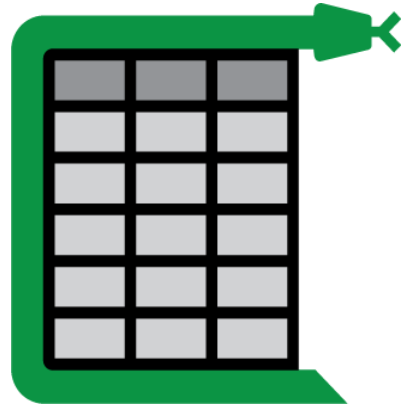
The Hitchhiker's Guide to CLIs in Python

Vinayak Mehta

@vortex_ape

$ whoami

https://github.com/vinayak-mehta

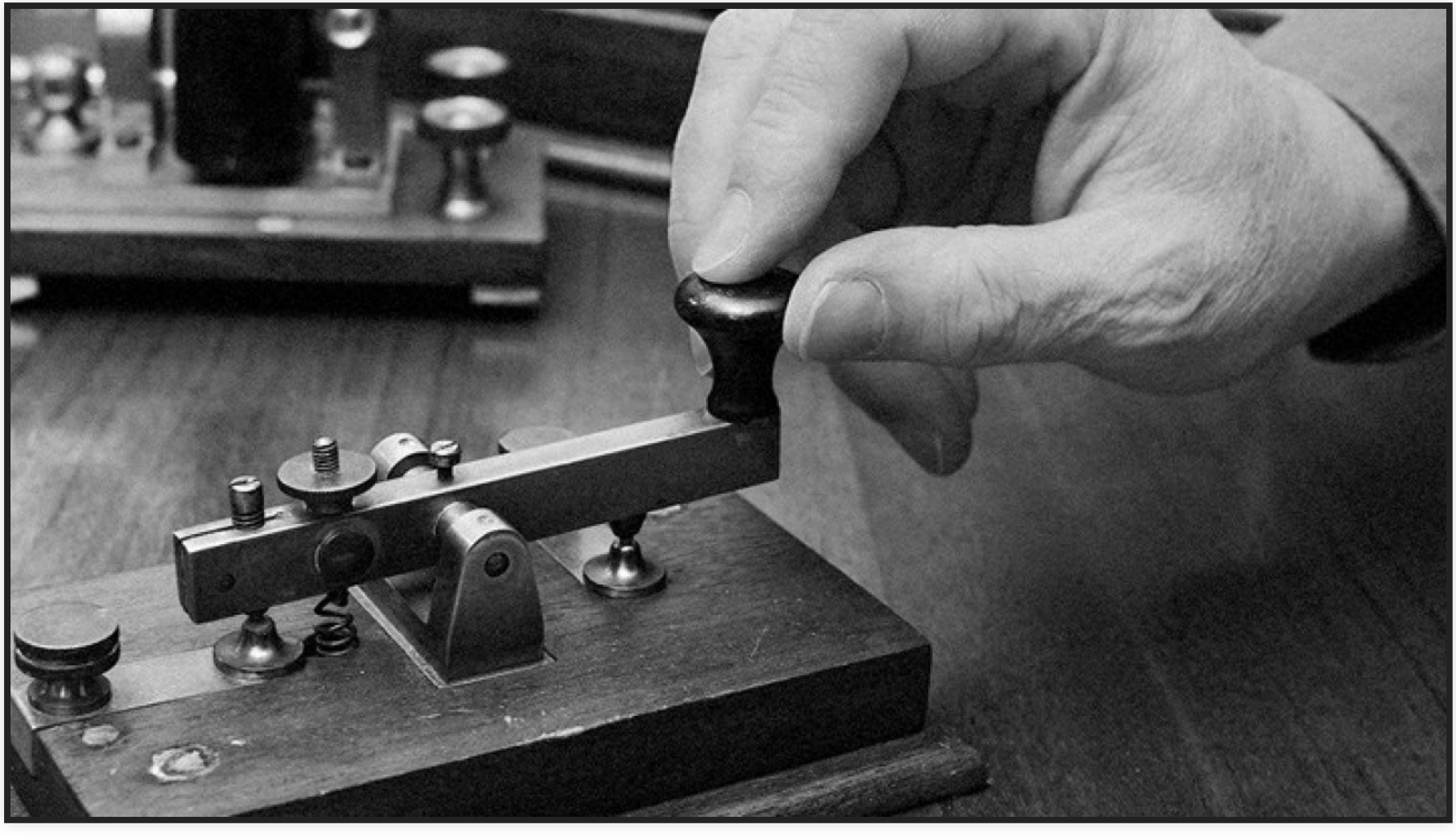


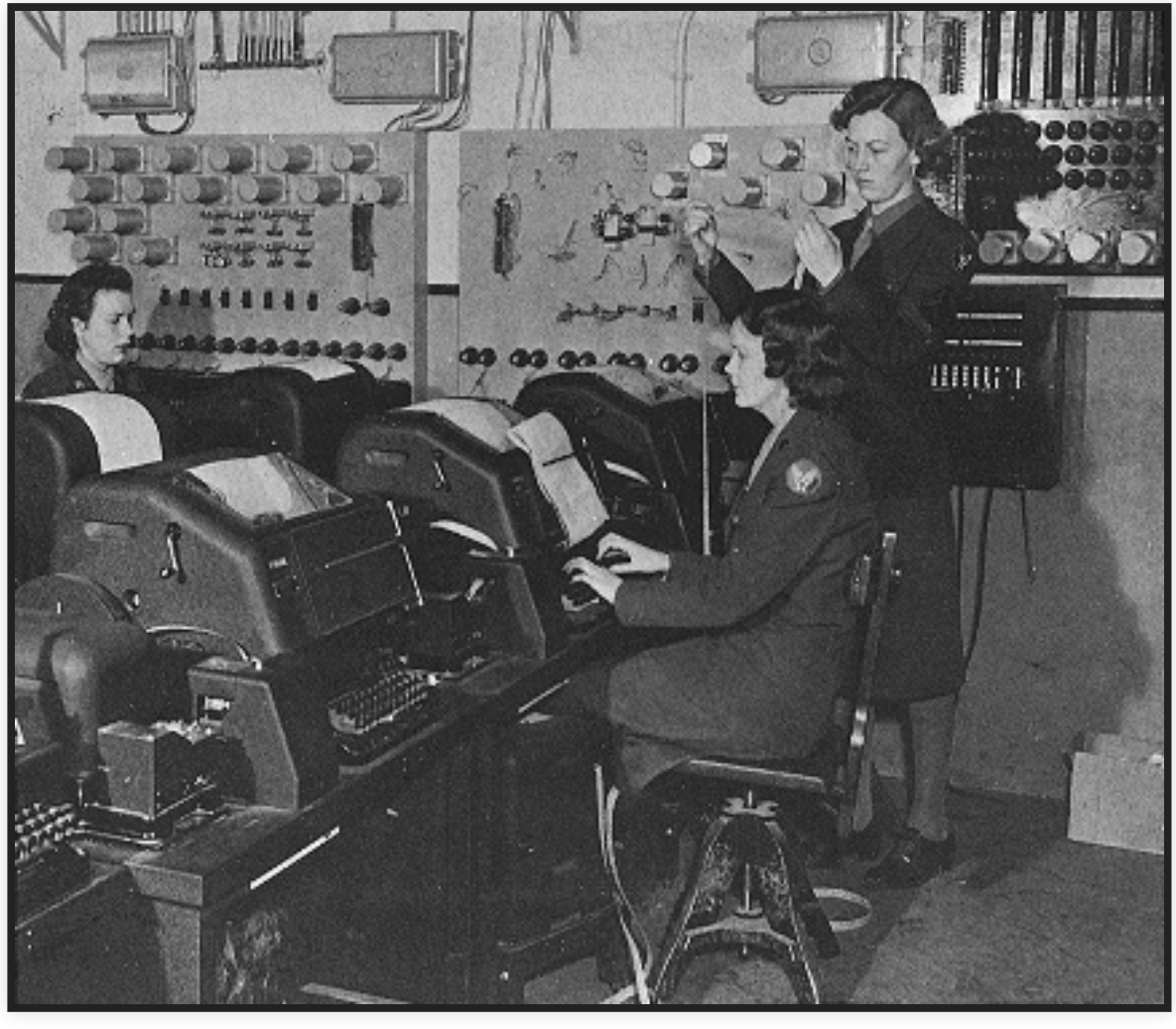
https://github.com/camelot-dev

https://www.recurse.com

In the beginning ...


