Script Best Practices

Description

Tips and tricks for running Filesets with scripts

Don't put passwords in scripts

If a script needs a username or password, we don't want those to be stored in the script locally on workstations for security reasons.

Example: password in command

```
somecommand -u username -p "PASSSWORD_HERE"
```

So, we can replace those values with variables from the fileset

Windows: PowerShell

```
somecommand -u $args[0] -p $args[1]
```

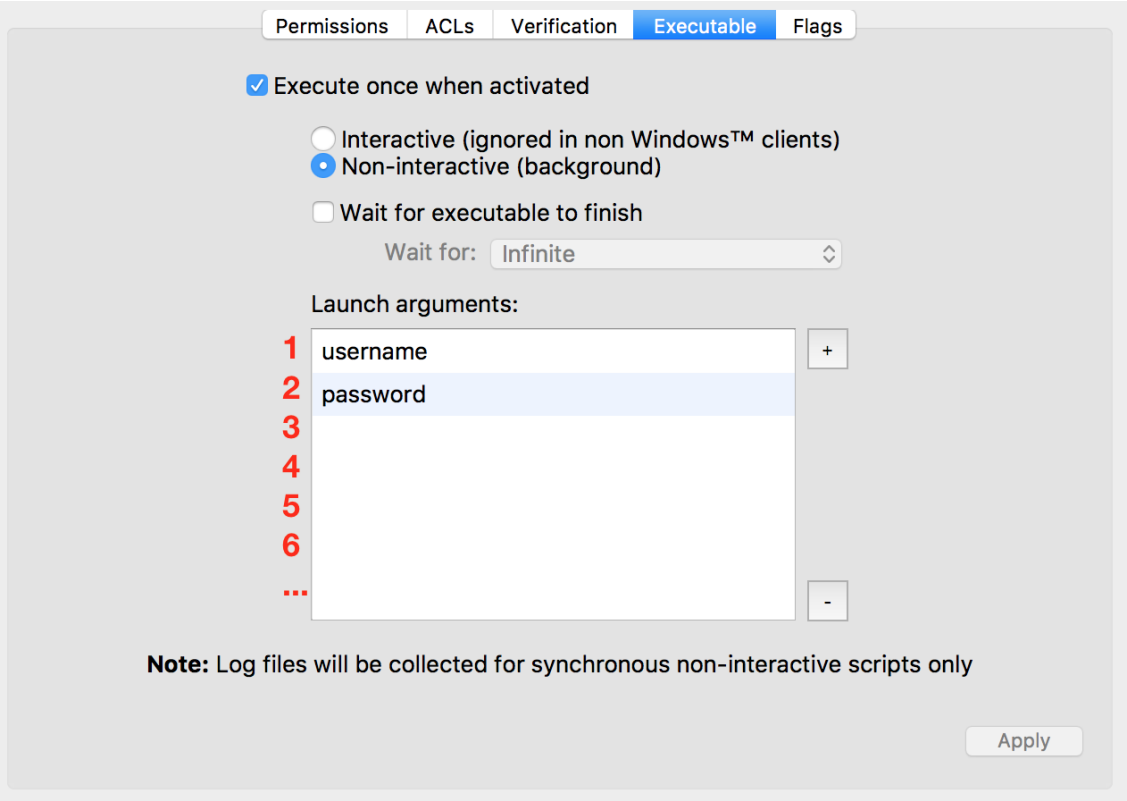
Windows: All other script types i.e. bat, python, perl

```
somecommand -u %1 -p %2
```

macOS / Linux

```
somecommand -u $1 -p $2
```

These will be pulled from the executable arguments



Keep Requirements Scripts Small

Requirements scripts are pulled from a fileset and sent before the remainder of the fileset.

It behaves this way because if a requirements script fails, there is no point in downloading and installing the remainder of the fileset.

> ⚠ A fileset whose requirements have failed will not even show up in the kiosk.

Where possible, avoid piping commands. This increases overhead on the scripts. If pipes are required, try to reduce the quantity of pipes. If nothing else, this makes the scripts easier to read.

```
$ time system_profiler SPHardwareDataType | grep "Model Identifier" | awk '{print $NF}'
MacBookPro11,4


$ system_profiler SPHardwareDataType | awk '/Model Identifier/ {print $NF}'
MacBookPro11,4
```

And other commands may achieve the same result more efficiently without the need to pipe.

```
$ time system_profiler SPHardwareDataType | grep "Model Identifier" | awk '{print $NF}'
MacBookPro11,4

real        0m0.192s
user        0m0.071s
sys        0m0.049s


$ time sysctl -n hw.model
MacBookPro11,4

real        0m0.004s
user        0m0.001s
sys        0m0.002s
```

Consider this for all scripts beyond just requirement scripts.

Leverage Dependancies and Scripts

If there is a script that several filesets will need, don't paste the same script into each one. Create an empty fileset with that script, and make the other filesets dependent upon it.

