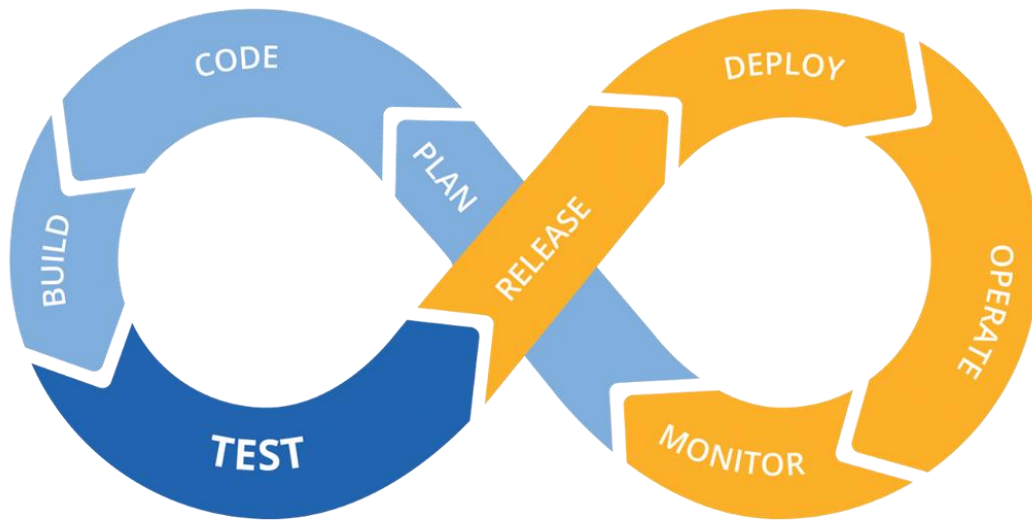


Automatic Build Pipeline

Sven van Huessen
232380



Contents

State of the Art.....	3
Typical Setups	3
Common Challenges	3
Shortcuts	3
Why?	4
What?.....	4
How?	5
Server set-up.....	5
Discord bot.....	5
Perforce syncing.....	6
Building the game	6
Zip it and ship it.....	6
Proof of Concept	7
Syncing the files	7
Compiling/cooking the game.	8
Zip it	8
Ship it	9
Combine it.....	9
Adopt into the Project	10
Feedback	11
Automatic builds	11
Naming convention checker	11
Future Improvements	12
Naming conventions	12
Speed	12
Reflection	13
Relevance	13
Python	13
Discord	13
Server	13
Sources.....	14

State of the Art

To understand the state of the art for automatic build pipelines, particularly in the context of game development using engines like Unreal, it's useful to consider several aspects: typical setups, common challenges, prevalent shortcuts, and best practices.

Typical Setups

- **Tools and platforms:**

Commonly used automation tools include:

- a. Jenkins (<https://www.jenkins.io/>)
- b. Travis CI (<https://www.travis-ci.com/>)
- c. CircleCI (<https://circleci.com/>)
- d. GitLab CI (<https://docs.gitlab.com/ee/ci/>)

Most used tools for version control are

- e. Perforce (<https://www.perforce.com/>)
- f. Git (<https://git-scm.com/>)

- **Pipeline Stages:** A typical pipeline involves stages for pulling the latest code, compiling code, running tests (unit, integration, etc.), packaging builds, deploying to test environments, and then pushing to production if all checks pass.

Common Challenges

- **Complex Dependency Management:** Game projects often have complex dependencies that need to be correctly configured in the build system to ensure consistent builds.
- **Large Build Sizes:** Games typically have large assets which can slow down the build process significantly.
- **Branch Management:** Handling multiple development branches simultaneously can complicate the build process, requiring careful management of which branches are being built and when.

Shortcuts

- **Partial Builds:** Instead of building the entire project, developers might set up the pipeline to build only parts of the project that changed. This is risky if dependencies are not correctly managed.
- **Skipping Tests:** Sometimes, to speed up the build process, developers might skip running tests or run only a subset of tests. This increases the risk of bugs going undetected into production.
- **Hardcoding Values:** To simplify configuration, developers might hardcode paths, credentials, or settings into build scripts, which can lead to security and maintainability issues.

Why?

Why did I choose this research topic?

My fascination with creating tools and programs to help game development began during previous projects. For instance, in the first year's Block B, I developed an endless runner game that required players to dodge obstacles. Given the variety of obstacle models available to me, I decided to enhance the game's complexity by combining different obstacles to create more challenging gameplay.

