Low-Level Network Device Interactions

From “Mastering Python Networking - Eric Chou”

Agenda

- Challenges of CLI-managed devices
- Building a safe virtual lab
- Python venv + Pexpect fundamentals
- Paramiko: SSH automation patterns

Why CLI management struggles (1/2)

- Legacy routers/switches were designed for human-driven CLIs
- Output is formatted for people, not programs
- Scaling changes across many devices becomes error-prone

Why CLI management struggles (2/2)

- Engineers must interpret prompts/states manually
- Inconsistent outputs between versions/vendors
- Increased risk with repetitive, manual workflows

Industry shift toward APIs & automation

- Around 2014, momentum built to move from manual CLIs to automation APIs
- Computers excel at repeatability and consistency
- Goal: Reliable, idempotent changes across fleets

Programmatic CLI interactions: the core idea

- Automate the ‘human at a terminal’ pattern
- Watch prompts; send commands; parse responses
- Tools: Pexpect, Paramiko; higher-level: Netmiko, Nornir

Building a lab: why it matters

- Safe sandbox to practice and iterate
- Repeatable scenarios; quick resets
- Lower blast radius while learning

Lab options overview

- Physical devices: Familiarity & realism vs. cost/rigidity
- Virtual devices: Cheap, fast, flexible; occasional feature gaps
- Mix as needed for your context

Physical lab: advantages

- Closer to production feel
- Easier for teams to understand and collaborate
- High comfort level for device handling

Physical lab: disadvantages

- Costly for learning environments
- Rack/stack overhead; limited flexibility
- Slower to iterate topologies

Virtual lab: advantages

- Faster to build and modify
- Lower cost; easier scaling
- Great for proofs of concept and learning

Virtual lab: disadvantages

- Sometimes reduced features vs. hardware
- Performance differences vs. physical gear
- Vendor image licensing considerations

Cisco Modeling Labs (CML)

- Official, widely used simulation platform
- Single download with multiple images (IOSv, IOS-XRv, NX-OSv, ASAv)
- API access, HTML UI; DevNet hosted option is sometimes available

CML tips

- Use provided topologies; adjust management IPs to your lab schema
- Import lab images; leverage dashboard views, multiuser grouping
- Integrations: Ansible, pyATS

Cisco DevNet resources

- Free sign-up; sandboxes (always-on/reservable)
- Guided tracks, docs, and examples for network automation
- Certification paths from associate to expert

Other virtual lab options

- GNS3: mature, vendor-neutral; GUI for topologies
- EVE-NG: strong multi-vendor emulation
- containerlab: container-based network emulation

Standalone virtual platforms

- Arista vEOS, Juniper vMX, Nokia SR-Linux, VyOS
- Great for platform-specific feature tests
- Cloud marketplace images for quick access

Keep topologies simple for learning

- Small node counts ≥ clarity
- Reuse topologies across multiple labs
- Document IP schemes and changes

Python virtual environments (venv)

- Isolate dependencies per project
- Avoid conflicts with global packages
- Typical workflow: install → create → activate → deactivate

venv quickstart

- python3 -m venv venv #create a virtual environment
- source venv/bin/activate #activate a virtual environment
- pip install necessary_libraries #install libraries to the separated virtual environment
- …#other commands
- deactivate #when done

Pexpect: what it is

- Pure-Python Expect-like library for automating interactive apps
- Spawn a child process; expect prompts; send commands
- **Great for Telnet/SSH CLI automation patterns**

Installing & testing Pexpect

- `pip install pexpect`
- Import and verify in Python REPL
- **Works on Linux/macOS;** Windows support is improving

Installing & testing Pexpect

- `python3 -m venv .venv`
- `source .venv/bin/activate`
- `pip install --upgrade pip`
- `pip install pexpect`

Creat a separate environment and install pexpect to it

- Type `python` or `python3` in your terminal to start.
- Then type:

```
import pexpect
pexpect.__version__
```

- Remember to deactivate when finish

Pexpect core methods

- spawn(): start a child app (e.g., telnet device)
- expect(): wait for a prompt/pattern
- sendline(): send a command (with newline)

Matching prompts & output

- Use exact host prompts (e.g., 'lax-edg-r1#’, ‘csr1kv#’)
- Use regex for variations: '[Uu]sername'
- Access .before and .after for captured text

