Learn GNU Parallel in 5 minutes

You just need to run commands in parallel. You do not care about fine tuning.

To get going please run this to make some example files:

```
# If your system does not have 'seq', we will use 'jot' instead
if ! seq 1 2>/dev/null; then alias seq=jot; fi

seq 5 | parallel 'seq {} > example.{}'
```

Input sources

GNU **parallel** reads values from input sources. One input source is the command line. The values are put after **:::** :

```
parallel echo ::: 1 2 3 4 5
```

This makes it easy to run the same program on some files:

```
parallel wc ::: example.*
```

If you give multiple **:::**s, GNU **parallel** will make all combinations:

```
parallel wc ::: -l -c ::: example.*
```

GNU **parallel** can also read the values from stdin (standard input):

```
seq 5 | parallel echo
```

Building the command line

The command line is put before the **:::**. It can contain contain a command and options for the command:

```
parallel wc -l ::: example.*
```

The command can contain multiple programs. Just remember to quote characters that are interpreted by the shell (such as **;**):

```
parallel echo counting lines';' wc -l ::: example.*
```

The value will normally be appended to the command, but can be placed anywhere by using the replacement string **{}**:

```
parallel echo counting {}';' wc -l {} ::: example.*
```

When using multiple input sources you use the positional replacement strings:

```
parallel echo count {1} in {2}';' wc {1} {2} ::: -l -c ::: example.*
```

Controlling the output

The output will be printed as soon as the command completes. This means the output may come in a different order than the input:

```
parallel sleep {}';' echo {} done ::: 5 4 3 2 1
```

You can force GNU **parallel** to print in the order of the values with **--keep-order**/**-k**. This will still run the commands in parallel. The output of the later jobs will be delayed, until the earlier jobs are printed:

```
parallel -k sleep {}';' echo {} done ::: 5 4 3 2 1
```

Controlling the execution

If your jobs are compute intensive, you will most likely run one job for each core in the system. This is the default for GNU **parallel**.

But sometimes you want more jobs running. You control the number of job slots with **-j**. Give **-j** the number of jobs to run in parallel:

```
parallel -j50 \
  wget http://ftpmirror.gnu.org/parallel/parallel-{1}{2}22.tar.bz2 \
  ::: 2012 2013 2014 2015 2016 \
  ::: 01 02 03 04 05 06 07 08 09 10 11 12
```

Pipe mode

GNU **parallel** can also pass blocks of data to commands on stdin (standard input):

```
seq 1000000 | parallel --pipe wc
```

This can be used to process big text files. By default GNU **parallel** splits on \n (newline) and passes a block of around 1 MB to each job.

That's it

You have now learned the basic use of GNU **parallel**. This will probably cover most cases of your use of GNU **parallel**.

The rest of this document is simply to go into more details on each of the sections and cover special use cases.

Learn GNU Parallel in an hour

In this part we will dive deeper into what you learned in the first 5 minutes.

To get going please run this to make some example files:

```
seq 6 > seq6
seq 6 -1 1 > seq-6
```

Input sources

On top of the command line, input sources can also be stdin (standard input or '-'), files and fifos and they can be mixed. Files are given after **-a** or **::::**. So these all do the same:

```
parallel echo Dice1={1} Dice2={2} ::: 1 2 3 4 5 6 ::: 6 5 4 3 2 1
parallel echo Dice1={1} Dice2={2} ::: <(seq 6) ::: <(seq 6 -1 1)
parallel echo Dice1={1} Dice2={2} :::: seq6 seq-6
parallel -a seq6 -a seq-6 echo Dice1={1} Dice2={2}
parallel -a seq6 echo Dice1={1} Dice2={2} :::: seq-6
parallel echo Dice1={1} Dice2={2} ::: 1 2 3 4 5 6 :::: seq-6
cat seq-6 | parallel echo Dice1={1} Dice2={2} :::: seq-6 -
```

If stdin (standard input) is the only input source, you do not need the '-':

```
cat seq6 | parallel echo Dice1={1}
```

You can link multiple input sources with **:::+** and **::::+**:

```
parallel echo {1}={2} ::: I II III IV V VI :::+ 1 2 3 4 5 6
parallel echo {1}={2} ::: I II III IV V VI ::::+ seq6
```

Building the command line

The command

The command can be a script, a binary or a Bash function if the function is exported using **export -f**:

```
# Works only in Bash
my_func() {
  echo in my_func "$1"
}
export -f my_func
parallel my_func ::: 1 2 3
```

The replacement strings

GNU **parallel** has some replacement strings to make it easier

Controlling the output

Controlling the execution

Remote execution

Pipe mode =head2 That's it

Advanced usage

env_parallel, parset, env_parset