To integrate these combined obstacles into the game, I needed a method to position them at will and then incorporate the configurations directly into the game. Initially, I considered using Blender to merge the obstacles and save them as a new object. However, this approach was resource-intensive, requiring significant computer memory to save and load multiple models.

My solution involved using a 3D software to arrange the obstacles and then creating a script to interpret the software's save file. Instead of generating new 3D models, my script produced a JSON file detailing the position, rotation, and scale of each model within the scene. This method allowed the game to load individual obstacle models and position them according to the JSON file's specifications, thereby optimizing the use of resources.

This experience is one example of how external software can help the development process. Leveraging my degree in Software Development, I believe I can significantly simplify the building process for my team, making it as efficient as possible.

What?

As mentioned in the previous paragraphs, there are already numerous options available for continuous integration, and some of these options are compatible with Unreal Engine. However, these solutions do not support integration with Discord.

Considering the widespread use of Discord, I will develop a continuous integration system that allows team members to easily create new builds and publish them on Itch.io through Discord by using simple commands (see [Discord bot](#)). And instead of communicating through email, team members will receive notifications directly on the Discord server.

This integration facilitates quick testing and sharing for the team, eliminating the need to distribute zip files. Additionally, it will assist developers by enforcing consistent naming conventions and coding guidelines, resulting in a cleaner and more organized project for everyone.

Having the build processing being done on a different machine instead of your own will allow you to start a build and work on something else that having to wait for the build to be done.

How?

In this section, I will outline the technical aspects of the [pipeline stages](#) and the expected user interactions.

Server set-up

I will deploy a dedicated server to handle all operations, ensuring that as the developer, I only need to configure the system once. This approach eliminates the need for each user to run the system locally, which would complicate matters due to the challenges of installing necessary libraries on various systems. Therefore, the server will operate continuously, capable of executing tasks at any time.

Proxmox:

<https://www.proxmox.com/en/>

Windows Server 2019:

<https://www.microsoft.com/en-us/evalcenter/evaluate-windows-server-2019>

Discord bot

The server will host a Discord bot that monitors and responds to commands issued within a Discord server. The bot will trigger scripts to manage various tasks based on these commands. Below are some key commands I plan on implementing:

- **/ping** - Checks if the server is online and the discord bot is running
- **/build** - This primary command performs several actions
 - Retrieves the latest files from perforce
 - Compiles the game
 - Zip the game
 - Upload to Itch
- **/sync** - Synchronizes the latest files from Perforce to the server and displays updates in Discord.
- **/check** - Reviews file naming conventions and alerts the user of any issues via Discord. Further research is needed to determine file contents.
- **/maps** – In the [Common Challenges](#) I explained how large build sizes can slow down the process. For that reason I'm adding the option for the team to add which maps need to be included for the build. This will reduce the size drastically.
- **/link** – link a discord username to a perforce username. This is so the program knows who to ping when receives files from Perforce

Perforce syncing

Like explained in the [State of the art](#) every CI needs to have a version control that handles all the files. Because school uses Perforce and all of my team members know how to use it. I will use this software to handle the version control.

After doing research on perforce syncing, I found that perforce has their own Python library. This library will allow me to easily get the latest files. It will also allow me to view who submitted what files were submitted etc.

P4Python:

<https://www.perforce.com/manuals/p4python/Content/P4Python/python.programming.html>

Building the game

Once files are synced from Perforce, I will utilize the Unreal Automation Tool (UAT) via Python's subprocess library. This setup initiates Unreal Engine in the background to compile the game into an executable file.

Unreal Engine UAT:

<https://unrealcontainers.com/docs/use-cases/continuous-integration>

Subprocess Python:

<https://docs.python.org/3/library/subprocess.html>

Zip it and ship it

After compiling the game, I will use Python's Shutil library to package the build into a ZIP file. Following this, the build will be uploaded to Itch.io using Butler, a command-line tool provided by Itch.io for managing uploads.

Shutil:

<https://docs.python.org/3/library/shutil.html>

Butler:

<https://itch.io/docs/butler/>

Proof of Concept

To automate everything I first had to know if every step of the process is possible. So instead of making everything at once, I made every step it's own functionality. Below is my process working on the proof of concept.