underwood
FIVE



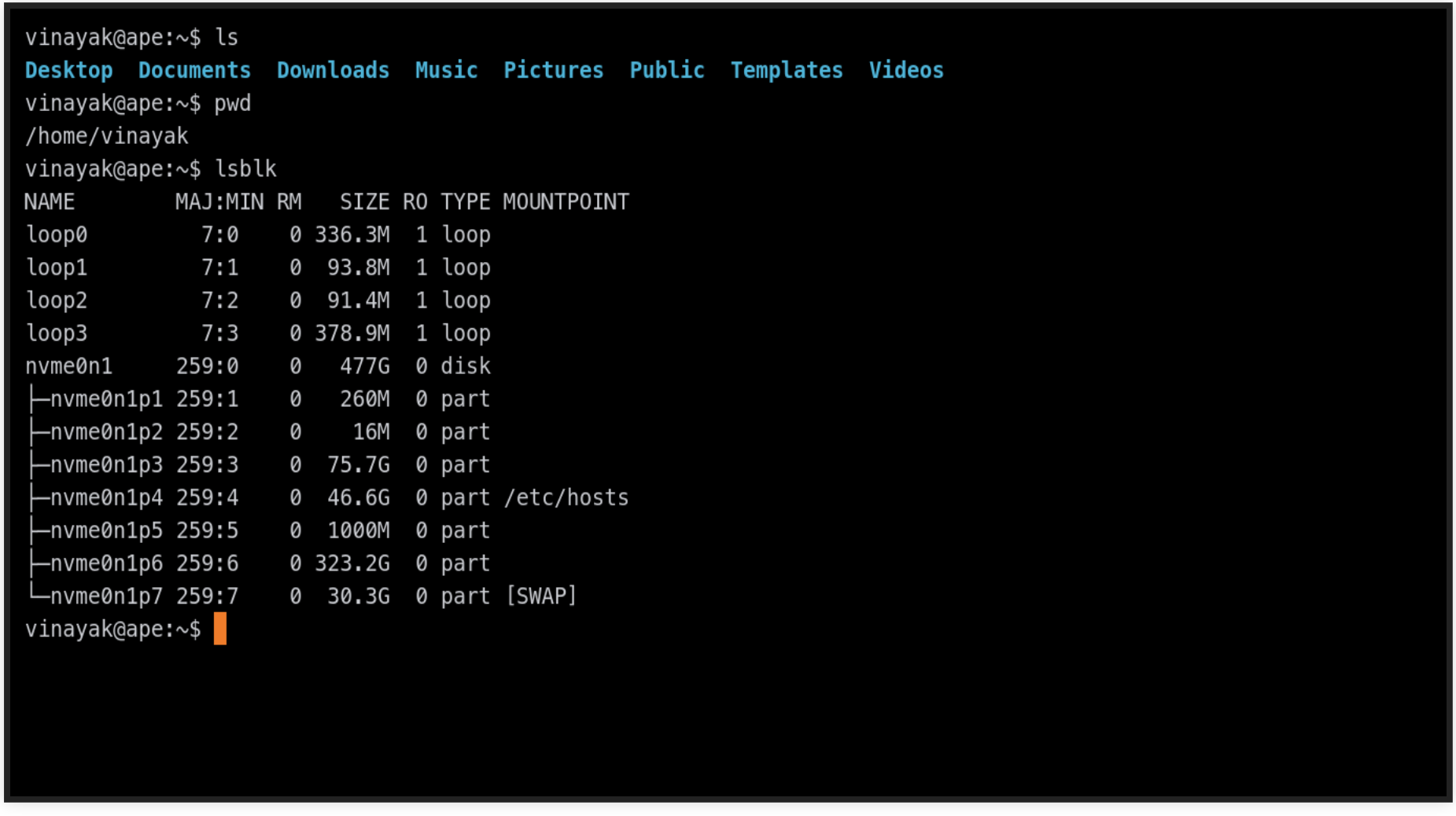


https://www.youtube.com/watch?v=n-eFFd5BmpU


Friden
flexowriter
START READ
STOP READ
NON PRINT
TAPE SKIP
PUNCH ADRES
PUNCH
ON
1


TimeShare
Corporation
START
STOP
FREE


digital VT100

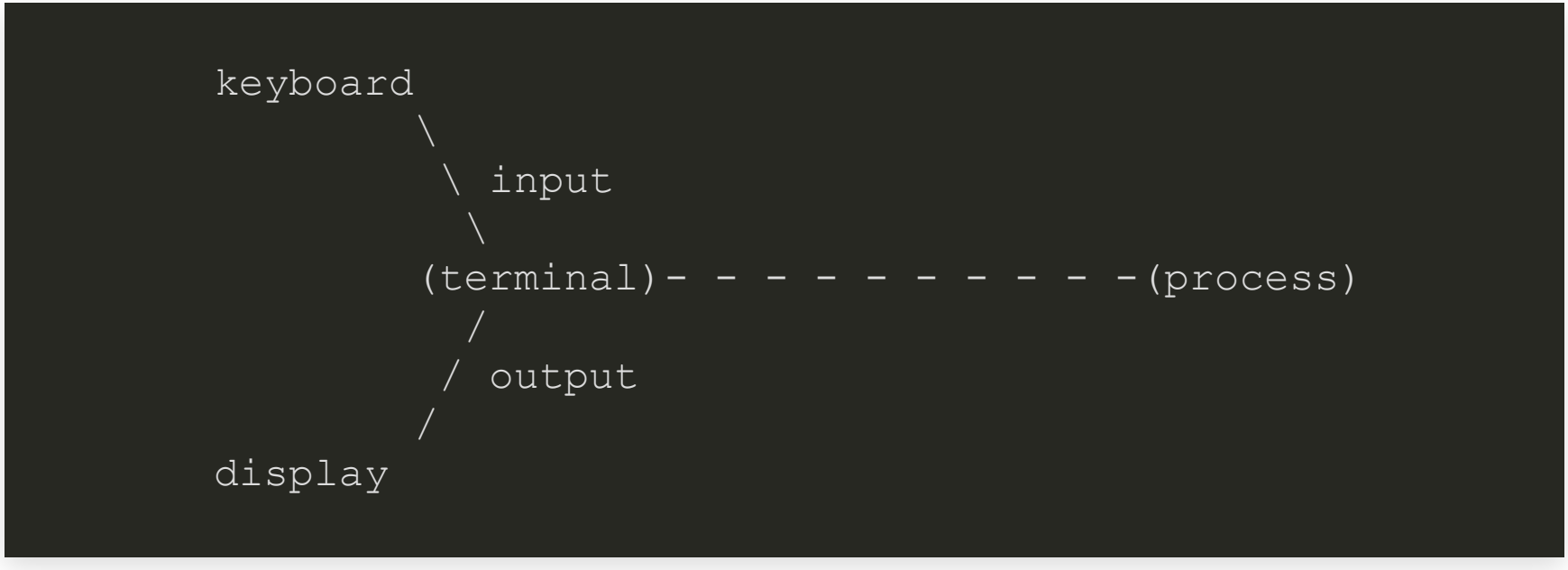

vinayak@ape:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
vinayak@ape:~$ pwd
/home/vinayak
vinayak@ape:~$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
loop0 7:0 0 336.3M 1 loop
loop1 7:1 0 93.8M 1 loop
loop2 7:2 0 91.4M 1 loop
loop3 7:3 0 378.9M 1 loop
nvme0n1 259:0 0 477G 0 disk
├─nvme0n1p1 259:1 0 260M 0 part
├─nvme0n1p2 259:2 0 16M 0 part
├─nvme0n1p3 259:3 0 75.7G 0 part
├─nvme0n1p4 259:4 0 46.6G 0 part /etc/hosts
├─nvme0n1p5 259:5 0 1000M 0 part
├─nvme0n1p6 259:6 0 323.2G 0 part
└─nvme0n1p7 259:7 0 30.3G 0 part [SWAP]
vinayak@ape:~$

