

Migrating from PC SAS® to SAS Grid on Linux

Susan Myers, Inga Allred, and M. Rita Thissen, RTI International

ABSTRACT

SAS® Grid is available for Windows or Unix, and it offers features and benefits that are not available under PC SAS. What challenges do programmers face when an organization makes the move from Windows-based PC SAS to Linux-based SAS Grid? To adapt to the new system, you must address differences between Windows and Linux as well as the differences between PC SAS and SAS Grid. To complete some batch operations, you also need to coordinate programs on both platforms as part of the same overall process. We present our experience and lessons learned regarding Windows versus Linux, the use of PuTTY versus MobaXterm terminal emulators, data management, program conversion, and the use of Windows command files that call Linux shell scripts. These combined operations may work well, but the learning curve can be steep.

INTRODUCTION

In 2016, after months of strategic planning, RTI undertook a migration from SAS software installed locally on Windows workstations (PC SAS) to an enterprise-wide installation of SAS Grid technology on a Linux platform (Wilson, Green, and Terminello 2017). It was a demanding transition, coordinated across more than 400 SAS users who lived in numerous U.S. states and several other countries. Some programmers had prior experience with Linux, but many did not. Likewise, some used stand-alone programs, but many operations required series of linked programs, with SAS among them, and many SAS products were in use. The entire migration had to be conducted in a way that minimized disruption to active operations. In short, the migration was complex in technical, coordination, productivity, and educational aspects.

This paper addresses issues that have a direct impact on the programmer. It is intended for data system programmers, SAS database managers, and especially process managers whose operations require interaction between SAS and external files or processes. During migration, RTI staff learned many new concepts and practices. We were hindered early in the transition by the lack of readily available information giving the user's perspective, although publications and presentations are now appearing that will help those who migrate in the future (Brinsfield 2017; Lopuski and Tan 2017). The learning curve can be quite steep and time-consuming, especially for programmers who are not well versed in network terms, distributed computing systems, Linux commands, and virtual environments. In this paper, we share fundamental tips from a programmer's perspective to aid others who may need to migrate from PC SAS to SAS Grid on Linux.

The first question for PC SAS users facing a move to Linux may be, What is SAS Grid? In brief, SAS Grid is a centralized installation of SAS software that uses a group of processing servers to distribute computing load and improve performance. Many programmers are familiar with terms like server farms, failover systems, load balancing, and shared storage. A grid has characteristics of all of these. Because processing is centralized, no local installation or software management is required. Patches, updates, and licensing are easier to control, and any user with access to the grid network can work at any hour outside of scheduled downtime. To understand changes to the way programs are written for and submitted to run on SAS Grid, it is important that you grasp the concept of grid execution.

Companies choose to adopt SAS Grid for a variety of reasons. Because of its technical and business benefits to customers, SAS software's organizational direction is heading toward grid architecture. The centralization of systems, support, and licensing offers savings and simplified management processes compared to overseeing a large and varied number of individual workstation installations. The grid is scalable: More process servers can be added without changing architecture. SAS Grid offers parallel processing, greater in-memory options, and faster execution times for large jobs—features that prove invaluable when working with very large datasets. Remote access to the Grid environment allows users to be located anywhere in the physical world, as long as they are able to connect to the network space, and their work does not suffer data transfer slowdowns because storage and processing take place centrally.

Some advantages of SAS Grid derive simply from centralization, demonstrating once again the opposing pulls of local control versus consolidation. Desktop adding machines were replaced by mainframes and time-shared access. Mainframes and terminals, in turn, were replaced by desktop computers. In the latest round of digital evolution, software-as-a-service, cloud computing, server farms, and SAS Grid offer the benefits of scale. The trade-off is accepting some loss of simplicity and personalization that come with managing your own desktop environment.

On the grid, SAS processes jobs somewhat differently than on a desktop workstation. Figure 1 (<https://support.sas.com/rnd/scalability/grid/gridarch.html>) shows the basic concepts of SAS Grid computing architecture. The grid manager analyzes programs that are submitted from a grid client and distributes them to the node with the most resources available, as is often done in network load balancing. Each processing node uses its own temporary workspace but shares access to the file servers that hold permanent copies of programs, data files, and objects, such as format catalogs. Programmers must learn where actions take place, or they will be puzzled when files are written to surprising locations.