Handling errors & timeouts

- Set expect(..., timeout=seconds) for slow links
- Log sessions: child.logfile = open('debug','wb')
- Use interact() to hand control back to a human

Pexpect pxssh for SSH

- Simplifies SSH login/logout sequences
- login(host, user, pass, auto_prompt_reset=False) for network gear

Pexpect program #1: multi-device, single command

- Device dict: {'prompt': ..., 'ip': ...} per host
- Loop: login → show version → capture → exit
- Print outputs for quick verification

Pexpect program #1: multi-device, single command

```
#!/usr/bin/env python
import pexpect
devices = {'iosv-1': {'prompt': 'R1>', 'ip': '192.168.68.118'}} #change the prompt and ip as needed
username = 'admin'
password = 'admin'
for device in devices.keys():
    device_prompt = devices[device]['prompt']
    child = pexpect.spawn('telnet ' + devices[device]['ip'])
    # child.expect('Username:')
    # child.sendline(username)
    # child.expect('Password:')
    child.sendline(password)
    child.expect(device_prompt)
    child.sendline('show version | i V')
    child.expect(device_prompt)
    print(child.before)
    child.sendline('exit')
```

Pexpect program #1: multi-device, single command

Welcome | pexpect-demo.py ●

pexpect-demo.py > ...

```
3   import pexpect
4
5   devices = {'iosv-1': {'prompt': 'R1>', 'ip': '192.168.68.118'}}
6   username = 'admin'
7   password = 'admin'
8
9   for device in devices.keys():
10      device_prompt = devices[device]['prompt']
11      child = pexpect.spawn('telnet ' + devices[device]['ip'])
12      # child.expect('Username:')
13      # child.sendline(username)SS
14      child.expect('Password:')
15      child.sendline(password)
16      child.expect(device_prompt)
17      child.sendline('show version | i V')
18      child.expect(device_prompt)
19      print(child.before)
20      child.sendline('exit')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: Python