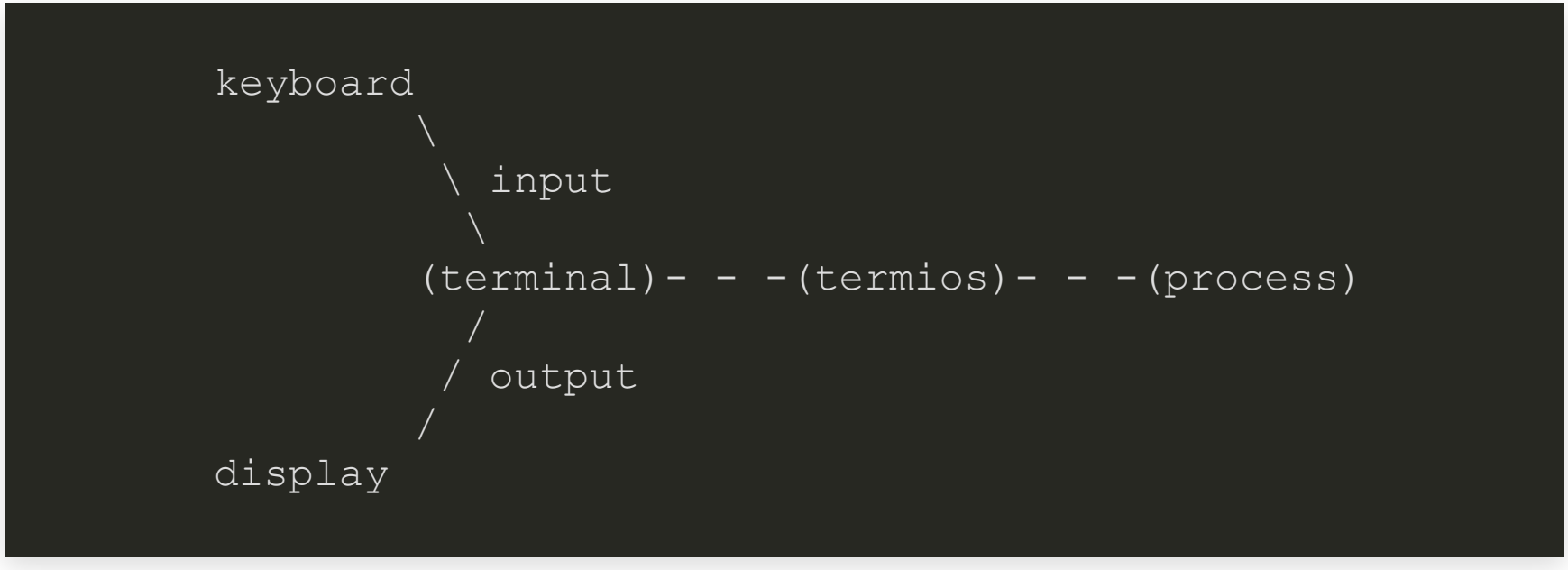
teletype

(t)ele(ty)pe

tty

shell


keyboard
\
\ input
\
(terminal)- - - - - - - - - -(process)
/
/ output
/
display

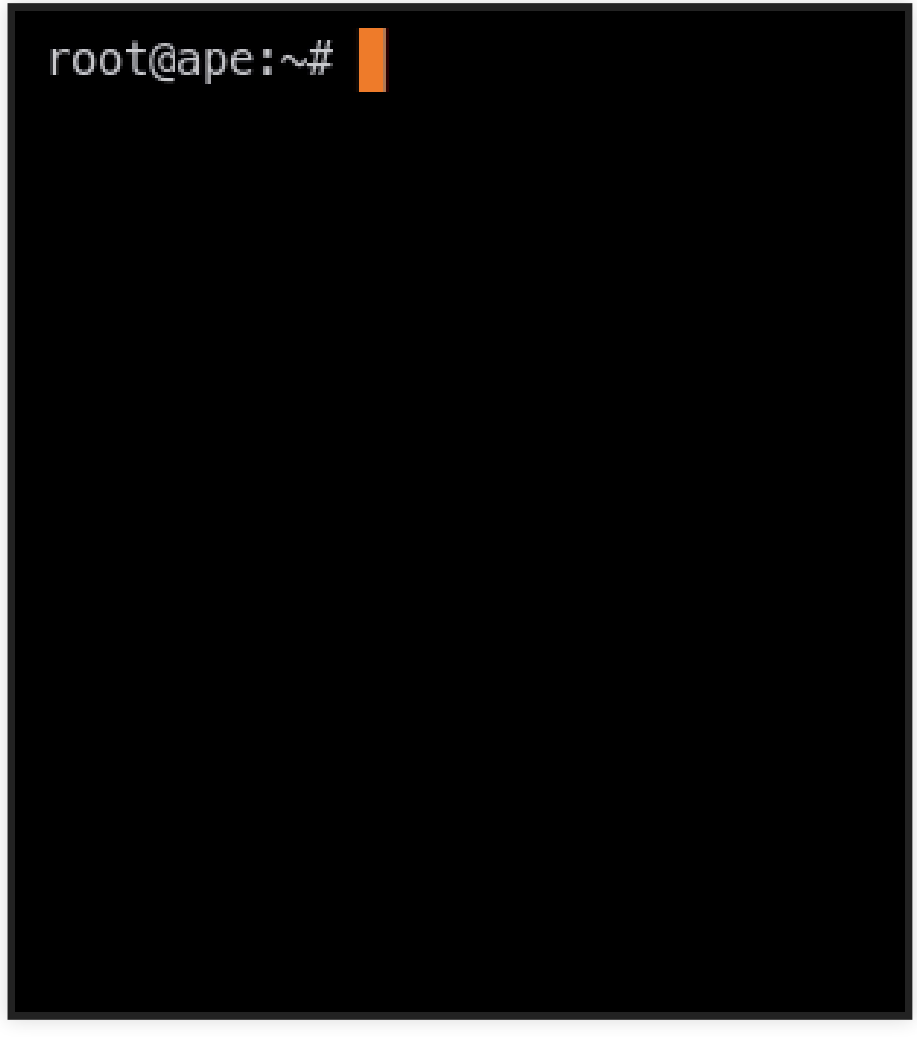

keyboard
\
\ input
\
(terminal)- - -(termios)- - -(process)
/
/ output
/
display

```
$ man termios
```

```
$ stty -a
speed 38400 baud; rows 34; columns 166; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 c
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -
```

```
$ man termios
...
    ICANON Enable canonical mode (described below).
...
```

```
$ stty -icanon
```


root@ape:~#

```
$ man termios
...
    ONLCR  (XSI) Map NL to CR-NL on output.
...
```