Log Script Output to the Client Log

Have a script that needs to write details steps to a log?

Want a quick status of the script.

Have a script write to the client log.

macOS / Linux

```
#!/bin/bash
exec 1>>/var/log/fwcld.log
exec 2>>/var/log/fwcld.log


... rest of script
```

You can then use Client Monitor, and pull and view the log as things are happening

Testing Scripts

Scripts run by FileWave are run by root or System. As such, scripts should be tested using the same user context to prevent erroneous results. Many commands will yield the same result regardless, but this cannot be relied upon.

Windows

E.g. Run the following on a Windows 10 Professional system locally through Powershell as either user or 'Run As Admin' will see the following result:

```
 (Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion").EditionId
Professional
```

However, as a Custom Field running the same script, the result is surprisingly different:

```
 (Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion").EditionId
Enterprise
```

This is because Windows is providing a different answer based upon either the user running the script or may provide different responses based on 32-bit or 64-bit.

Take a look at Getting a CMD prompt as SYSTEM in Windows Vista and Windows Server 2008 for details about running scripts as System. Note, that by default, this will start an executable as 64-bit, for native 64-bit OS. However, the above example is because the FileWave fwcld process is calling the 32-bit version of PowerShell.

> **PsTools**
>
> This relies on downloading and installing, onto the test machine, PsTools.

To mimic this experience, consider the guide to starting the CMD. When launching an executable, like PowerShell, the 32-bit version would need to be referenced. For example:

```
>PSEXEC  -i -s -d C:\Windows\SysWOW64\windowsPowerShell\v1.0\powershell.exe
```

For CMD, the equivalent would be:

```
>PSEXEC -i -s -d %windir%\SysWoW64\cmd.exe
```

Similarly, when attempting to run some commands, it may be necessary to ensure Windows is using the correct version of a binary with the 'sysnative' redirect. An example would be bitlocker's 'manage-bde.exe'. To use this in a Fileset, try the following:

```
C:\Windows\sysnative\manage-bde.exe -status
```

If you have a requirement to run a particular command through the 64 bit version of Powershell this can be achieved as follows:

```
If ( [IntPtr]::Size * 8 -ne 64 )
{
    C:\Windows\SysNative\WindowsPowerShell\v1.0\PowerShell.exe -File $MyInvocation.MyCommand.Path
}
Else
{
    # Add code here
}
```

Example

Create a new admin account

Two Fileset Launch Arguments would be supplied. To add the user 'rstephens' with the password 'filewave'

1. rstephens
2. filewave

Parameters may be supplied, that can then be added to the execution of Powershell from within the script:

```
Param (
    [string]$MyUsername = $args[0],
    [string]$MyPassword = $args[1]
)


If ( [IntPtr]::Size * 8 -ne 64 )
{
    C:\Windows\SysNative\WindowsPowerShell\v1.0\PowerShell.exe -File $MyInvocation.MyCommand.Path -MyUsername
$MyUsername -MyPassword $MyPassword
}
Else
{
    (New-LocalUser -AccountNeverExpires:$true -Password ( ConvertTo-SecureString -AsPlainText -Force
$MyPassword) -Name $MyUsername | Add-LocalGroupMember -Group administrators)
}
```

macOS

On macOS, running commands as sudo is not necessarily the same as actually becoming root.

E.g. Run the following commands to evaluate the local variable $HOME, once using sudo and once as root.

```
$ whoami
auser
$ sudo echo $HOME
/Users/auser
$ sudo su -
$ whoami
root
$ echo $HOME
/var/root
```

It is common to see plist files edited with the 'defaults' command. However this command is unique when it comes to ownership and permissions of files. The 'defaults' command will both take ownership and change permissions of files when used to write to plist files:

```
$ whoami
root
$ ls -al /tmp/example_plist.plist
-rw-r--r--  1 rstephens  staff  66 Feb 28 10:03 /tmp/example_plist.plist
$ defaults write /tmp/example_plist Label example_plist
$ ls -al /tmp/example_plist.plist
-rw-------  1 root  wheel  66 Feb 28 10:05 /tmp/example_plist.plist
```

As such, ensure to add a repair to scripts to reset permissions and ownership after the command has been used or consider using the following command instead:

```
/usr/libexec/PlistBuddy
```

Other Script Recipes

- DeepFreeze (Knowledge Base)
- Execute as Console User (Knowledge Base)
- Firmware (macOS 10.6 - macOS 10.12) (Knowledge Base)
- Full macOS Model name to Inventory (Knowledge Base)
- Hide local admins (Knowledge Base)
- Remotely Enable Screen Sharing for a user (Knowledge Base)
- Remove Casper JSS Client Components (Knowledge Base)
- Remove mobileconfig signature (Knowledge Base)
- Script Best Practices (Knowledge Base)
- User Icon (Knowledge Base)
- Windows Firewall rules (Knowledge Base)