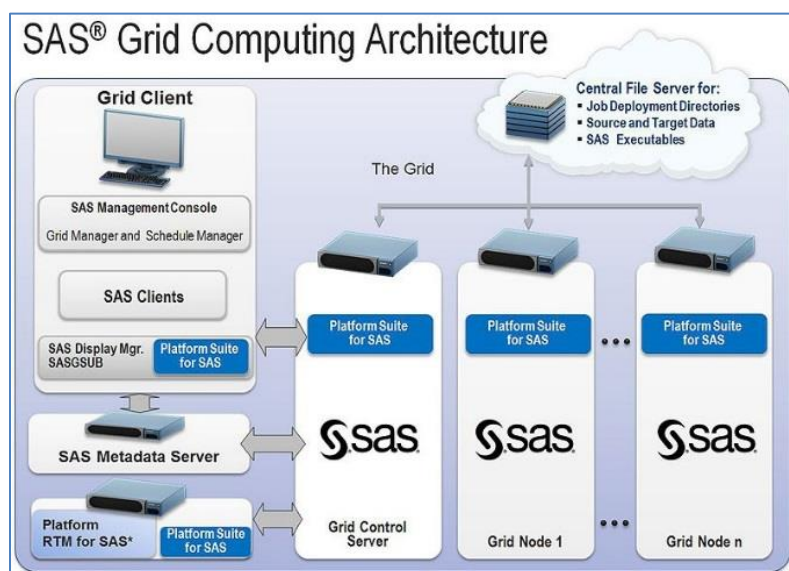


Figure 1. SAS Grid Architecture

In the following sections, we provide information about technical issues SAS programmers commonly face when making the switch from PC SAS to SAS Grid. These include statement, option, and object changes needed in programs, particularly those that interact with the external environment of folders, files, and platform-specific commands. We also discuss working with shell client software as a virtual interface or terminal to the Linux servers that host SAS Grid. We offer some simple examples of program submission, the SASGSUB utility, using batch jobs, and integrating Windows work with grid work. Throughout, we compare working in a PC SAS environment with working on a Linux SAS Grid, so programmers can anticipate and meet the challenges of migration.

MIGRATION FROM WINDOWS PC SAS TO A SAS GRID ON LINUX

From your viewpoint as a programmer, migration may require a series of steps, each of which must be planned, executed, reviewed, and possibly repeated. First, the grid must exist and function correctly; steps for setting up the architecture are beyond the scope of this document, but if it's a new system, you may be involved in testing the system at the same time you are learning how it is intended to work. Second, you must have access to the grid. Third, you need to be sure their access includes all of the necessary SAS components, drivers, and software needed for preparation and submission of programs. Finally, you have to modify your PC SAS programs for the new environment, including objects such as format catalogs, SAS statements within the program, connections, and parameters for reading or writing external data. Except for installing and configuring the SAS Grid itself, each of the programmer's steps is described in more detail in the sections below.

Though the word *migration* may sound like a well-defined, straightforward process, RTI's transition team found that even experienced programmers faced a steep learning curve. It is a different programming environment in which to use familiar programs, and like any change from one computing platform to another, the early weeks of use can be confusing.

There are many common pain-points at which the experienced SAS programmer may once again feel like a novice. Common items of confusion stem from differences between Windows and Linux operating systems, differences between grid-based processing and desktop processing, adapting to working through a terminal server or script file, or changing from direct interaction with SAS on a command line to use of the SAS go-between utility, SASGSUB.

The following are some of the stumbling blocks that programmers may encounter during migration:

- Where are the log and listing files?
- Why does SAS say that none of the paths are valid?
- Why did my LIBNAME and FILENAME statements stop working?
- Why does my program generate a format library error when it used to execute successfully?
- Why won't my SQL Server connection work?
- I used DDE a lot, but it's not available on Linux SAS-Grid. What can I do?
- Why won't my program read and write Excel correctly?
- My Windows share path has spaces in it, and it's not working with SASGSUB. How can I make it work?
- Many of my jobs are part of an overnight process on a Windows computer. What should I do?