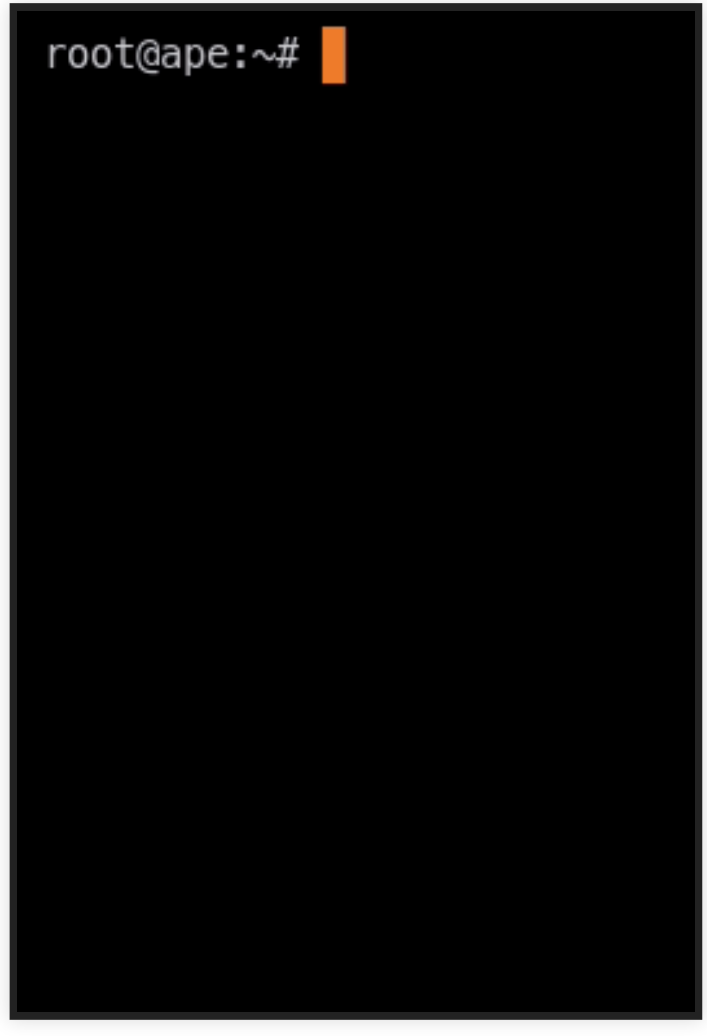

vinayak@ape
smol-git
~/dev
smol-git push origin master
Username for 'https://github.com': vinayak
Password for 'https://vinayak@github.com':

```
$ stty -onlcr
```


root@ape:~#

```
$ man termios
...
    ECHO   Echo input characters.
...
```

```
$ stty -echo
```


root@ape:~#

```
$ reset
```

```
import termios
```

Signals

In-band signaling

Control characters

Control characters

- **^H** backspace
- **^J** newline
- **^C** interrupt the running process
- **^D** end text input or exit the shell

Escape sequences

Escape sequences

- `\u001b[2J`: clear screen
- `\u001b[1m`: make text bold
- `\u001b[31m`: make text red
- `\u001b[{n}A`: moves cursor up by n

Streams

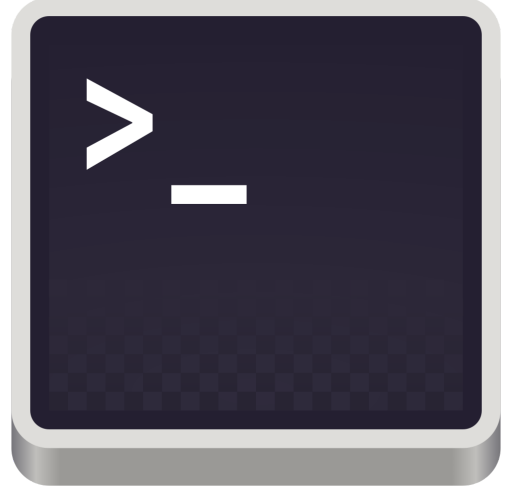
stdin

stdout and stderr

Redirection

```
$ echo "hello" > file
$ echo "world" >> file
```

```
$ echo "hello" | cat
hello
```



Command-line interfaces

Command-line interfaces

```
Prompt
```

Command-line interfaces

```
Prompt command
```

Command-line interfaces

```
Prompt command option1 option2
```

Command-line interfaces

```
Prompt command option1 option2 argument1 argument2 <Enter>
```

Command-line interfaces

```
Prompt command option1 option2 argument1 argument2 <Enter>
Output
```

Arguments

Arguments

```
$ cp src dst
```

Options

Options

```
$ cp -r src dst
```

Help

Help

```
$ cp --help
```

Man pages

```
$ man termios
```

Standards

POSIX

XDG base directory specification

XDG base directory specification

- `$XDG_CONFIG_HOME=$HOME/.config`
- `$XDG_DATA_HOME=$HOME/.local/share`
- `$XDG_CACHE_HOME=$HOME/.cache`