Syncing the files

Before you can compile the game into a playable .exe, we first have to get all the latest files so that everyone's work will be inside the game. like explained in [How?](#) I will use the P4Python library to sync to perforce.

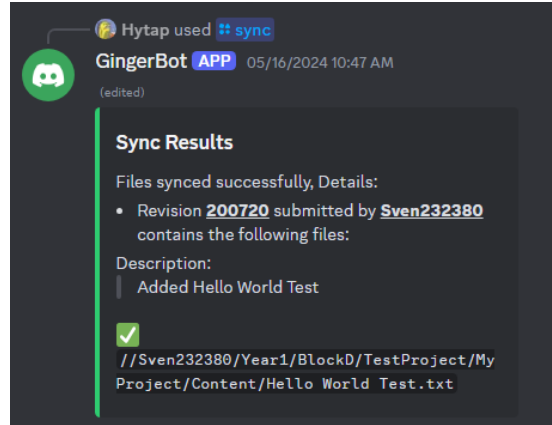
The code that fetches the latest files:

```
def sync_perforce(port, user, client, depotPath):
    p4 = connect_to_perforce(port, user, client)
    errors, warnings, invalid_names = [], [], []
    try:
        logging.info("Connected to Perforce server")
        sync_result = p4.run_sync(depotPath+"/...")
        print(sync_result)
        revisions = {}
        for file in sync_result: ...

        message = "Files synced successfully, Details:\n"
        #sort the revisions based on the rev id
        revisions = dict(sorted(revisions.items(), key=lambda item: item[0]))
        for rev_id, details in revisions.items():...

    return {
        'success': True,
        'errors': errors,
        'warnings': warnings,
        'naming_issues': invalid_names,
        'message': message
    }
```

What happens when I type /sync in Discord:



This shows that one part of my proof of concept is working. The server now fetches the latest files when I type /sync.

Compiling/cooking the game.

Now that I can fetch the latest files, I should compile the game into a playable .exe. By using the Unreal Command line tool I can call an .exe, give it some parameters and it will run the compiler.

The code that calls the exe into compiling the game.

```
# Construct the command with full path to RunUAT.bat
command = [
    str(fullUATPath), "BuildCookRun",
    f"-project={projectPath}", "-noP4",
    "-platform=Win64", "-clientconfig=Development", "-serverconfig=Development",
    "-cook", "-build", "-stage", "-archive", "-verbose", "-DEFINPUTINI=DefaultInput.ini", "INPUTINI=Input.ini",
    map_argument,
    f"-archivedirectory={outputDirectory}"
]

# Start the subprocess and capture output in real time
process = subprocess.Popen(command, cwd=uatScriptPath, stdout=subprocess.PIPE, stderr=subprocess.STDOUT, text=True, bufsize=1)
automationToolTime = ""

# Read output line by line
while True:
    line = process.stdout.readline()
    if "AutomationTool" in line and "executed for" in line:
        automationToolTime = line.split("executed for ")[1].split("\n")[0]

    if not line:
        break
    logging.info(line.strip())
    log_to_file(logFile, line.strip())
```

After a couple of minutes I have a playable exe:

Name ^

- Engine
- Y1BlockD2024
- Manifest_DebugFiles_Win64.txt
- Manifest_NonUFSFiles_Win64.txt
- Manifest_UFSFiles_Win64.txt
- Y1BlockD2024.exe

Zip it

Now that I have a playable .exe, I should zip it so that it can be easily shared.

The code that zips the game:

```
async def zip_folder(folder_path, output_path, version_string):
    # Calculate the number of weeks since the project start date
    start_date = datetime.datetime(2024, 5, 6)
    today_date = datetime.datetime.now()
    week_number = ((today_date - start_date).days // 7) + 1

    # Construct the zip file name based on the week number, team name, and game name
    file_name = f"Week{week_number}_TeamGinger_RaidOnBungelingBay_{version_string}"
    final_output_path = f"{output_path}/{file_name}"

    try:
        # Create the zip archive
        shutil.make_archive(final_output_path, 'zip', folder_path)
        print(f"Successfully zipped {folder_path} into {final_output_path}.zip")
        return {
            "success": True,
            "message": "Zip successful",
            "output": f"{final_output_path}.zip"
        }
    except Exception as e:
        print(f"An error occurred while zipping: {e}")
        return {
            "success": False,
            "message": f"Zip failed: {e}"
        }
```


Ship it

Now that I have a shareable zip, I can use Buttlr to upload it automatically to Itch.io.