While they may appear to be simple programming issues, each one results from the change in environment, not from programming errors. To resolve the issues, the troubleshooter needs to know whether the problem lies in the SAS code, an interaction with Linux, or the way the grid functions as compared to desktop computation.

Programmers can take several steps to ease the challenges of migration from PC SAS to SAS Grid and the related surprises of migrating from Windows to Linux as the programming platform. First and foremost, programmers benefit from access to good documentation. What SAS provides may give a clear picture of how grid processing works yet still leave the user wondering where to start. Documentation should be closely tied to the users' needs, and it should be readily available for repeated consultation.

In addition, if an organization can provide a set of simple programming examples that execute correctly on its own grid installation, those examples help illustrate what needs to be done and allow programmers to pinpoint the source of failure in code that doesn't work. Finally, programmers need someone to consult; one way to provide this expertise is for the transition team to include experienced programmers who have worked through the learning curve themselves and can produce sample code for use by novices to the grid.

We hope that these remarks alert you to the difficulties that can arise in migration from PC SAS to SAS Grid. Even more, we hope that the programming tips and examples that follow help you make the transition more easily than attempting to figure it all out on your own.

PROGRAMMING CHANGES

CHANGES NEEDED TO SAS PROGRAM OBJECTS

When changing from a SAS for Microsoft Windows environment to a SAS Grid environment on Linux, you need to update a few programming objects. Luckily, SAS datasets are not among them. SAS datasets can be accessed, read, and modified across environments. SAS uses Cross-Environment Data Access (CEDA) to accomplish this task behind the scenes. A note appears in the log file when CEDA is used. You should notice no difference, other than the expenditure of a small bit of extra processing resources.

However, SAS will not recognize any dataset that has had its name manually changed to have a name with mixed case.

If formats were created in an environment other than the Grid environment, you need to recreate them or move them in transport format. Indexes are also nontransferable and need to be created or moved in transport format to the Grid environment.

CHANGES NEEDED TO SAS PROGRAM STATEMENTS

When converting SAS statements from the SAS for Microsoft Windows environment to a SAS Grid environment on Linux, one of the most important things to remember is that a Grid in the Linux environment is case sensitive. This is a Linux characteristic, not a SAS characteristic. All path names in an existing SAS program must be updated to match the case of the path name in the Linux system. This includes LIBNAME statements, FILENAME statements, and any other programming statement that contains a path name.

In PC SAS statements like INFILE and FILE, it is acceptable to use only the name of a file; SAS will read or create the file in the current working directory. In a Linux Grid, it is best to have the SAS program contain the entire case sensitive path of the input or output file because, in most cases, users do not have permission to the default working directory in the Linux environment.

Programmers should remember to change programming statements that use operating system commands like X, Systask, or PIPE to include the Linux operating system commands, which are different from Windows operating system commands.

Finally, SQL connections are more difficult to navigate in SAS Grid environment on Linux. Linux does not recognize predefined local ODBC connections and does not support all SQL drivers. SQL server port numbers may be required as well. The SQL connection can be performed in a LIBNAME statement:

```
LIBNAME SQL_LIB ODBC complete ="driver=SQL Server;  
uid = userid;  
pwd=password;  
database=SQL database name;  
server=server name"  
schema=schema;
```

CHANGES NEEDED TO WORK WITH MICROSOFT EXCEL

Microsoft Excel is integral to many programmers' processes, from the data source to the product of a program. Unfortunately, many of the SAS statements programmers use to interact with Excel on a Windows PC SAS platform are no longer viable on the Linux Grid without some changes. As indicated in Table 1, most statements require some code modification, but nothing major. The exception is Dynamic Data Exchange (DDE), which is not available in any form on Linux SAS Grid. ODS provides some control over Excel output, but in some instances you need explicit control to write to specific Excel cells. For example, DDE is able to populate Excel templates, but ODS cannot. To overcome this stumbling block, RTI programmers developed an alternate way to achieve the same results using SAS, R, and IML (Kinney, Wilson and Carper 2017).