CLIs in Python

smol-pip

```
$ smol-pip install --upgrade package_name
```

Standard library

sys

sys.argv

getopt

```
import sys

help = "Pip Installs Packages."

if __name__ == "__main__":
    arguments = sys.argv
```

```python
import sys

help = "Pip Installs Packages."

if __name__ == "__main__":
    arguments = sys.argv
    if arguments[1] in ["-h", "--help"]:
        print(help)
```

```python
import sys

help = "Pip Installs Packages."

if __name__ == "__main__":
    arguments = sys.argv
    if arguments[1] in ["-h", "--help"]:
        print(help)
    elif arguments[1] in ["-v", "--version"]:
        print("0.1.0")
```

```python
import sys

help = "Pip Installs Packages."

if __name__ == "__main__":
    arguments = sys.argv
    ...
    else:
        print(arguments)
        # ['smol-pip', 'install', '--upgrade', 'Click']
        if arguments[1] == "install":
            # dispatch to install / upgrade code
        else:
            raise ValueError("Unknown subcommand!")
```

optparse

PEP 389

argparse

argparse

- `-pf`
- `-file`
- `+f`
- `+rgb`
- `/f`
- `/file`

argparse

- `pip install`
- `pip freeze`
- `pip search`

```
import argparse

parser = argparse.ArgumentParser(
    description="Pip Installs Packages."
)
```

```python
import argparse

parser = argparse.ArgumentParser(
    description="Pip Installs Packages."
)
parser.add_argument(
    "-v",
    "--version",
    action="version",
    version="0.1.0"
)
```

```
subparsers = parser.add_subparsers(dest="subparser_name")
install = subparsers.add_parser("install")
```

```
subparsers = parser.add_subparsers(dest="subparser_name")
install = subparsers.add_parser("install")
install.add_argument(
    "-u",
    "--upgrade",
    action="store_true",
    help="Upgrade package to the newest available version.",
)
install.add_argument("package_name")
```

```
if __name__ == "__main__":
    arguments = parser.parse_args()
    print(arguments)
    # Namespace(package_name='Click', upgrade=True)
```

```python
if __name__ == "__main__":
    arguments = parser.parse_args()
    print(arguments)
    # Namespace(package_name='Click', upgrade=True)
    if arguments.subparser_name == "install":
        # dispatch to install / upgrade code
    else:
        raise ValueError("Unknown subcommand!")
```

```
$ smol-pip --help
usage: smol-pip [-h] [-v] {install} ...

Pip Installs Packages.

positional arguments:
  {install}

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit
```

Python Package Index

docopt

```
help = """Pip Installs Packages.

Usage:
  smol-pip install PACKAGE_NAME
  smol-pip install --upgrade PACKAGE_NAME

Options:
  -h --help     Show this screen.
  --version     Show version.
"""
```

```python
from docopt import docopt

if __name__ == "__main__":
    arguments = docopt(help, version="0.1.0")
    print(arguments)
    # {'--upgrade': True,
    #  'PACKAGE_NAME': 'Click',
    #  'install': True}
```

```python
from docopt import docopt

if __name__ == "__main__":
    arguments = docopt(help, version="0.1.0")
    print(arguments)
    # {'--upgrade': True,
    #  'PACKAGE_NAME': 'Click',
    #  'install': True}
    if arguments["install"]:
        # dispatch to install / upgrade code
    else:
        raise ValueError("Unknown subcommand!")
```

click

```python
import click

def cli(*args, **kwargs):
    """Pip Installs Packages."""
    pass
```

```python
import click

@click.group("pip")
def cli(*args, **kwargs):
    """Pip Installs Packages."""
    pass
```

```python
import click

@click.group("pip")
@click.version_option("0.1.0")
def cli(*args, **kwargs):
    """Pip Installs Packages."""
    pass
```

```
def install(*args, **kwargs):
    """Install packages."""
    # install / upgrade package_name
```

```
@cli.command("install")
def install(*args, **kwargs):
    """Install packages."""
    # install / upgrade package_name
```

```python
@cli.command("install")
@click.option(
    "-u",
    "--upgrade",
    is_flag=True,
    help="Upgrade package to the newest available version.",
)
def install(*args, **kwargs):
    """Install packages."""
    # install / upgrade package_name
```

```python
@cli.command("install")
@click.option(
    "-u",
    "--upgrade",
    is_flag=True,
    help="Upgrade package to the newest available version.",
)
@click.argument("package_name")
def install(*args, **kwargs):
    """Install packages."""
    # install / upgrade package_name
```

```python
if __name__ == "__main__":
    cli()
```

```python
@cli.command("install")
@click.option(
    "-u",
    "--upgrade",
    is_flag=True,
    help="Upgrade package to the newest available version.",
)
@click.argument("package_name")
def install(*args, **kwargs):
    """Install packages."""
    print(kwargs)
    # {'upgrade': True, 'package_name': 'Click'}
    # install / upgrade package_name
```

```
$ smol-pip --help
Usage: smol-pip [OPTIONS] COMMAND [ARGS]...

  Pip Installs Packages.

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  install  Install packages.
```

click

smol-git

```
$ smol-git --help
Usage: smol-git [OPTIONS] COMMAND [ARGS]...

  smol-git - the stupid content tracker

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  clone   Clone a repository into a new directory.
  commit  Record changes to the repository.
  config  Get and set repository or global options.
  log     Show commit logs.
```

```python
import click

@click.group("smol-git")
@click.version_option("0.1.0")
def cli(*args, **kwargs):
    """smol-git - the stupid content tracker"""
    pass
```

