Efficient model running with automation

Jaap van der Velde
Greg Collecutt

Efficient Model Running with Automation

Jaap van der Velde
TUFLOW (BMT)

Greg Collecutt
TUFLOW (BMT)

"Running a model efficiently relies primarily on making the right planning and design decisions. But once you know what results you need, and your models have been properly set up, there are many tools, tips, and tricks that can help you run them more efficiently. [..] Whilst the presentation will cover a great variety of technical subjects, like Python, command line scripting, hardware configuration, cloud computing, you don't need to be familiar with any of these to get something out of this webinar."

Overview

Motivation

- Why this a webinar on automation, and why now?
- Why automate at all?

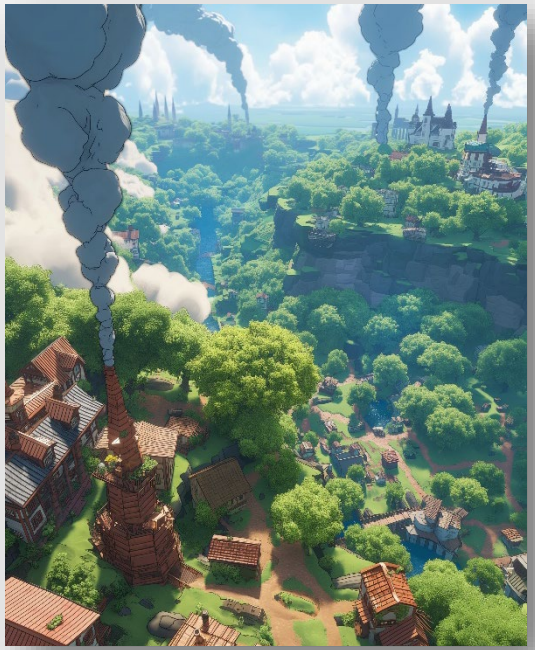
Basics of command line automation

Where and how automation can make a difference

- Infrastructure
- Model / Data
- Application / Workflow

Limitations of automation, a dose of realism

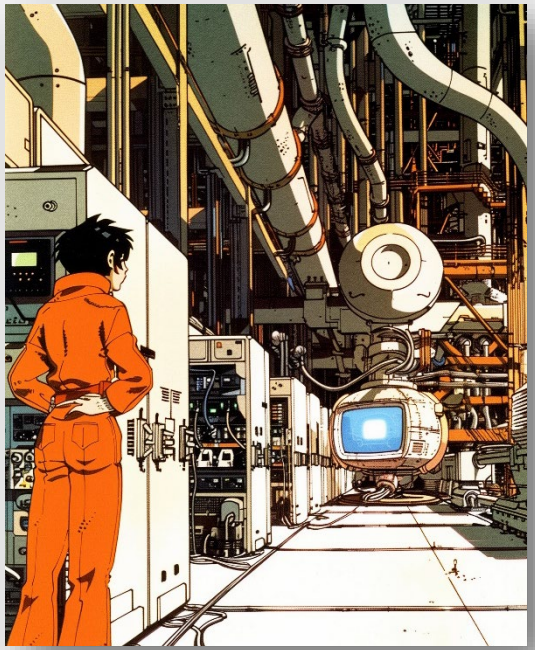
Your Questions & Answers.



Motivation

Why a webinar on automation, and why now?

- Personal: background as software developer
  - When you see everything as a software problem.
- Professional: relationship between SE / IT & modellers
  - Wasted potential of people, hardware, and licences.
  - When is a problem for you to solve, and what is for “IT”?
- The elephant in the room: take the bull by the horns (tusks?)
  - LLMs to explore, learn, get started, and more
  - Useful across all applications; useless without some understanding