The code that calls the Buttlr application and publish it on Itch.io

```
async def upload_file(BUTLER_LOCATION, zip_path, itchLink, channel_name):
    # Command to push the .zip to Itch.io
    command = f"{BUTLER_LOCATION} push {zip_path} {itchLink}:{channel_name}"
    print(command)
    try:
        # Execute the command
        result = subprocess.run(command, shell=True, check=True, text=True)
        print("Upload successful!")
        status = await GetStatus()

        if(status["success"]):
            return{
                "success": True,
                "message": "Upload successful!",
                "version": status["version"]
            }
    
```

Now if you go to the Itch page, the build is there.

TestProject


[More information](#) ✓

Download

Download Now

Name your own price

Click download now to get access to the following files:

testproject-win.zip 446 MB 
Version 32



Combine it

Now that all the functionality is done, I combined it all into 1 single command. If I run /build, it will do all the steps above automatically without me having to do anything else.

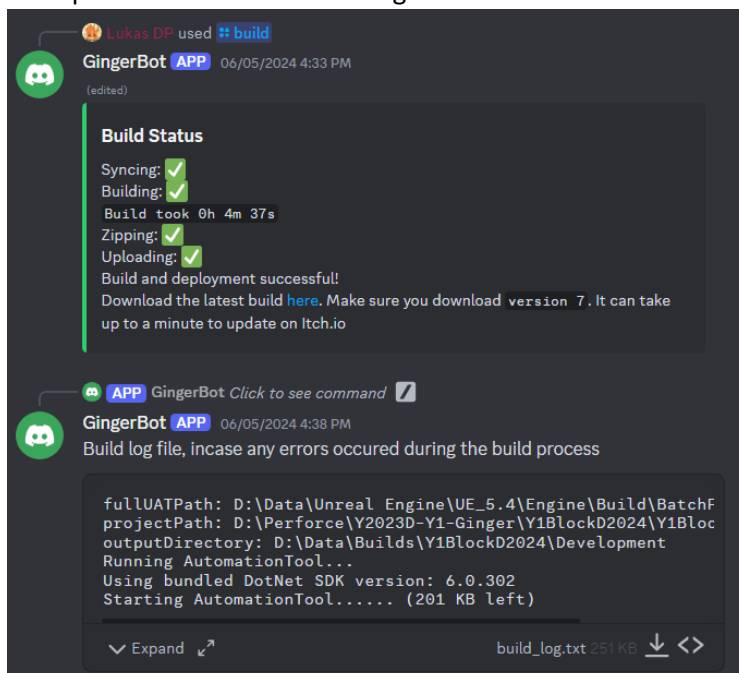
Adopt into the Project

Now that the POC is done, I should implement it into the real project and give my team members the option to use my research project.

While working on the POC I saw that I was getting quite a lot of hardcoded values, like the perforce settings, output location, etc. So during the POC I made a file that stores all the hardcoded value/settings. By having this file I was able to easily port the project over to the teams project settings, without having to change the code.

```
{
  "ProjectName": "Y1BlockD2024",
  "ProjectLocation": "D:/Perforce/Y2023D-Y1-Ginger/Y1BlockD2024",
  "UnrealEngineLocation": "D:/Data/Unreal Engine/UE_5.4",
  "PerforcePort": "ssl:perforce.buas.nl:1666",
  "PerforceUser": "Sven232380",
  "PerforceClient": "Sven232380_Server",
  "ButlerLocation": "C:/Data/Butler/butler.exe",
  "ItchLink": "buas/team-ginger",
  "DownloadLink": "https://buas.itch.io/team-ginger",
  "OutputLocation": "D:/Data/Builds/Y1BlockD2024",
  "DepotPath": "//Y2023D-Y1-Ginger",
  "IgnoreFolderJsonLocation": "C:/Data/CI/IgnoredFolders.json",
  "NamingConvention": "^([A-Z][a-zA-Z0-9]*)$"
}
```

Example of a team member using the bot:



The screenshot shows a Discord chat interface. At the top, a user named 'Lukas DP' has used the command '# build'. Below this, a message from 'GingerBot APP' (timestamped 06/05/2024 4:33 PM) displays the 'Build Status'. The status is as follows:

- Syncing: ✓
- Building: ✓
- Build took 0h 4m 37s
- Zipping: ✓
- Uploading: ✓
- Build and deployment successful!
- Download the latest build [here](#). Make sure you download version 7. It can take up to a minute to update on Itch.io

Below the status message, another message from 'GingerBot APP' (timestamped 06/05/2024 4:38 PM) says 'Build log file, incase any errors occurred during the build process'. This message includes a code block with the following text:

```
fullUATPath: D:\Data\Unreal Engine\UE_5.4\Engine\Build\BatchF
projectPath: D:\Perforce\Y2023D-Y1-Ginger\Y1BlockD2024\Y1Bloc
outputDirectory: D:\Data\Builds\Y1BlockD2024\Development
Running AutomationTool...
Using bundled DotNet SDK version: 6.0.302
Starting AutomationTool..... (201 KB left)
```

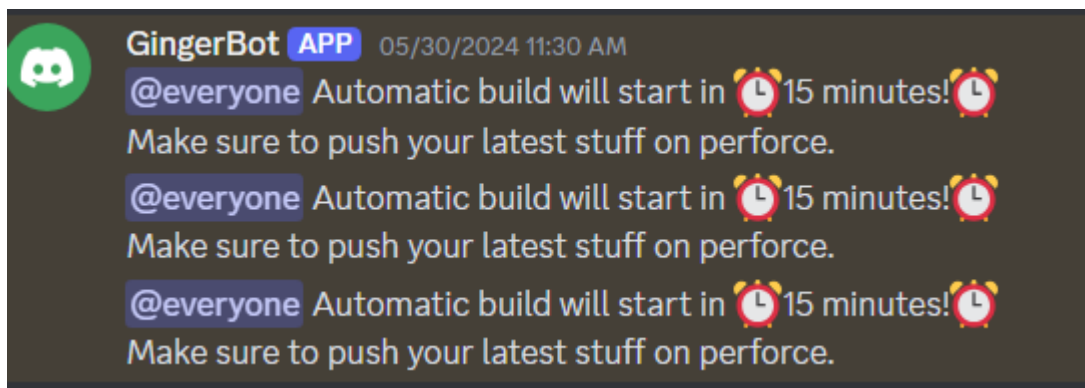
At the bottom of the chat, there is an 'Expand' button with a double arrow icon, and a file attachment 'build_log.txt 251 KB' with download and expand icons.

Feedback

After implementing the bot into the team Discord I got a lot of positive feedback. I also got some feedback on things they would like to see changed. Below are some of the things I added after receiving feedback on it.