```
Connection closed by foreign host.
devasc@labvm:~/labs/Demo$ python3 /home/devasc/labs/Demo/pexpect-demo.py
b'show version | i V\r\nCisco IOS XE Software, Version 16.09.05\r\nCisco IOS Software [Fuji], Virtual XE Software (X86_64_LINUX_IOSD-UNIVERSALK9-M),
Version 16.9.5, RELEASE SOFTWARE (fc1)\r\nlicensed under the GNU General Public License ("GPL") Version 2.0.  The\r\nsoftware code licensed under GPL
 Version 2.0 is free software that comes\r\nGPL code under the terms of GPL Version 2.0.  For more details, see the\r\ncisco CSR1000V (VXE) processor
 (revision VXE) with 2182252K/3075K bytes of memory.\r\n'
devasc@labvm:~/labs/Demo$
```

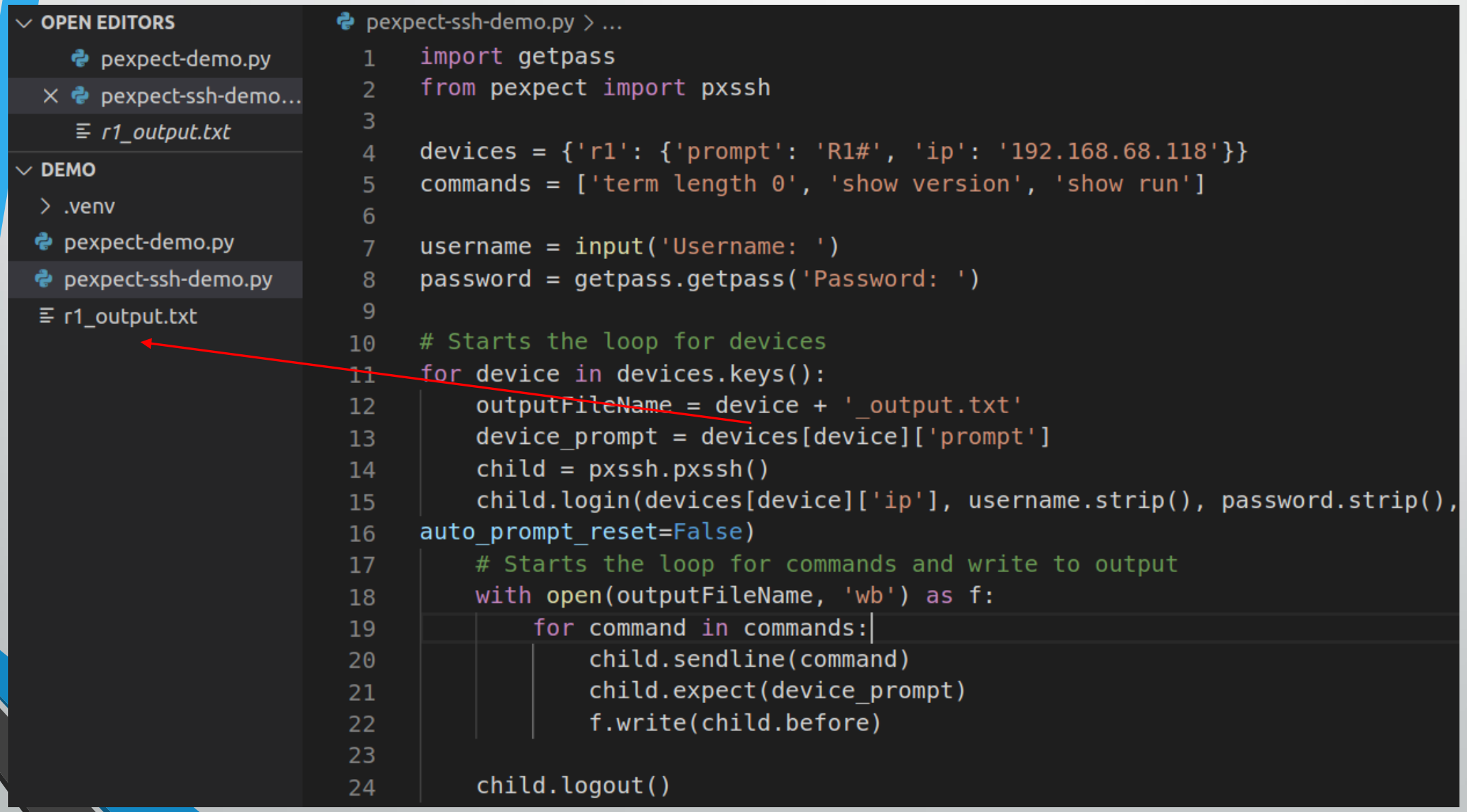
Pexpect program #2: SSH + multi-commands

- Use pxssh + list of commands
- Prompt for credentials via getpass
- Write outputs per device to files
- **Enable and check ssh login before running**

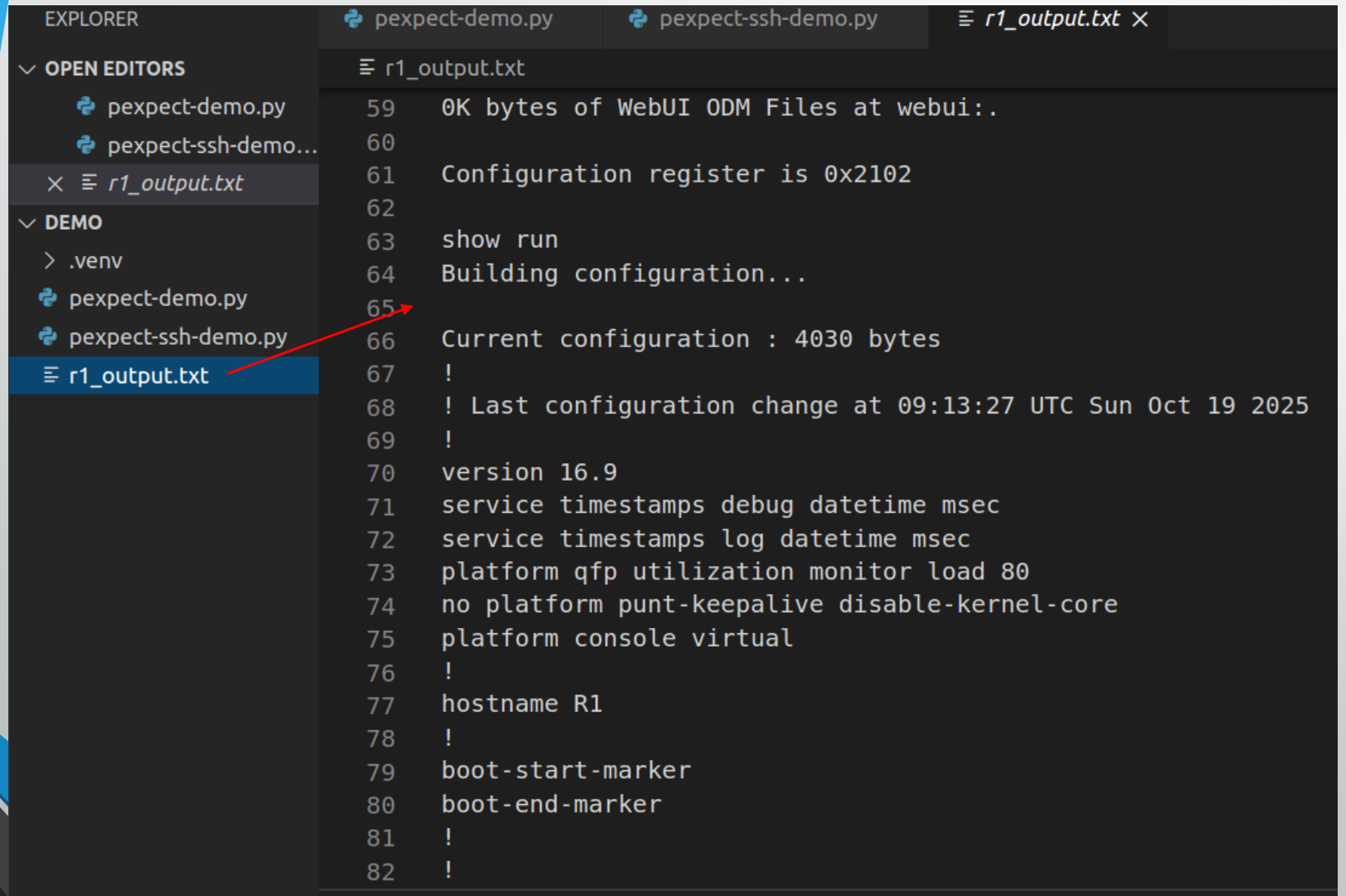
Pexpect program #2: SSH + multi-commands

