I/O redirection & piping

Adapted from materials by Dr. Carrier

Matt Benson - Unsplash

Have we seen this before?

Have we seen this before?

Yes!

Have we seen this before?

Yes!

```
sort unsorted.txt > sorted.txt
```

Have we seen this before?

Yes!

```
sort unsorted.txt > sorted.txt
```

```
sort unsorted.txt | uniq
```

I/O Redirection

The standard streams

We have three:

The standard streams

We have three:

0. stdin - Standard input (user typing)

The standard streams

We have three:

0. stdin - Standard input (user typing)
1. stdout - Standard output (print to terminal)

The standard streams

We have three:

0. stdin - Standard input (user typing)
1. stdout - Standard output (print to terminal)
2. stderr - Standard error (also prints to terminal)

The standard streams

We have three:

0. stdin - Standard input (user typing)
1. stdout - Standard output (print to terminal)
2. stderr - Standard error (also prints to terminal)

Redirect stdout to file:

```
ls -la > dir_contents.txt
```

The standard streams

We have three:

0. stdin - Standard input (user typing)
1. stdout - Standard output (print to terminal)
2. stderr - Standard error (also prints to terminal)

Redirect stdout to file:

```
ls -la > dir_contents.txt
```

Redirect stderr to file:

```
ls fake_dir 2> error.txt
```

The standard streams

We have three:

0. stdin - Standard input (user typing)
1. stdout - Standard output (print to terminal)
2. stderr - Standard error (also prints to terminal)

Redirect stdout to file:

```
ls -la > dir_contents.txt
```

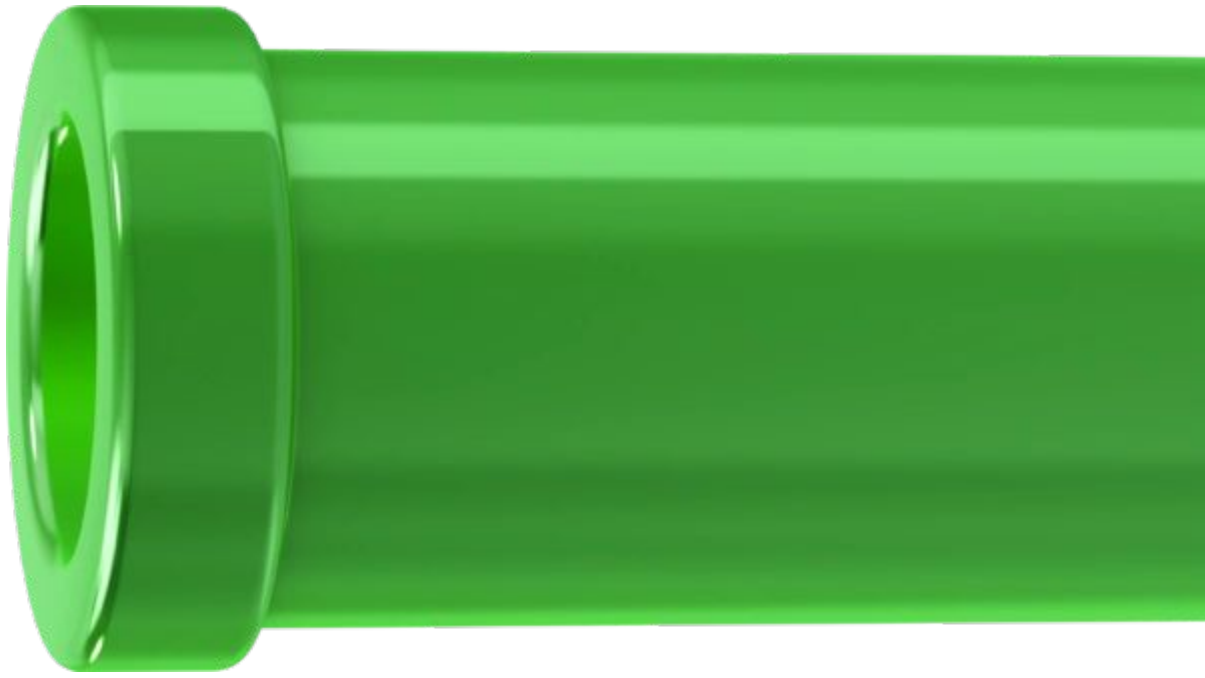
Redirect stderr to file:

```
ls fake_dir 2> error.txt
```