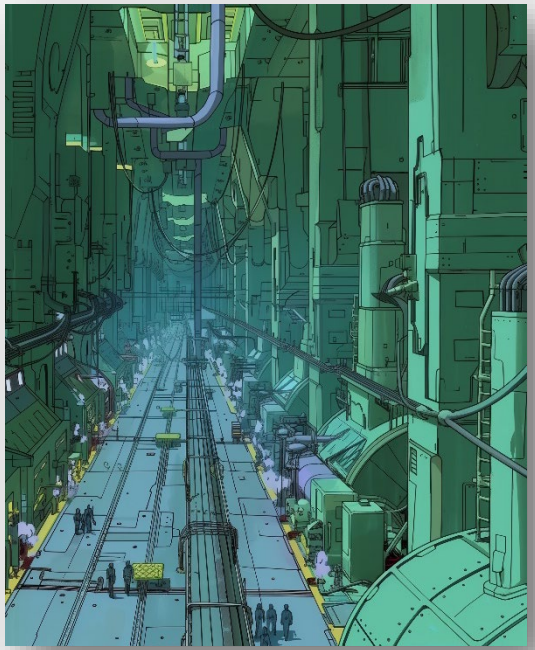


Motivation

Why automate at all - principal motivations

Automating model running allows you to be

- Efficient
  - Eliminate manual intervention
  - Optimise resource use: people, hardware, licences
- Consistent
  - Minimise human error
- Scalable
- Repeatable & Reproducible
  - Do the same time every time & confidently produce the same outputs



Motivation

Why automate at all?

Efficient, Consistent, Scalable, Repeatable & Reproducible

Secondary advantages

- Benefits Collaboration
- Flexible and Adaptable
- Allows 24/7 Operation
- Reduces (uncaught) Errors
- Empowers Users

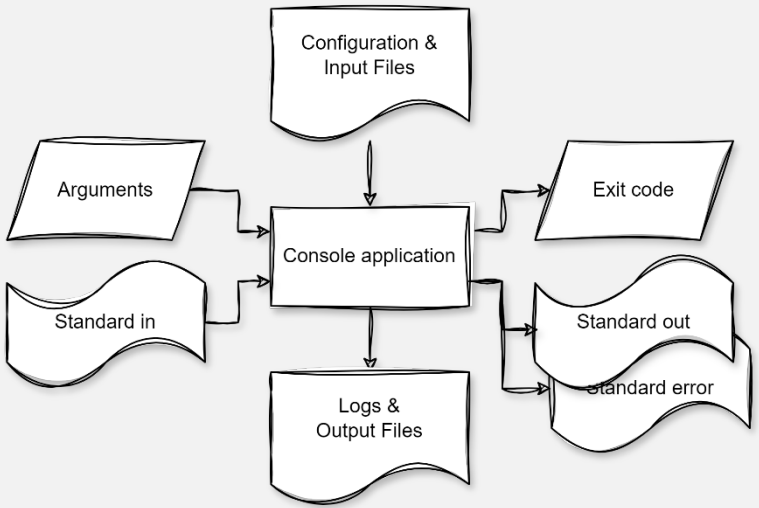
Basics of command line automation

All console applications, regardless of platform or operating system, operate on the same principles:

- Whether started from a command line interface (CLI) or from a script, a console application is passed a series of *arguments* on the command line, which may be required or optional, based on the application's requirements.
- When it completes, a console application returns an integer value *exit code*, which will indicate whether the application exited normally (exit code zero) or with some error status (exit code non-zero, typically one).
- A console application takes its text input from the '*standard in*' input/output (I/O) stream and writes its text output to the '*standard out*' I/O stream. Error text is sometimes written to the '*standard error*' I/O stream.

In a shell, or a shell script, these streams can be redirected and 'piped' from one console application to another, creating a 'pipeline'.

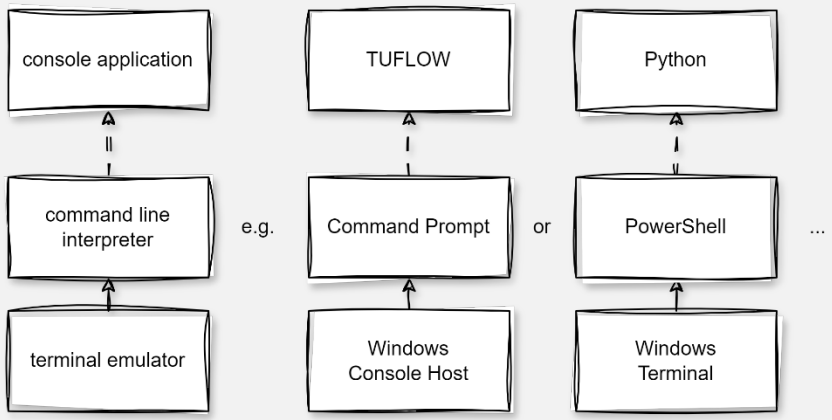
- While running, a console application may read additional instructions from configuration files, read and process input files, write output files, and write logs. Of course, other I/O is also possible (network messages, etc.)