```
import getpass
from pexpect import pxssh

devices = {'r1': {'prompt': 'R1#', 'ip': '192.168.68.118'}}
commands = ['term length 0', 'show version', 'show run']

username = input('Username: ')
password = getpass.getpass('Password: ')

# Starts the loop for devices
for device in devices.keys():
    outputFileName = device + '_output.txt'
    device_prompt = devices[device]['prompt']
    child = pxssh.pxssh()
    child.login(devices[device]['ip'], username.strip(), password.strip(), auto_prompt_reset=False)
    # Starts the loop for commands and write to output
    with open(outputFileName, 'wb') as f:
        for command in commands:
            child.sendline(command)
            child.expect(device_prompt)
            f.write(child.before)
    child.logout()
```

Pexpect program #2: SSH + multi-commands



Pexpect program #2: SSH + multi-commands



Paramiko: what it is

- Python SSHv2 client (and server) implementation
- Focuses on SSH; no Telnet
- Foundation for many tools (e.g., Ansible transport)

Paramiko: what it is

- Allow:
  - Connect to remote hosts securely
  - Run commands
  - Transfer files (via SFTP)
- Two main modes of interaction:
  - exec_command() → Non-interactive
  - invoke_shell() → Interactive

Installing Paramiko

We will show the Paramiko installation steps for our Ubuntu 22.04 virtual machine:

```
sudo apt-get install build-essential libssl-dev libffi-dev python3-dev
pip install cryptography
pip install paramiko
```

Paramiko: invoke_shell()

- Opens a persistent, interactive session (like a live SSH terminal)
- **Use .send() / .recv()** to send commands and read output
- Ideal for network devices (e.g., Cisco IOS, JunOS) needing multiple CLI commands
- Supports mode transitions and prompt-based workflows

Paramiko: invoke_shell()

```
shell = connection.invoke_shell()
shell.send("terminal length 0\n")
shell.send("show ip int brief\n")
output = shell.recv(5000).decode() # Read up to 5000 bytes from the
```

remote SSH channel’s receive buffer

Paramiko: exec_command()

- Runs a **single**, non-interactive command per call
- Opens a new channel for each command
- Best for **servers** (Linux, UNIX)
- On some network gear, session may end after one command
- Example: (next slide)

Paramiko: exec_command()