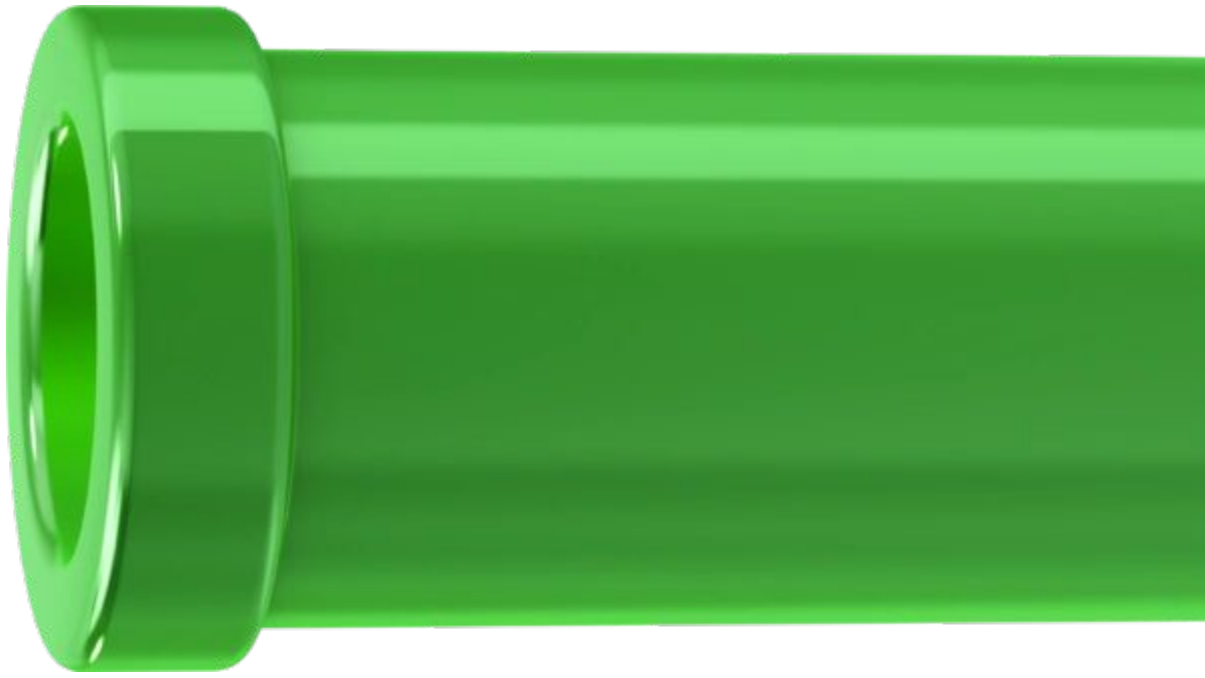
Redirecting a file to stdin:

```
sort < my_file.txt
```