Basics of command line automation

- The most important phrase on the previous slide: "*All console applications, regardless of platform or operating system*"
- 'Command Prompt', 'PowerShell', and 'bash' are examples of command-line interpreters (CLIs).
- On Windows, you typically start a CLI using a terminal emulator like Windows Console Host or Windows Terminal – perhaps without realising.
- A single running instance of a CLI is a process like any other, with its own environment, called a 'shell'.

Note: when running console applications directly from a shell, they run as part of the current session, taking over its input and output streams.



Basics of command line automation

Understanding the basics helps you create interesting applications or understand why your scripts may behave differently from what you expect.

- TUFLOW is a console application like any other.
- It has specific arguments that change how it behaves on the console, causing it to be entirely quiet, or not open any dialogs and behave entirely like a text-only console application.
- Consider that you can redirect the output on screen from TUFLOW to a file or create a pipeline to filter its output as it is written, for example with a Python script.

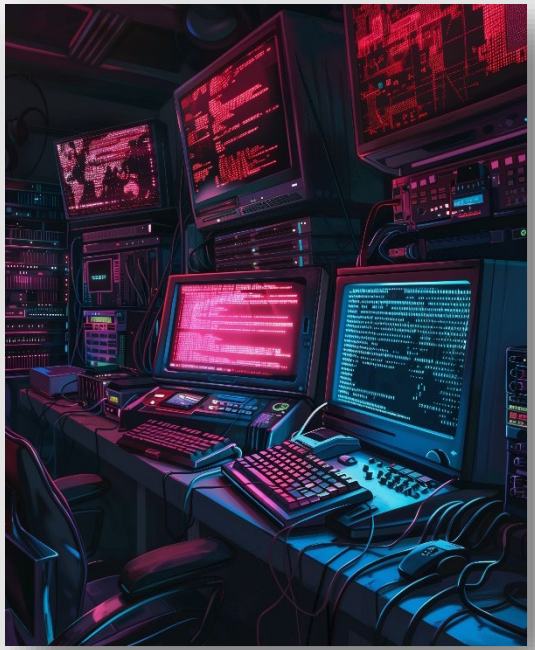
These concepts are not just useful when using a CLI in a terminal emulator (i.e., a ‘window’) but equally so when writing a batch file, PowerShell script, or Python script.

Basics of command line automation

- Command Prompt and batch files (.bat)
  - Traditionally, TUFLOW users rely on batch files for automation.

    (tip: when in doubt, `call` other programs instead of just running them)
- PowerShell and PowerShell scripts (.ps1)
  - PowerShell provides more commands and structure, and object pipelines.

    (tip: if you cannot run `powershell`, try `pwsh` and consider an alias)
- Bash and bash scripts (.sh)
  - Bash scripts are primarily useful on Linux (and thus TUFLOW FV).
- Python and Python scripts (.py)
  - More powerful on either Windows or Linux, but more requirements.



Basics of command line automation

Python, an automation tool, or a replacement even?

- Python is a console application like any other.

Python is not *as* easy as the alternatives for:

- Executing other commands.
- Piping and redirection streams.
- Running without any installation.

But once installed and with some basic training, you can perform these tasks – and everything else – with Python, a lot better and often faster. Worth considering – if you can count on having Python everywhere.