Excel Component	Caveat
DDE	DDE not available on a Linux SAS Grid
DBMS	DBMS=excel replaced by either DBMS=xls or DBMS=xlsx
GetNames=No	PC SAS: F1, F2, F3... Grid: A, B, C...
GetNames=Yes	Options validvarname=v7 (to avoid '_' in names with blanks)

Excel Component	Caveat
Mixed Option	Don't use <code>MIXED=YES</code> on <code>proc import</code> (The option is not available under Unix.)
Drivers	Microsoft drivers needed on grid servers (\$).

Table 1. SAS Code Changes Needed for Working with Microsoft Excel on Linux

SUBMITTING SAS PROGRAMS ON THE GRID

The SAS coding modifications we described for the Linux Grid allow you to function if you submit code interactively via SAS Studio or SAS Enterprise Guide. However, when systems surround the SAS programs, further updates are necessary to let SAS talk to the Linux Grid.

For the purposes of this paper, we define *interactive submission* as SAS Enterprise Guide and SAS Studio. Interactive methods of submitting SAS programs include the following:

- SAS Enterprise Guide
- SAS Studio

Batch methods of submitting SAS programs include the following:

- From Linux command line
- From a Linux shell script
- From other Windows programs (e.g., .Net, TextPad, .cmd, .bat)
- Right-click Batch Submit

When submitting interactively, programmers do not need to worry about further updates and can submit code that has been modified as described earlier.

One convenient batch method of submitting SAS programs in a PC SAS environment is from the context menu. With PC SAS, programmers could do this by right-clicking on a program and choosing Batch Submit from the menu. This not only ran the program but also located the log and lst files in the same folder as the program. This option was no longer available to the programmers at RTI with SAS Grid. To emulate this lost feature, RTI programmers developed right-click utilities that allow batch submitting and capturing the SAS Grid Linux path (Wilson, Green, and Terminello 2017).

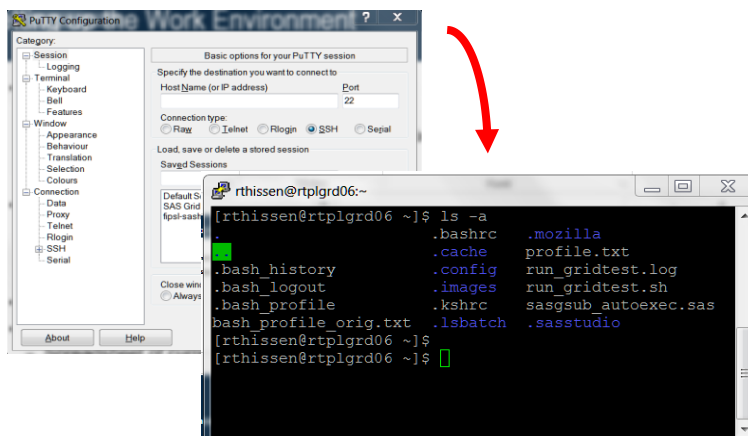
As mentioned earlier, we were experienced with a system running PC SAS on Windows. Many long-running projects used Windows-based programs to make the connection between SAS programs and various surrounding systems, such as shell scripts, .Net, .bat, or .cmd programs. Previously, these programs could call the SAS executable and trigger a SAS program. With the Linux Grid, we no longer had an executable and needed to find another way to make the connection.

To connect to the Linux server grid in a way that allows you to execute commands on it, you first need a connection from the PC to the Linux system. One easy way to do this is to install a terminal emulation program, often called a Secure Shell (SSH) or SSH client software.

Before we continue to describe code migration from Windows system calls to Linux calls within .bat and .Net programs, we need to explain why we need at least one of these terminal emulators. With the Grid, SAS now lives on remote servers. To submit our programs, we must be able to communicate with these servers. SSH software packages serve that purpose for us. There are more available, but we will only discuss the ones we have used in our migration at RTI.