Piping



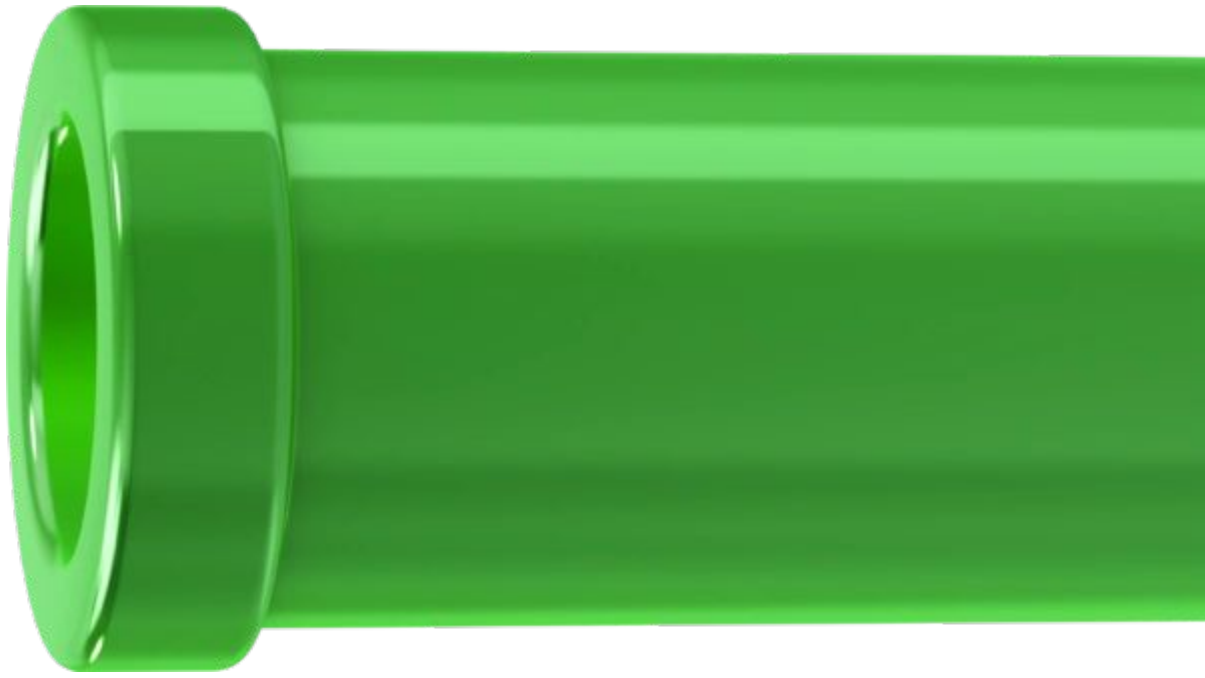
Piping



We want to prevent this situation:

```
command1 > temp_file.txt
command2 temp_file.txt
```

Piping

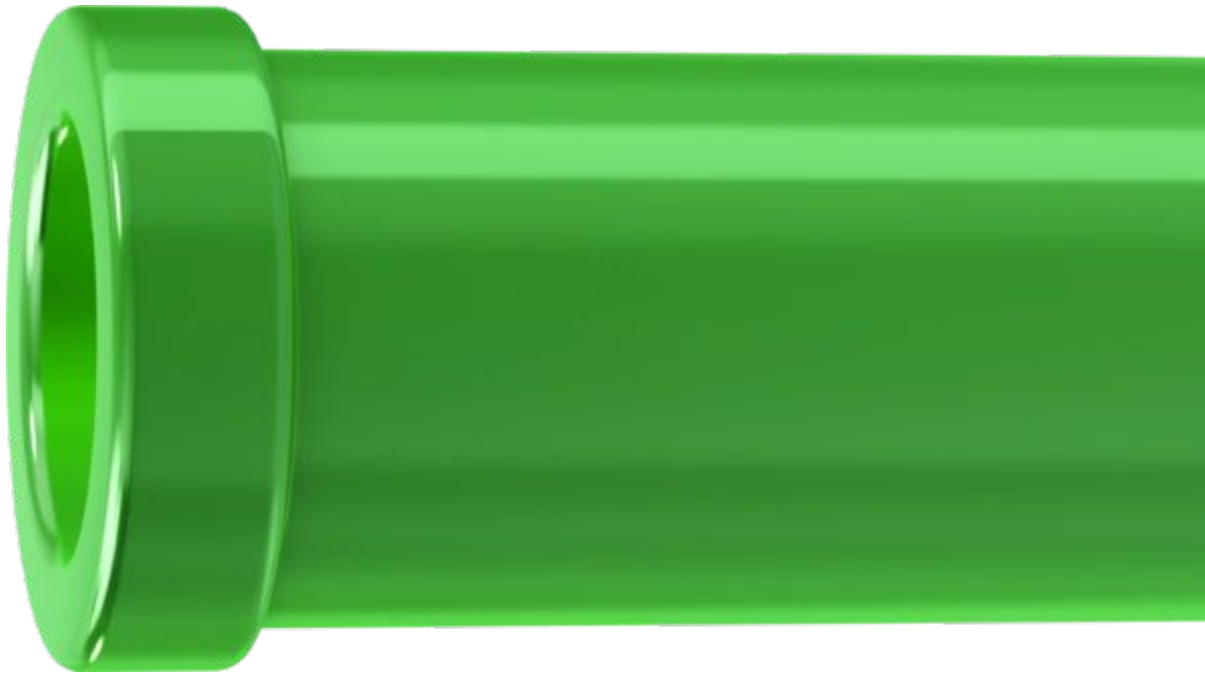


We want to prevent this situation:

```
command1 > temp_file.txt
command2 temp_file.txt
```

Instead, we can use a pipe:

Piping



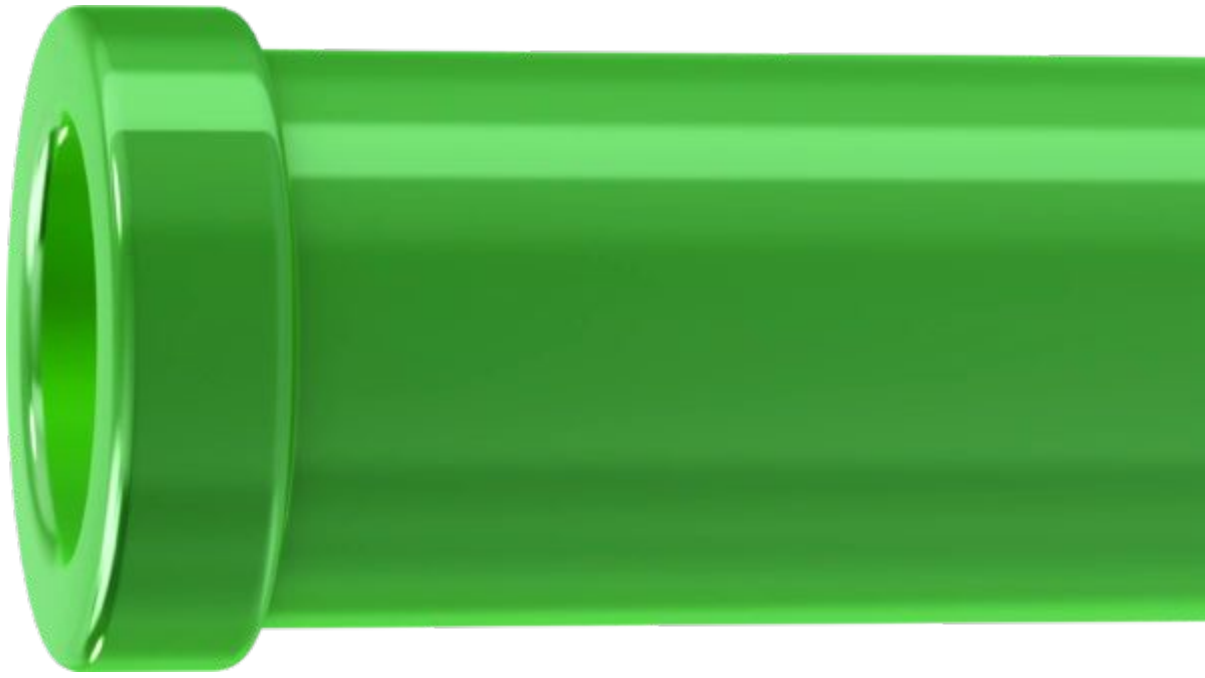
We want to prevent this situation:

```
command1 > temp_file.txt
command2 temp_file.txt
```

Instead, we can use a pipe:

```
command1 args | command2
```

Piping



We want to prevent this situation:

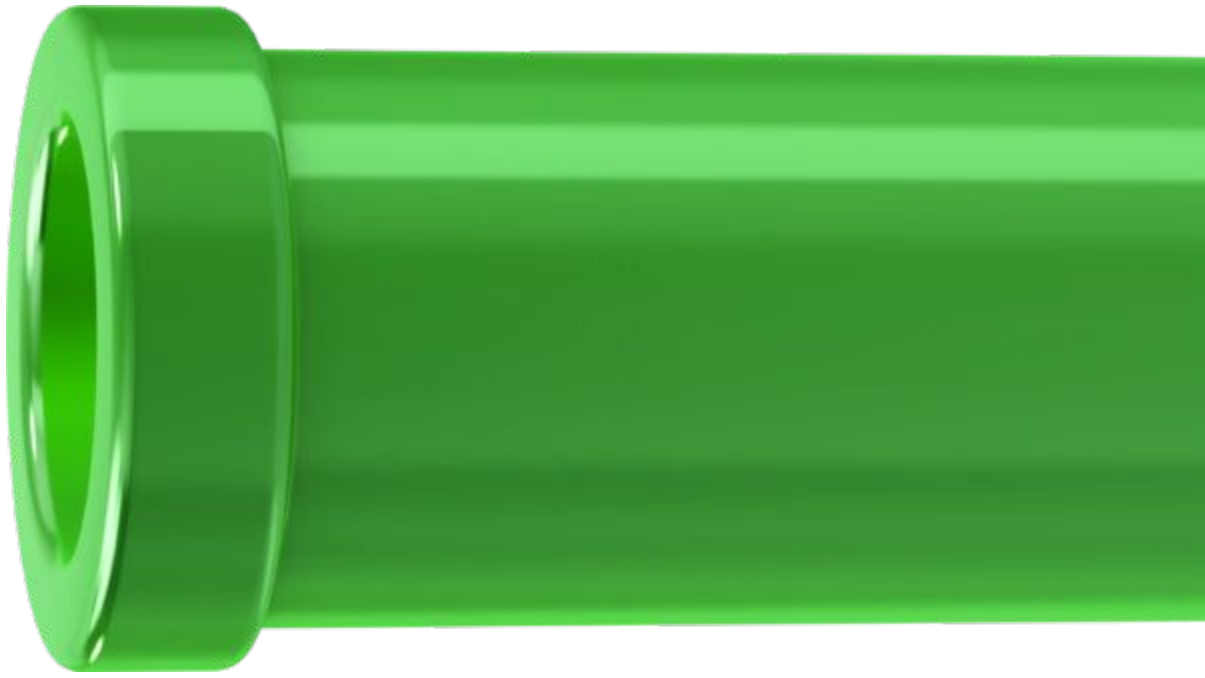
```
command1 > temp_file.txt
command2 temp_file.txt
```

