\{secondLevelSubKeyName}";
                        DateTime? lastWrite = GetKeyLastWriteTime(fullSubKeyPath);

                        if (lastWrite.HasValue)
                        {

Console.WriteLine($"{firstLevelSubKeyName}\\{secondLevelSubKeyName} =>
{lastWrite.Value}");
                        }
                        else
                        {

Console.WriteLine($"{firstLevelSubKeyName}\\{secondLevelSubKeyName} => Error
fetching time");
                        }
                    }
                }
            }
        }
    }
}
```

“

Get-SmbShare | Select-Object -ExpandProperty Name

Get-LocalGroupMember -Group "Administrators" | Select-Object -ExpandProperty Name

Cmdlets

🔥 **Major PowerShell Cmdlets & Usage for Users, Groups, and System Info**

1. Get-LocalGroupMember

List members of a local group (e.g., Administrators).

powershell

CopyEdit

Get-LocalGroupMember -Group "Administrators"

Filter just names:

powershell

CopyEdit

Get-LocalGroupMember -Group "Administrators" | Select-Object -ExpandProperty Name

---

2. Get-LocalUser

Get all local users.

powershell

CopyEdit

Get-LocalUser

Filter enabled users only:

powershell

CopyEdit

Get-LocalUser | Where-Object {$_.Enabled -eq $true}

---

3. Get-LocalGroup

List all local groups.

powershell

CopyEdit

Get-LocalGroup

---

4. Get-WmiObject (alias gwmi)

Query WMI for many system info, including users, processes, etc.

Example: List all user accounts (local and domain)

powershell

CopyEdit

Get-WmiObject -Class Win32_UserAccount

Filter only local users:

powershell

CopyEdit

Get-WmiObject -Class Win32_UserAccount | Where-Object {$_.LocalAccount -eq $true}

---

5. Get-Service

List services and filter by status or name:

powershell

CopyEdit

Get-Service

Running services only:

powershell

CopyEdit

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

---

6. Get-Process

List all running processes.

powershell

CopyEdit

Get-Process

Filter by process name:

powershell

CopyEdit

Get-Process -Name "notepad"

---

7. Get-EventLog

Query event logs, for example, to get the last 10 system events:

powershell

CopyEdit

Get-EventLog -LogName System -Newest 10

Filter by EntryType (Error, Warning, Information):

powershell

CopyEdit

Get-EventLog -LogName System -EntryType Error -Newest 5

---

8. Get-SmbShare

List all SMB network shares on the local machine:

powershell

CopyEdit

Get-SmbShare

Get just share names:

powershell

CopyEdit

Get-SmbShare | Select-Object -ExpandProperty Name

---

9. Get-ChildItem (alias ls or dir)

List files/folders.

Example: List all .txt files in C:\Temp:

powershell

CopyEdit

Get-ChildItem C:\Temp -Filter *.txt

Recursive:

powershell

CopyEdit

Get-ChildItem C:\Temp -Recurse -Filter *.txt

---

10. Select-Object

Select specific properties or expand them.

Example:

powershell

CopyEdit

Get-Process | Select-Object -Property Id, ProcessName, CPU

Expand property (if it's an object):

powershell

CopyEdit

Get-LocalGroupMember -Group "Administrators" | Select-Object -ExpandProperty Name

---

11. Where-Object

Filter based on a condition.

Example:

powershell

CopyEdit

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

---

12. Sort-Object

Sort output.

Example:

powershell

CopyEdit

Get-Process | Sort-Object CPU -Descending

---

13. Format-Table / Format-List

Format output neatly.

Example:

powershell

CopyEdit

Get-Process | Format-Table Id, ProcessName, CPU -AutoSize

📚 **How to Query & Filter: Summary**

Cmdlet	**Filtering/Selecting Example**
Get-LocalUser	Get-LocalUser | Where-Object {$_.Enabled -eq $true}
Get-LocalGroupMember	Get-LocalGroupMember -Group "Administrators" | Select-Object -ExpandProperty Name
Get-Service	Get-Service | Where-Object {$_.Status -eq 'Running'}
Get-Process	Get-Process -Name "chrome"
Get-SmbShare	Get-SmbShare | Select-Object -ExpandProperty Name
Get-WmiObject	Get-WmiObject -Class Win32_UserAccount | Where-Object {$_.LocalAccount}

🔥 **How to Discover Available Cmdlets & Functions**

- Get all cmdlets:

powershell

CopyEdit

Get-Command

- Find cmdlets related to users:

powershell

CopyEdit

Get-Command *user*

- Find cmdlets related to groups:

powershell

CopyEdit

Get-Command *group*

Bonus: Get Help for Cmdlets

Get detailed info and examples for any cmdlet:

powershell

CopyEdit

Get-Help Get-LocalUser -Full

Update help files (if needed):

powershell

CopyEdit

Update-Help

---

1. Select-Object

- Use to select specific properties from objects.
- Can select one or many properties.
- Does **not** extract property values, but keeps them as objects (unless you expand).

Example: Select only Name and Status properties from services

powershell

CopyEdit

Get-Service | Select-Object Name, Status

---

2. Select-Object -ExpandProperty

- Extracts (expands) the value of a property.
- Returns just the raw value, not wrapped in an object.

- Useful when you want a list of strings or numbers instead of objects.

Example: Get only the Name strings of local group members

powershell

CopyEdit

Get-LocalGroupMember -Group "Administrators" | Select-Object -ExpandProperty Name

This outputs plain usernames (strings), not objects.

---

3. Where-Object

- Filter objects based on conditions.
- Use script block {} with property checks.

Example: Get services that are currently running

powershell

CopyEdit

Get-Service | Where-Object { $_.Status -eq "Running" }

Example: Get local users enabled only

powershell

CopyEdit

Get-LocalUser | Where-Object { $_.Enabled }

---

4. ForEach-Object

- Process each object in the pipeline individually.
- Useful for running commands or expressions on each item.

Example: Output each service name with a custom message

powershell

CopyEdit

Get-Service | ForEach-Object { "Service: $($_.Name)" }

---

5. Sort-Object

- Sorts objects by property.

Example: Sort processes by CPU usage descending

powershell

CopyEdit