(tip: consider using conda + conda-forge over pip + pypi, given the availability of geospatial packages there)

Basics of command line automation

A few more key concepts for automation:

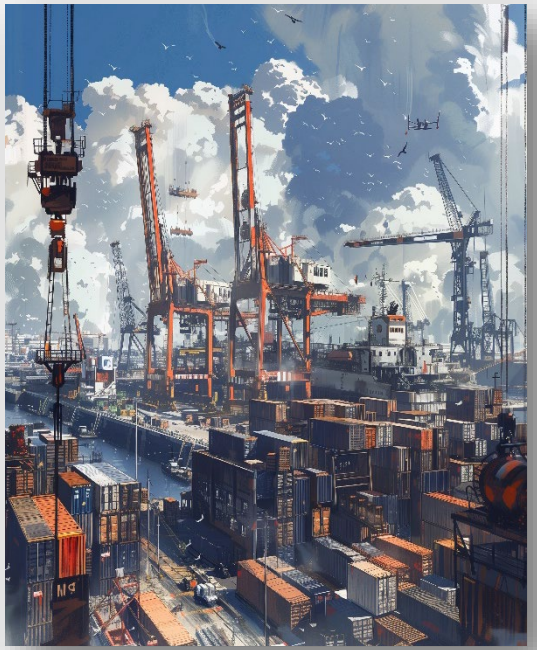
- environment variables
- working directory
- absolute and relative paths
- network locations, UNC paths (and drives on Windows)
- user context

Applied Automation – Infrastructure - Network

- Know your topology
  - What virtual locations matter to you?
    - Storage, Compute, and User Access (tip: users care about latency, everyone talks bandwidth)
  - Where are they physically, and how are they connected?
- Design your workflow accordingly
  - Avoid detours, no going from A to A via B
  - Understand where best to read, write, log
  - Put your compute near your data
  - When you can't get it right in one go, automate the logistics

Applied Automation – Infrastructure - Storage

- Know your storage
  - How much space do you need, and how much is truly available?
  - What storage is fast and why, what is backed up, what are the costs?
- Horses for courses
  - Scratch: fast, disposable, right next to compute
  - Persistent: available, safe, convenient access; consider databases vs. file storage
  - Archival: cheap, durable, low maintenance
- Standardise access
  - For large Windows networks, look at DFS-N; set standards, and follow them
  - Consider generating configurations from templates, or make use of environment variables
- Alternatives to keeping data
  - Recomputing data can be faster & cheaper than storing & moving it around



Applied Automation – Infrastructure - Compute

- Know your hardware
  - Learn what matters and collect data: cores, memory, GPU model, GPU memory, etc.
  - Is the hardware dedicated to your work, or shared?
- Configure for optimal performance
  - Count physical cores on your CPU, not logical cores (hyperthreading)
    (tip: consider disabling hyperthreading on dedicated model running computers)
  - On very new Intel processors, count P-cores, not E-cores, and avoid them for your models
    (tip: from scripts, consider setting processor affinity for a process, especially when running models in parallel)
  - On shared environments (data centre or cloud), partition GPUs, don't share them
    (tip: select NVIDIA hardware that supports MIG for real partitioning, don't blindly accept shared VDI solutions)
  - Ensure you have 2 cores minimum per GPU
  - Only combine multiple GPUs on a single run, if the size of your model requires it
  - Ensure you have 4-6x the RAM available for the CPU compared to RAM you need on the GPU
- Automate compute configuration
  - Standardise model running systems by scripting (de)installation of software, licensing, and scripting environments, and anything else specific to your model running workflow.

Applied Automation – Infrastructure - Cloud

- Virtual Desktop Infrastructure
  - Ideal to do all your work in the cloud but keep an eye on shared GPU resources.
- Batch Services
  - Azure Batch and AWS Batch
    - Full-featured: Provisioning, Scheduling, Setup, Teardown, Access control
    - Costly mistakes, complex APIs (both Python and CLI tools)
- Specialised Cloud Modelling Platforms
  - SCALGO
  - Sentient Hubs
  - BMT Deep
- Simple VMs
  - Sometimes all you need is a fast computer to do some work.