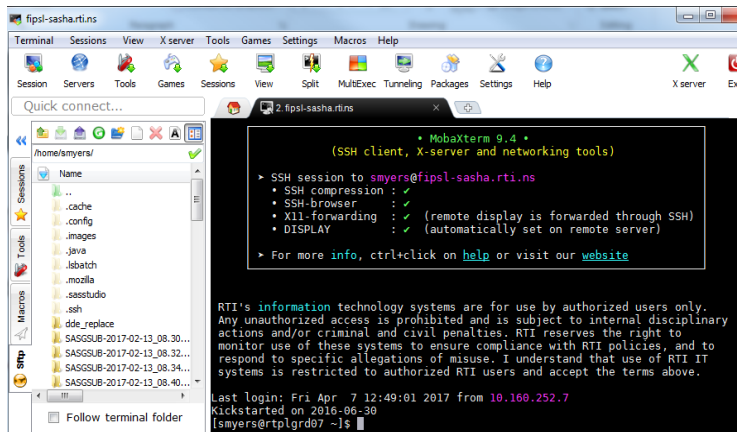
One of the most important features we needed from SSH was the ability to suppress prompts for passwords. Because many systems are scheduled to run with no one actively watching the process, it is imperative that no manual intervention be required.

At RTI we initially chose PuTTY as our preferred SSH (Display 1). It worked very well, but the setup and interface were not user friendly.



Display 1. PuTTY SSH Client Interfaces for Windows and Linux

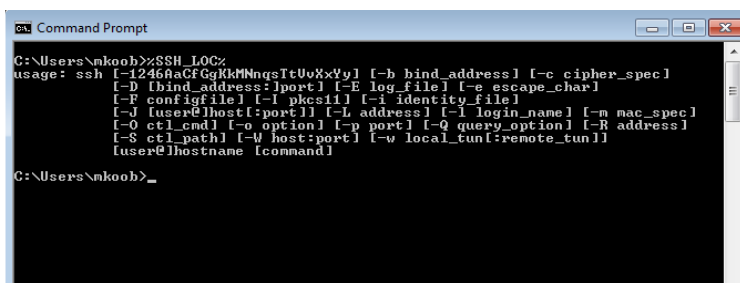
Many teams then moved to MobaXterm, which also allows password suppression (Display 2). The benefits of



Display 2. . MobaXterm SSH Client Interface

MobaXterm included an easier setup for nontechnical staff and a GUI interface.

A third SSH in use at RTI is Git Bash (Display 3). We chose Git Bash over PuTTY and MobaXterm because it proved to be more stable in some circumstances.



Display 3. Git Bash SSH Client Interface (from PortableGit)

There are many SSH options available, and we do not endorse any particular one. However, we recommend that you thoroughly test whichever SSH you select to evaluate pros and cons for each situation. It is also necessary to coordinate the SSH software within teams. Although individuals may use

whatever SSH they prefer, each team should reach consensus on which package to use. It may be advantageous to use a Windows environment variable for the SSH executable. This alleviates issues when team members operate under different versions of the same SSH software. Shared programs may then call the windows environment variable rather than requiring that all members remain at the same version.

SASGSUB

The SASGSUB utility is a command that, in its simplest form, submits a SAS program to the grid. This command works in both the Windows and Linux environments, but the Linux system's case sensitivity requires that you submit the SASGSUB command using all lowercase letters.

In addition to submitting jobs to the grid, the SASGSUB command provides some very useful features, including viewing job status, retrieving results, and terminating jobs.

Here's an example of a simple SASGSUB statement:

```
sasgsub -GRIDSUBMITPGM mypgm.sas -GRIDWAIT -GRIDWORK /server/share/path
```

GRIDSUBMITPGM is a required argument that you must include when invoking the SASGSUB command. The name of the program to be submitted must follow the argument.