Progress bars

```python
@cli.command()
@click.argument("src")
@click.argument("dest", required=False)
def clone(src, dest):
  ...
  with click.progressbar(files) as _files:
      for file in _files:
          # download file
```


vinayak@ape
smol-git
~

Application folders

```python
@cli.command()
@click.argument("key")
@click.argument("value")
def config(key, value):
    app_dir = click.get_app_dir("smol_git")
    if not os.path.exists(app_dir):
        os.makedirs(app_dir)
    cfg = os.path.join(app_dir, "config")
    # set repository or global options
```


vinayak@ape
smol-git
~/smol-git

Paged output

```
@cli.command()
def log():
    ...
    click.echo_via_pager(log_string)
```


vinayak@ape
smol-git
~/smol-git

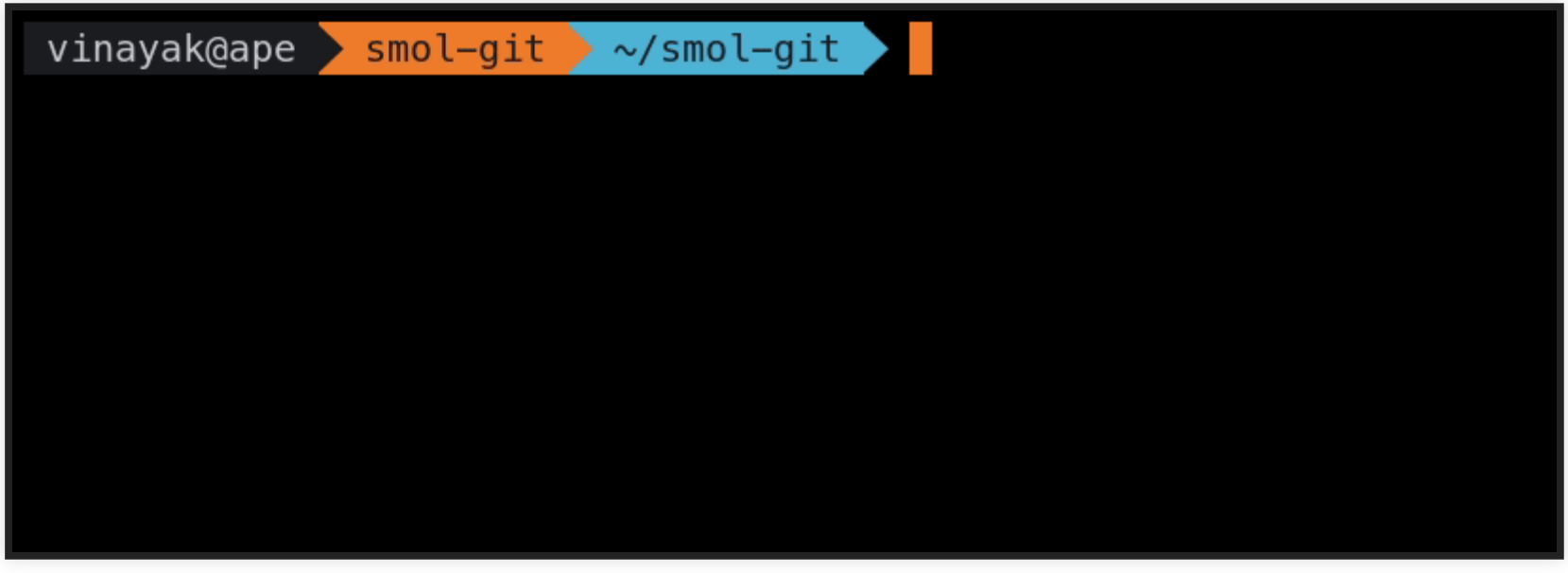
Colored text

```python
@cli.command()
def status():
    ...
    for file in files:
        file_status = "new file" if file.added else "modifie
        status += click.style(
            f"\t{file_status}:   {file.name}\n",
            fg="green",
            bold=True
        )
    click.echo(status_string)
```


vinayak@ape
smol-git
~/smol-git

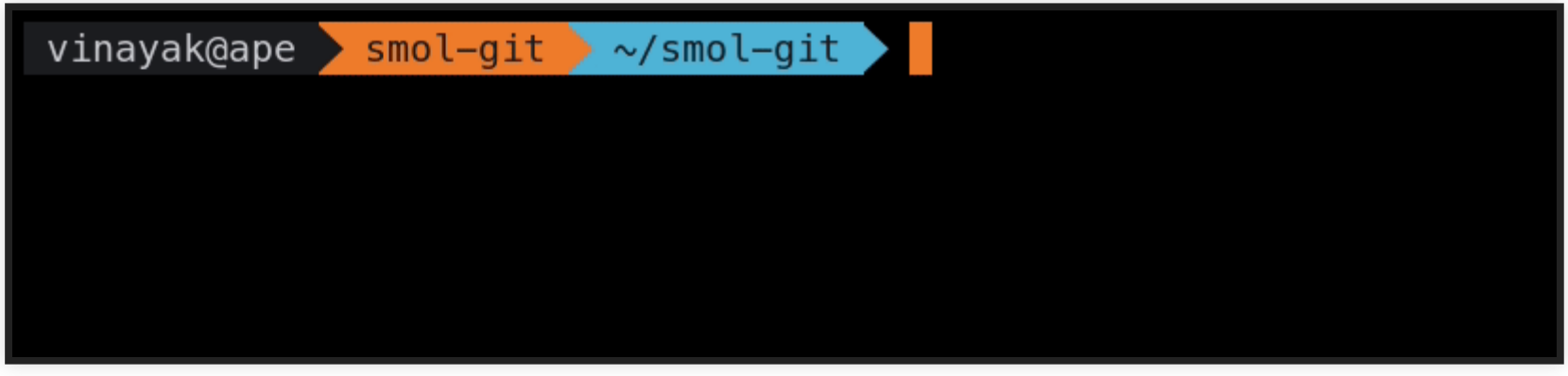
Launching editors

```python
@cli.command()
@click.option("-m", "--message", help="The commit message.")
def commit(*args, **kwargs):
    if kwargs["message"] is None:
        commit_message = click.edit()
    else:
        commit_message = kwargs["message"]
    # commit changes
```