Applied Automation – Model / Data – Inputs & Outputs

- Collection and Extraction
  - Consider downloading data on demand, where and when you need the data
    (tip: look at tools like GetAtmos, GetOcean, etc. for TUFLOW FV for example, and give us feedback)
  - Model runs with large inputs and very different events and scenarios may not need all data
    (tip: TUFLOW specifically allows copying or listing files that are needed for a run for automation, with -c options)
- Logistics
  - If you cannot compute close to the storage (particularly output), consider moving your data to and/or from the compute instead of directly accessing it remotely.
  - Perform any file logistics with standard and fast system tools, like Robocopy on Windows, which allow resuming an interrupted copy or move and provide feedback on the result.
- Analysis and Processing
  - Outputs can be extremely large, even with a properly configured model without needless outputs.
    - Consider performing early analysis and post-processing on storage where outputs were written.
    - Environments like JupyterHub or running JupyterLab on model running machines can provide efficient User Access for running Python scripts on data that would otherwise need to be moved.

Applied Automation – Model / Data – Configuration

- Get the model configuration right
  - Test a configuration before shipping it
    (tip: TUFLOW's test option -t results in an exit code indicating success or failure)
  - Avoid bad configurations by generating them (fully or scaffolding)
  - If you don't run the model where you designed it, check paths where you do
- Configure your automation
  - TUFLOW is popular in part because of its configurability, do the same for your automation and drive it with configuration
    (tip: consider JSON or YAML, these are popular and well-supported formats, both for authors and processing)
- Develop your models, don't edit copies
  - Structure your input databases for efficient reuse, selectively copy what you need
  - Version your configurations, developing a model is very similar to developing code
    (tip: nearly every software developer prefers Git for versioning, consider it for your model configurations)

Applied Automation – Application / Workflow - Schedule

- Scheduling is hard, and it shows
  - On Linux there are good options, but these are not trivial to set up
    - Slurm (or Torque / PBS for CPU-based compute)
  - Relatively few Windows-based solutions are available
    - Networked, some have had success with HTCondor
    - Locally, TRIM, and TUFLOW Runner for some use cases
      (Tip: TUFLOW Runner can also be used from the QGIS Runner)
- Azure Batch & AWS Batch
  - Great on a per-project basis, if compute cost is not the main concern
  - Require a great deal of development, and a difficult cost trade-off

Applied Automation – Application / Workflow - Run

- Running self-contained
  - Run the engine in a suitable mode, no user interaction
  - Configure your model to be self-contained, or have a clear and limited set of dependencies
  - Consider containerisation (Docker) as an option, if on Linux
    (tip: consider podman as an alternative, which can work in more restrictive security environments)
- Monitoring & Reporting
  - Set up your logging locations to be remotely accessible or write the logs remotely.
  - TUFLOW logs (.tlf, .hpc.tlf) are essentially recordings of what the application writes to standard out, so you could redirect or process that stream.
  - Consider processing outputs during or after a run, just don't lock them.

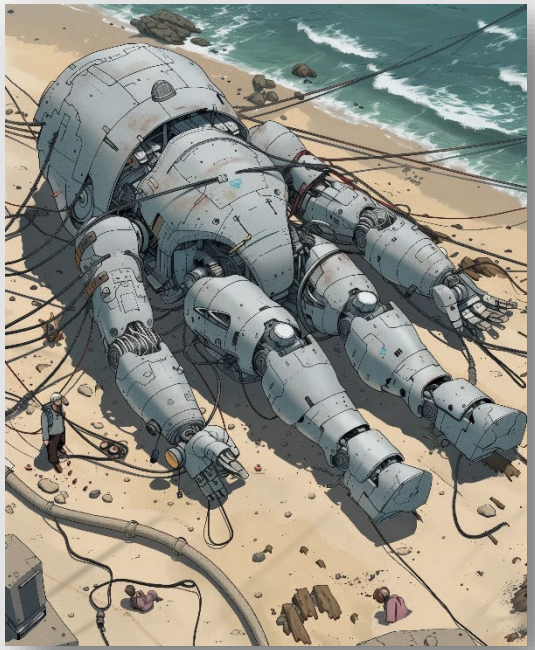
Limitations of automation

Automate, keep it real – automation is development

- You will have to maintain and support what you put out

  (tip: YAGNI, KISS, DRY)

- IT may impose restrictions and limit rights
  - Security norms or practical limits can restrict you
  - Administrative rights may be needed for some tasks
- Consider the handover
  - Can your client deal with a solution as delivered? Do they need to?
  - Whose intellectual property is the solution, and can you share it?
- Prefer an existing good solution over building a perfect one



Questions?