```
stdin, stdout, stderr = ssh.exec_command(“ls -l")
print(stdout.read().decode())
```

Paramiko host keys & policies

- AutoAddPolicy(): add unknown host keys automatically
- Alternative: load_system_host_keys() for stricter security
- Be explicit about look_for_keys / allow_agent

Buffer discipline with Paramiko

- Always read from recv() to drain buffers
- Use helper (function) to clear buffer before next command
- Avoid stale output contaminating results

Key-based SSH for servers

- Generate keypair (ssh-keygen)
- Copy public key to authorized_keys
- Paramiko: RSAKey.from_private_key_file + exec_command()

Paramiko example: invoke_shell()

- Keep interactive shell open; send commands sequentially
- Use sleeps judiciously for busy/slow endpoints
- Close cleanly when done

Paramiko invoke_shell() example program structure

- Define devices, commands; connect and open shell
- terminal length 0; clear buffer; run commands
- Write outputs to per-device files

```python
#!/usr/bin/env python
import paramiko, getpass, time
devices = {'csr1kv': {'ip': '192.168.68.117'}} # change the prompt and ip accordingly
commands = ['show version\n', 'show run\n']
username = input('Username: ')
password = getpass.getpass('Password: ')
max_buffer = 65535
def clear_buffer(connection):
    if connection.recv_ready():
        return connection.recv(max_buffer)
# Starts the loop for devices
for device in devices.keys():
    outputFileName = device + '_output.txt'
    connection = paramiko.SSHClient()
    connection.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    connection.connect(devices[device]['ip'], username=username, password=password, look_for_keys=False, allow_agent=False)
    new_connection = connection.invoke_shell()
    output = clear_buffer(new_connection)
    time.sleep(5)
    new_connection.send("terminal length 0\n")
    output = clear_buffer(new_connection)
    with open(outputFileName, 'wb') as f:
        for command in commands:
            new_connection.send(command)
            time.sleep(5)
            output = new_connection.recv(max_buffer)
            print(output)
            f.write(output)

    new_connection.close()
```

Paramiko example: exec_command()

```
import paramiko
key = paramiko.RSAKey.from_private_key_file('/home/echou/.ssh/id_rsa')
client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect('192.168.199.182', username='echou', pkey=key)
stdin, stdout, stderr = client.exec_command('ls -l')
stdout.read()
```

Externalizing inventory & commands

- devices.json for hosts; commands.txt for actions
- Avoid hard-coding; reduce edit errors
- Simple JSON/YAML → Python dicts/lists

Externalizing inventory & commands

```
$ cat commands.txt
config t
logging buffered 30000
end
copy run start
```

+

```
$ cat devices.json
{
    "lax-edg-r1": {
        "ip": "192.168.2.51"
    },
    "lax-edg-r2": {
        "ip": "192.168.2.52"
    }
}
```

```
with open('devices.json', 'r') as f:
    devices = json.load(f)
with open('commands.txt', 'r') as f:
    commands = f.readlines()
```

Externalizing inventory & commands

```
#!/usr/bin/env python

import paramiko, getpass, time, json

with open('devices.json', 'r') as f:
    devices = json.load(f)

with open('commands.txt', 'r') as f:
    commands = f.readlines()

username = input('Username: ')
password = getpass.getpass('Password: ')

max_buffer = 65535

def clear_buffer(connection):
    if connection.recv_ready():
        return connection.recv(max_buffer)
```

```
# Starts the loop for devices
for device in devices.keys():
    outputFileName = device + '_output.txt'
    connection = paramiko.SSHClient()
    connection.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    connection.connect(devices[device]['ip'], username=username, password=password, look_for_keys=False, allow_agent=False)
    new_connection = connection.invoke_shell()
    output = clear_buffer(new_connection)
    time.sleep(2)
    new_connection.send("terminal length 0\n")
    output = clear_buffer(new_connection)
    with open(outputFileName, 'wb') as f:
        for command in commands:
            new_connection.send(command)
            time.sleep(2)
            output = new_connection.recv(max_buffer)
            print(output)
            f.write(output)

    new_connection.close()
```