vinayak@ape
smol-git
~/smol-git

User prompts

```python
@cli.command()
@click.argument("repository")
@click.argument("branch")
def push(repository, branch):
    username = click.prompt("Username for 'https://github.co
    password = click.prompt(
        f"Password for 'https://{username}@github.com'",
        hide_input=True
    )
    # push changes
```


vinayak@ape
smol-git
~/smol-git

```
$ smol-git --help
Usage: smol-git [OPTIONS] COMMAND [ARGS]...

  smol-git - the stupid content tracker

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  clone   Clone a repository into a new directory.
  commit  Record changes to the repository.
  config  Get and set repository or global options.
  log     Show commit logs.
```

Testing click code

```python
from click.testing import CliRunner
from smol_git.cli import cli

def test_git_log():
    runner = CliRunner()
    result = runner.invoke(cli, ['log'])
    assert result.exit_code == 0
    assert result.output == expected_output_log
```

https://click.palletsprojects.com

Packaging the CLI


.
├── setup.py
└── smol_git
    ├── cli.py
    ├── __init__.py
    ├── utils.py
    └── __version__.py

```
from setuptools import setup

setup(
    ...
    name="smol-git",
    entry_points={
        "console_scripts": [
          "smol-git = smol_git.cli:cli"
        ]
    },
    ...
)
```

Pushing to PyPI

```
$ python setup.py sdist bdist_wheel
$ twine upload dist/*
```

User experience

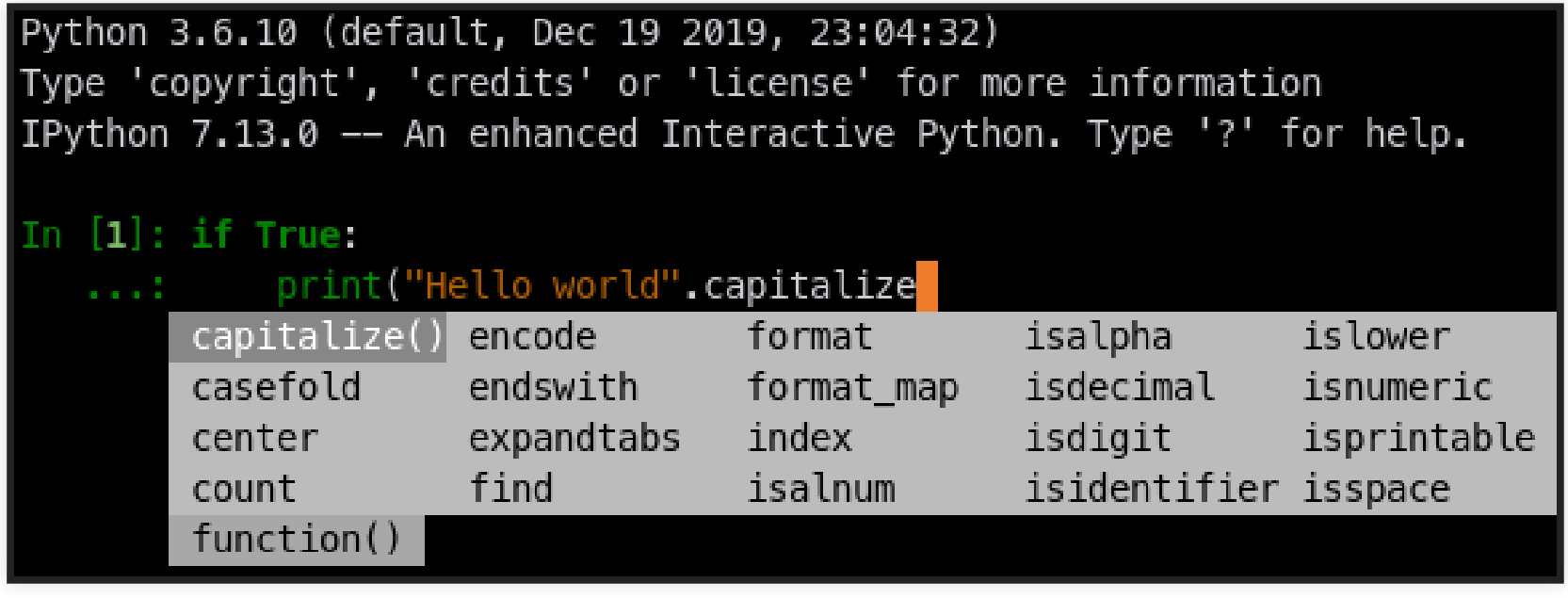
Unix philosophy

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

Make features discoverable

- Persistent history
- History search
- Auto-completion

prompt-toolkit



Resources

- Talk slides: https://vinayak.io/talks
- The TTY demystified
- What is the exact difference between a terminal, a shell, a tty and a console?
- Keynote by Brandon Rhodes - North Bay Python 2017
- Terminal whispering by Thomas Ballinger - PyCon 2015
- Writing Command Line Applications that Click by Dave Forgac - PyCon 2019
- Fish shell design document
- Awesome CLI Tools by Amjith Ramanujam - PyCon 2017
- https://github.com/vinayak-mehta/smol-git

@vortex_ape

vinayak.io

#talk-guide-to-clis