```
Get-Process | Sort-Object CPU -Descending
```

---

6. Format-Table / Format-List

- Formats output as table or list.
- Use for better readability in console.

Example: Show services as a table with specific columns

powershell

CopyEdit

```
Get-Service | Format-Table Name, Status -AutoSize
```

Combining Techniques in Pipeline

You can combine filtering techniques to get exactly what you want.

Example: Get names of running services sorted alphabetically

powershell

CopyEdit

```
Get-Service | Where-Object { $_.Status -eq "Running" } |
  Select-Object -ExpandProperty Name |
  Sort-Object
```

Summary Table

Cmdlet	Purpose	Example
Select-Object	Pick specific properties	`Get-Process
Select-Object -ExpandProperty	Extract raw property values	`Get-LocalUser
Where-Object	Filter objects by condition	`Get-Service
ForEach-Object	Process each object individually	`Get-Process

Cmdlet	Purpose	Example
Sort-Object	Sort output by property	`Get-Process
Format-Table / Format-List	Nicely format output	`Get-Service

Bonus: Using aliases

- Select-Object → select
- Where-Object → where
- ForEach-Object → foreach
- Sort-Object → sort

So this is equivalent:

powershell

CopyEdit

Get-Service | where {$_.Status -eq "Running"} | select -ExpandProperty Name

---

If you want, I can provide **real-life examples** for specific scenarios or how to use these in your C# code calling PowerShell. Just ask!

You said:

are there more common properties like skip and all

ChatGPT said:

Great question! In PowerShell, Select-Object indeed has some additional common parameters that help you **control how many objects you get** or **skip objects**, plus others. These can be super useful for pagination or limiting output.

Common Select-Object Parameters

Parameter	Description	Example
-First <n>	Selects the first **n** objects	`Get-Process
-Last <n>	Selects the last **n** objects	`Get-Process
-Skip <n>	Skips the first **n** objects	`Get-Process

Parameter	Description	Example
-Unique	Selects only unique objects (no duplicates)	`Get-Content file.txt
-ExpandProperty	Expands the value of a property as explained	`Get-LocalUser

---

Using -Skip and -First Together (Pagination Example)

If you want to get items **6 to 10**, skip first 5, then take next 5:

powershell

CopyEdit

Get-Process | Select-Object -Skip 5 -First 5

What about "-All"?

- Select-Object **does not have** a -All parameter.
- By default, it selects all objects if you don't specify -First or -Skip.

Other Useful Cmdlets with Similar Parameters

Cmdlet	Similar Parameters	Description
Get-ChildItem	-Recurse, -Depth	Control how deep recursive listing goes
Get-Content	-TotalCount (like -First), -Tail	Read only first or last N lines from file
Sort-Object	-Descending, -Unique	Sort descending or get unique sorted values

---

Quick Example Putting it All Together

Get names of running services, skip first 3, then take next 5:

powershell

CopyEdit

Get-Service | Where-Object {$_.Status -eq "Running"} | Select-Object -Skip 3 -First 5 | Select-Object -ExpandProperty Name

---

Summary

- Select-Object commonly uses -First, -Last, -Skip, and -ExpandProperty.
- No -All parameter because default is "all".
- Combine these to control output size and shape.

If you want me to cover **other cmdlets' common parameters** or give detailed examples, just ask!