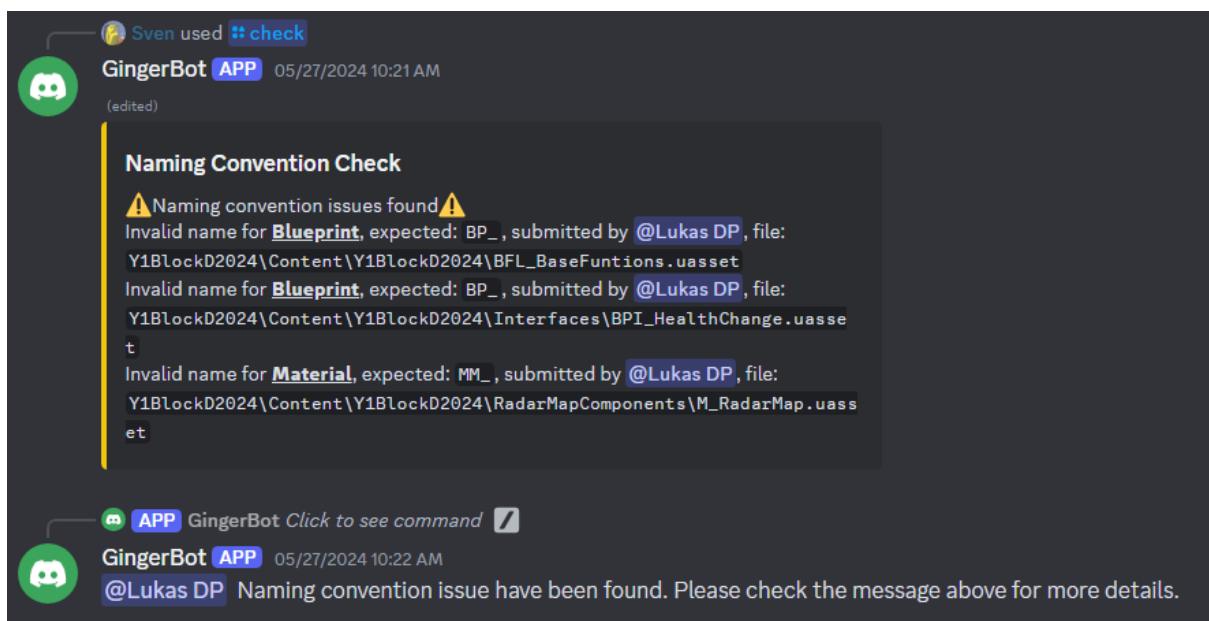
Automatic builds

The teachers requested to have a build every Thursday at 12:00. So a team member suggested automating this. I added a check to the bot for the when it's 11:45 to automatically start a build and let the team member know 15 minutes beforehand that a build is going to start and that they should push their latest stuff.



Naming convention checker

If you call the /check command it will check all the files if they have the correct names. It will also ping the people that submitted it so that they can change and update the name.



Future Improvements

Because of the short time span and me wanting to work on the game, I couldn't add everything for a fully functioning continuous integration. I did implement everything I stated in the [How?](#) Section. If I had more time I would implement/improve the things listed below.

Naming conventions

Although I have a naming convention checker for file names, I couldn't find a way to check for variable names inside the blueprints. Having a variable check would've been handy for keeping the project clean.

Speed

Right now the building times can differ a lot. Sometimes it takes 4 minutes and other time up to 40 minutes. This time difference is based on the stuff added to the project. Whenever a programmer added or changed something using C++, it has to compile everything again, making it cost more time.

In the future I want to optimize this and research how other programs do this.

Build Status

Syncing: ✓

Building: ✓

Build took 0h 42m 14s

Zipping: ✓

Uploading: ✓

Build and deployment successful!

Download the latest build [here](#). Make sure you download **version 4**. It can take up to a minute to update on Itch.io

(edited)

Build Status

Syncing: ✓

Building: ✓

Build took 0h 2m 6s

Zipping: ✓

Uploading: ✓

Build and deployment successful!

Download the latest build [here](#). Make sure you download **version 5**. It can take up to a minute to update on Itch.io

Reflection

Relevance

My choice to focus on the automatic building pipeline was insightful as it remained relevant regardless of the game my team decided to work on. My prior experience in software development and python allowed me to adapt and tailor the pipeline to fit different game specifications and requirements. This adaptability proved invaluable as some of my peers were struggling to find a reason to add their research to the game.

Python

Opting to use Python for this project was a strategic decision due to its extensive range of libraries and tools, which helped various aspects of the build process. Python's capabilities allowed me to automate numerous tedious tasks efficiently, thus speeding up the overall process and reducing the potential for human error. The versatility and robustness of Python provided a solid foundation for implementing complex automation tasks within my pipeline.

Discord

Integrating the build pipeline with Discord was a pivotal move. Eliminating the need for external tools like Jenkins, which might have required additional setup and learning curve for team members. This integration ensured that every team member could stay updated and use the pipeline right from a familiar platform, enhancing usability and accessibility.

Server

Utilizing a dedicated server for the pipeline's operations significantly improved my workflow. It handled builds and tasks without taxing local development machines, allowing team members to continue working on other aspects of the game/project without performance drawbacks. This setup also meant that builds could be executed at any time, enhancing our workflow's flexibility and efficiency.

Sources

Jenkins

<https://www.jenkins.io/>

Travis CI:

<https://www.travis-ci.com/>

CircleCI:

<https://circleci.com/>

GitLab CI:

<https://docs.gitlab.com/ee/ci/>

Reddit:

https://www.reddit.com/r/gamedev/comments/5yv6t7/guide_to_continuous_integration_and_continuous/

Proxmox:

<https://www.proxmox.com/en/>

Windows Server 2019:

<https://www.microsoft.com/en-us/evalcenter/evaluate-windows-server-2019>

P4Python:

<https://www.perforce.com/manuals/p4python/Content/P4Python/python.programming.html>

Unreal Engine UAT:

<https://unrealcontainers.com/docs/use-cases/continuous-integration>

Subprocess Python:

<https://docs.python.org/3/library/subprocess.html>

Shutil:

<https://docs.python.org/3/library/shutil.html>

Butler:

<https://itch.io/docs/butler/>