Instead, we can use a pipe:

```
command1 args | command2
```

```
cat file | sort
```

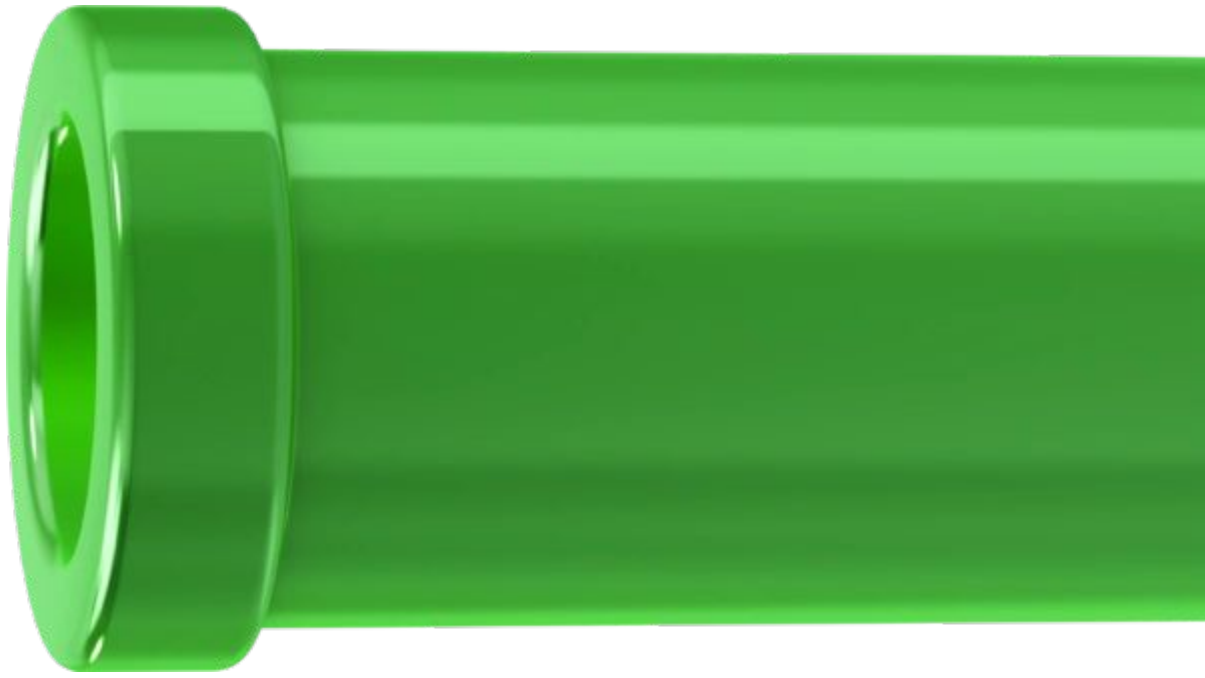
Piping



We can have however many pipes we need:

```
command1 | command2 | command3
```

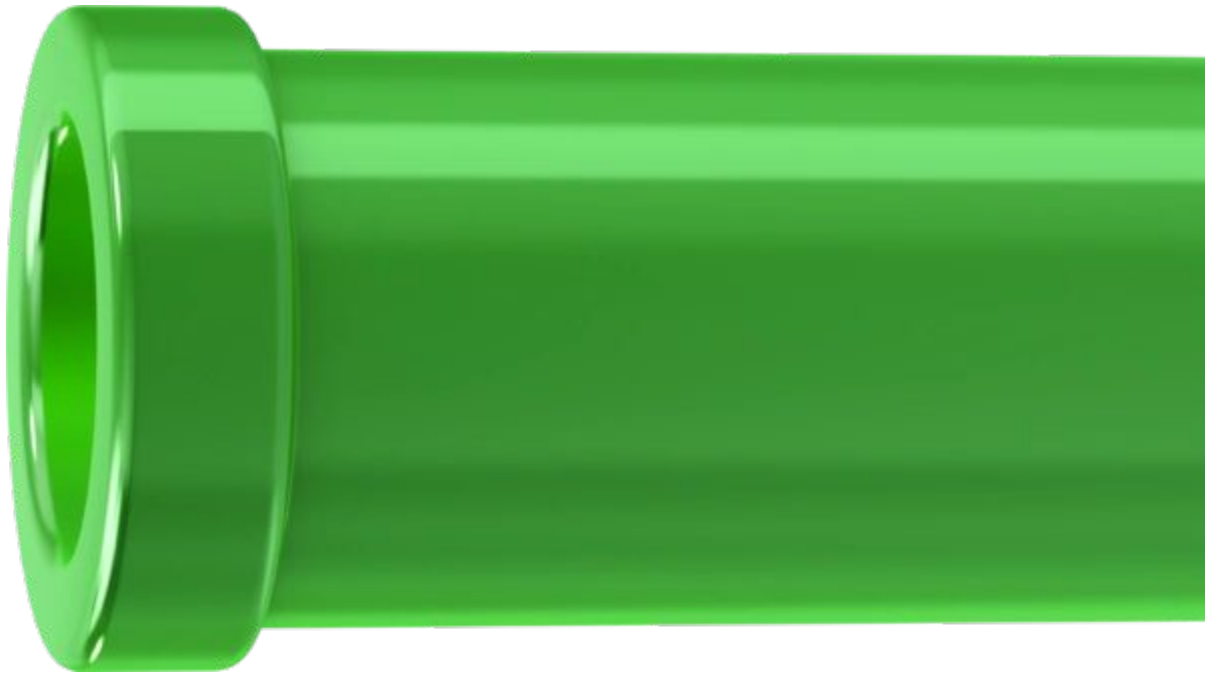
Piping



We can have however many pipes we need:

```
command1 | command2 | command3
cat file | sort | uniq
```

Piping



We can have however many pipes we need:

```
command1 | command2 | command3
```

```
cat file | sort | uniq
```

We can also combine with I/O redirection

```
cat file | sort | uniq > result.txt
```

Chaining + Misc

What if you want someone to run multiple commands in order, but you don’t want them to copy + paste multiple times?

Chaining + Misc

What if you want someone to run multiple commands in order, but you don’t want them to copy + paste multiple times?

```
command1; command2
```

Chaining + Misc

What if you want someone to run multiple commands in order, but you don’t want them to copy + paste multiple times?

```
command1; command2
```

This does not care if command1 succeeds

Chaining + Misc

What if you want someone to run multiple commands in order, but you don’t want them to copy + paste multiple times?

```
command1; command2
```

This does not care if command1 succeeds

```
command1 && command2
```

Chaining + Misc

What if you want someone to run multiple commands in order, but you don’t want them to copy + paste multiple times?

```
command1; command2
```

This does not care if command1 succeeds

```
command1 && command2
```

This only executes command2 if command1 succeeded!

Chaining + Misc

What if you want someone to run multiple commands in order, but you don’t want them to copy + paste multiple times?

```
command1; command2
```

This does not care if command1 succeeds

```
command1 && command2
```

This only executes command2 if command1 succeeded!

This is different from:

```
command1 &
```

Chaining + Misc

What if you want someone to run multiple commands in order, but you don’t want them to copy + paste multiple times?

```
command1; command2
```

This does not care if command1 succeeds

```
command1 && command2
```

This only executes command2 if command1 succeeded!

This is different from:

```
command1 &
```

This runs it in the background