The following optional arguments are very useful:

- **GRIDWAIT**—specifies that the SAS Grid Manager Client Utility waits until the job has completed running, either successfully or with an error.
- **GRIDWAITRESULTS**—functions similarly to the **-GRIDWAIT** argument but also returns the results of processing.
- **GRIDWORK**—specifies the path for the shared directory that the job uses to store the program, output, and job information.
- **GRIDSASOPTS**—specifies any SAS options that are applied to the SAS session started on the grid.

As discussed earlier, the Linux environment is case sensitive, so the SASGSUB command must be in lower case. The arguments to the SASGSUB command do not share the same case sensitivity. However, any path or file names that are used with those arguments are case sensitive in the Linux environment.

You can use the SASGSUB utility on a command line or in a batch file. Table 2 shows some examples of how to use SASGSUB.

Topic	Comments
Log and listing files	• Where you expect them: /server/share/path/ • Where you find them if you do not use the -gridwork options: <code>/sas/grid/gridwork/username/SASGSUB-yyyy-mm-dd_hr.mm.ss.sss_mypgm/</code> • Where you find them if you use the -gridwork option: /server/share/path/username/SASGSUB-yyyy-mm-dd_hr.mm.ss.sss__mypgm/
Altlog/Altprint (use it in a Linux script)	<pre>cd '/server/share/path' here=\$(pwd) sasgsub -GRIDSUBMITPGM mypgm.sas -GRIDWAIT -GRIDSASOPTS "(-altlog '\$here' -altprint '\$here')"</pre>
Sysparm	<pre>sasgsub -GRIDWAITRESULTS -GRIDSUBMITPGM /server/share/path/mypgm.sas -GRIDWORK /server/share/path -GRIDSASOPTS "(-sysparm 'YYYY Q')"</pre>

Table 2. Examples of Using SASGSUB

Redirecting the output files (i.e., the log and lst files) is probably one the most useful options SASGSUB offers. There are a few ways to accomplish this task. The -GRIDWORK argument is the simplest; use it in the SASGSUB statement followed by the folder path that should contain the output files. -ALTLOG and -ALTPRINT perform the same task and can be used as an alternative to the -GRIDWORK option. They are specified with the -GRIDSASOPTS option and move the SAS created output to any folder that is specified. If the output belongs in two separate folders, the -ALT* options are the way to go.

PUTTING IT ALL TOGETHER

Most programmers start as Windows PC SAS users; we have described many of the changes necessary for migrating Windows work to the Linux grid. On Windows with PC SAS, the process is straightforward: You write your program and submit it. But for those who have a Windows desktop and want to run batch jobs using SAS Grid on Linux, there may be many steps, each of which has to work correctly, and troubleshooting takes more sleuthing ability than simply reading the SAS log. For the grid novice who is also a Linux novice, it can be intimidating.

Yet aspiring Linux programmers can learn the pieces little by little because the effort of creating and running a new SAS job on the grid may be fairly simple or quite complicated, depending on what you need to do. If you work interactively through SAS Studio or SAS Enterprise Guide, you only need to learn a few things. If you have Windows programs that need to launch SAS jobs on the grid interspersed with other software jobs that run under Windows, you have some work to do and may need to master all the steps in Figure .

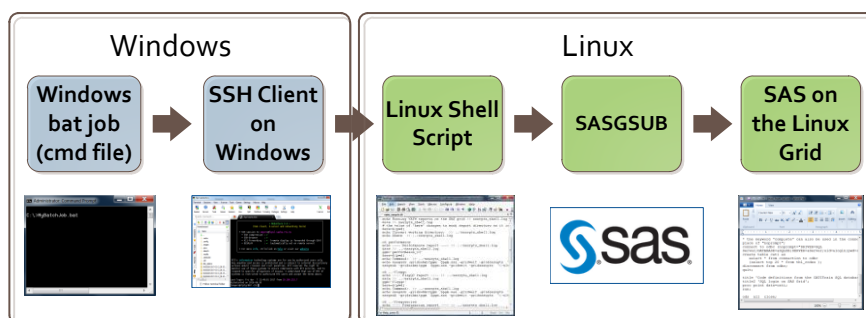


Figure 2. Full Sequence of Potential Batch Processing Steps

Learning is easiest if you start with the simplest operation and follow it with more complex ones. In Figure that means starting with SAS on the Linux Grid, then trying SASGSUB and Linux shell script, and finally learning to manage the full left-to-right sequence. The following scenarios correspond to learning each step.

Scenario 1: SAS on Linux. Interactive work from Windows requires the least learning, and learning to use the interactive environment aids in learning more challenging ones.

1. If you are using any external files, determine the Linux-compatible path to the network location for your input or output data. This information depends on your organization's installation, and it is likely that the paths will differ at least somewhat from Windows paths.
2. Open SAS Studio or SAS Enterprise Guide.
3. Import or write your program, following the tips offered earlier for making changes to programs that used to run on PC SAS.
4. Execute the program and view the results within the interface.

Scenario 2: SASGSUB. Assuming you have successfully accomplished Scenario 1, you can begin working directly within the Linux environment. Working from the Linux command line is more challenging than interactive work from Windows, but doing so lets you learn and control more of what takes place during execution. From a command line, SAS Grid programs must be executed through SASGSUB, which acts as an intermediary between the program and grid management components (please refer to Figure 1 for SAS architectural details).

1. Install and test the SSH client software of your choice, as discussed earlier. Use the command line to confirm that you can navigate to the desired network locations for your SAS programs and files.
2. Create a simple SAS program to use in testing, such as one that contains only a single LIBNAME and a couple lines of code. Be sure to save your program as a text file in Unix format. Because Linux is a version of Unix, doing so provides the correct end-of-line control characters within the file.
3. Submit the program by using SASGSUB and its options as described earlier in this paper. You may want to pass parameters to SAS for the output, log, and listing files to make them easier to locate after execution is successful, as shown in this example:

```
sasgsub -GRIDSUBMITPGM mypgm.sas -GRIDWAIT -GRIDWORK '/server/share/path'
```

Scenario 3: Script File. Embedding the SASGSUB command in a script file gives you additional command-line options and lets you execute a sequence of programs without interacting.

1. Write the SASGSUB command and parameters from Scenario 2 into a text file, typically with the file extension ".sh" to indicate that the file contains a shell script. Be sure the file is saved in Unix format. The web offers many good references for mastering Linux commands and writing script files beyond simply using the SASGSUB command.
2. Open the SSH client, log into the Linux network, and submit the script file from the command line. For example, if your installation uses the Bourne Again Shell (bash) version of Linux, do the following:

```
bash myscript.sh
```

4. Optionally, you can use the Cron command to schedule the script to run unattended on the Linux platform. Linux commands are outside the scope of this paper.

Scenario 4: Windows Command Line. You can run some SSH clients, including the three mentioned earlier, unattended on a Windows platform by invoking them through a command window or from another program. You need to explore your SSH client's documentation for command-line syntax, encryption keys, password passing, and other details. Some examples of Windows command-line execution of Linux shell scripts follow, with each example intended to be submitted as a single line.

- PuTTY without a shell script:

```
"c:\program files\putty\putty.exe" -batch "myPuttyConnection"
"cd /server/share/path; sasgsub -GRIDWAIT -GRIDSUBMITPGM
mypgm.sas -GRIDWORK /server/share/path"
```

- PuTTY with a Linux shell script:

```
"c:\program files\putty\putty.exe" -batch "myPuttyConnection"
"cd /server/share/path; bash myscript.sh"
```

- 3. MobaXterm without a shell script:

```
"c:\program files\MobaXterm\MobaXterm_Personal_9.4.exe" -exitwhendone
-exec "ssh -t myLinuxLogin 'cd /server/share/path; Sasgsub -GRIDWAIT -
GRIDSUBMITPGM mypgm.sas -GRIDWORK /server/share/path'"
```

- 4. MobaXterm with a Linux shell script:

```
"c:\program files\MobaXterm\MobaXterm_Personal_9.4.exe" -exitwhendone
-exec "ssh -t myLinuxLogin 'cd /server/share/path; bash myscript.sh'"
```

Scenario 5: Windows Batch Job. The commands listed in Scenario 4 can be embedded in a Windows command or .bat file, or called from other programs such as Microsoft .Net applications or PowerShell, and those programs can be scheduled to run unattended through Windows scheduling functionality. The value of using a Windows-based job is that you can interleave Windows and Linux programs in a single

process, such as starting a Windows-based application to generate a file that in turn serves as input to a SAS Grid program. Output from the SAS program may feed into a subsequent Windows job.

In each scenario, if something does not work correctly, it may be necessary to retrace all the earlier scenarios to determine where the failure occurred. For that reason, we advise mastering all of the approaches, from simple interactive work in Scenario 1 through the most complex scenario that you are likely to use.

After you put together a full sequence with a simple example or testing job, you have a working example to use as a template or guide for later work. Creating and sharing such jobs with colleagues can ease the stress of migration from PC SAS to SAS Grid on Linux.

CONCLUSION

Conversion from PC SAS to Linux Grid SAS is not trivial, particularly when complex systems are involved. The learning curve can be steep if project teams are inexperienced with the Linux operating system. With any software update, many requirements are dependent on the user's specific environment. As detailed by Wilson, Green, and Terminello (2017), even the updates needed to change a folder path from Windows to Linux may be much more labor intensive than merely changing back slashes to forward slashes. Extraneous factors, such as folder aliases, may complicate program updates more at some institutions than others with similar paths for Linux and Windows. One common theme, independent of specific grid implementation, is to expect the unexpected.

When an organization chooses a Linux platform, it is a double transition—one to the Grid and another to Linux. We caution you to not underestimate the time it will take to make the leap.

Finally, as trivial as it may sound, support groups are critical to the migration progress. The technical aspect is one level of challenge, but the mental aspect is a challenge of its own. Expert coders are humbled at the level of relearning required for systems that have been in place for long periods. Sharing code among programmers who are making the transition is key to a successful transition. As mentioned earlier, the simple task of locating resulting log and listing files can take an inordinate amount of time. The need for a coordinated effort to communicate results and tips among programmers cannot be emphasized enough (Gordek 2017). It is important to keep everyone who is involved engaged and provide peer support. We sincerely hope our experience in our own migration helps others who are presented with a similar challenge.

REFERENCES

- Wilson, David, Annette M. Green, and Karen Terminello. 2017. "Getting on SAS Grid, Our Journey." *Proceedings of the 2017 South East SAS Users Group (SESUG) Conference*, Cary, NC.
- Kinney, Satkartar, David Wilson, and Ben Carper. 2017. "A SAS macro replacement for Dynamic Data Exchange (DDE) for use with SAS grid computing." *Proceedings of the 2017 SESUG Conference*, Cary, NC.
- Gordek, Harper. 2017. "Intracompany Social Media Support for a SAS GRID Migration." *Proceedings of the 2017 South East SESUG Conference*, Cary, NC.
- Brinsfield, Eric C. 2017. "SAS Programmer's Guide to Life on the SAS Grid." *Proceedings of the 2017 PharmaSUG Conference*, Baltimore, Maryland. Available at <https://www.pharmasug.org/proceedings/2017/BB/PharmaSUG-2017-BB11.pdf>
- Lopuski, Amanda, and Yongxuan Mike Tan. 2017. "Migration to SAS Grid: Steps, Successes, and Obstacles for Performance Qualification Script Testing." *Proceedings of the 2017 PharmaSUG Conference*, Baltimore, Maryland. Available at <https://www.pharmasug.org/proceedings/2017/AD/PharmaSUG-2017-AD16.pdf>

ACKNOWLEDGMENTS

We would like to thank David Rice and Kelly Rickard for their efforts in editing and formatting this paper, as well as all the members of RTI's SAS Grid Migration Team for their collaboration during the transition.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Susan Myers
RTI International
919-541-7441
smyers@rti.org

Inga Allred
RTI International
919-541-7072
irb@rti.org

M. Rita Thissen
RTI International
919-485-7728
rthissen@rti.org