Running jobs with dependencies on the Peregrine computer cluster

© 2016 Richel Bilderbeek

https://github.com/richelbilderbeek/Peregrine

What, Why, Mastery

- I will show how to set up jobs with one or more dependencies

- Starting each step manually is tedius and might lead to irreproducible research

- You can create jobs that are dependent on one or more jobs

- Not: running a program in parallel on multiple cores

Overview

- bash #1
- Job without dependencies
- bash #2
- Job dependent on one job
- Job dependent on multiple jobs
- bash #3
- Conclusion

bash #1

```
echo "Hello world"
```

Display 'Hello World'

```
ls
```

List directory content

```
echo "Hello world" > tmp.txt
```

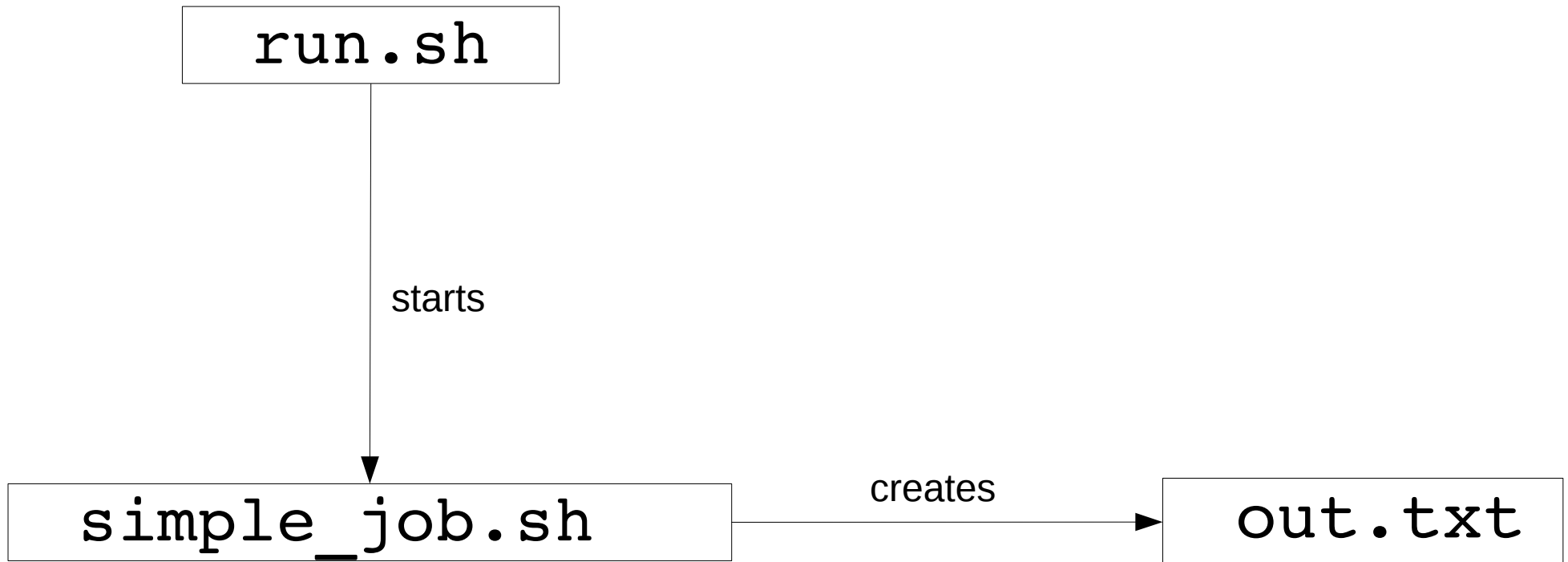Streams the text 'Hello world' to file tmp.txt in overwrite mode

```
cat tmp.txt
```

Shows the file tmp.txt

```
./run.sh
```

Run the local executable bash script called run.sh

Job without dependencies

run.sh

starts

simple_job.sh → creates → out.txt

Run and check it

```
p230198@pg-login:simple_job ./run.sh

Submitted batch job 7144352
```

```
p230198@pg-login:simple_job cat out.txt

Created by simple_job.sh
```

run.sh

```bash
#!/bin/bash
sbatch simple_job.sh
```

simple_job.sh

```bash
#!/bin/bash

#SBATCH --time=0:01:00

#SBATCH --nodes=1

#SBATCH --ntasks-per-node=1

#SBATCH --ntasks=1

#SBATCH --mem=1M

#SBATCH --job-name=simple_job

#SBATCH --output=simple_job.log
echo "Created by simple_job.sh" > out.txt
```

bash #2

```
echo "Hello world" | cut -d " " -f 1
```

Pipe the words 'Hello world' to cut. Show, using space as a delimiter, the first field

```
echo "Hello world" | cut -d " " -f 1 > tmp.txt
```

Pipe the words 'Hello world' to cut.
Stream, using space as a delimiter, the first field to tmp.txt in overwrite mode

```
echo "Hello world" | cut -d " " -f 2 >> tmp.txt
```

Pipe the words 'Hello world' to cut.
Append, using space as a delimiter, the second field to tmp.txt

```
hi=`echo "Hello world" | cut -d " " -f 1`

echo $hi
```
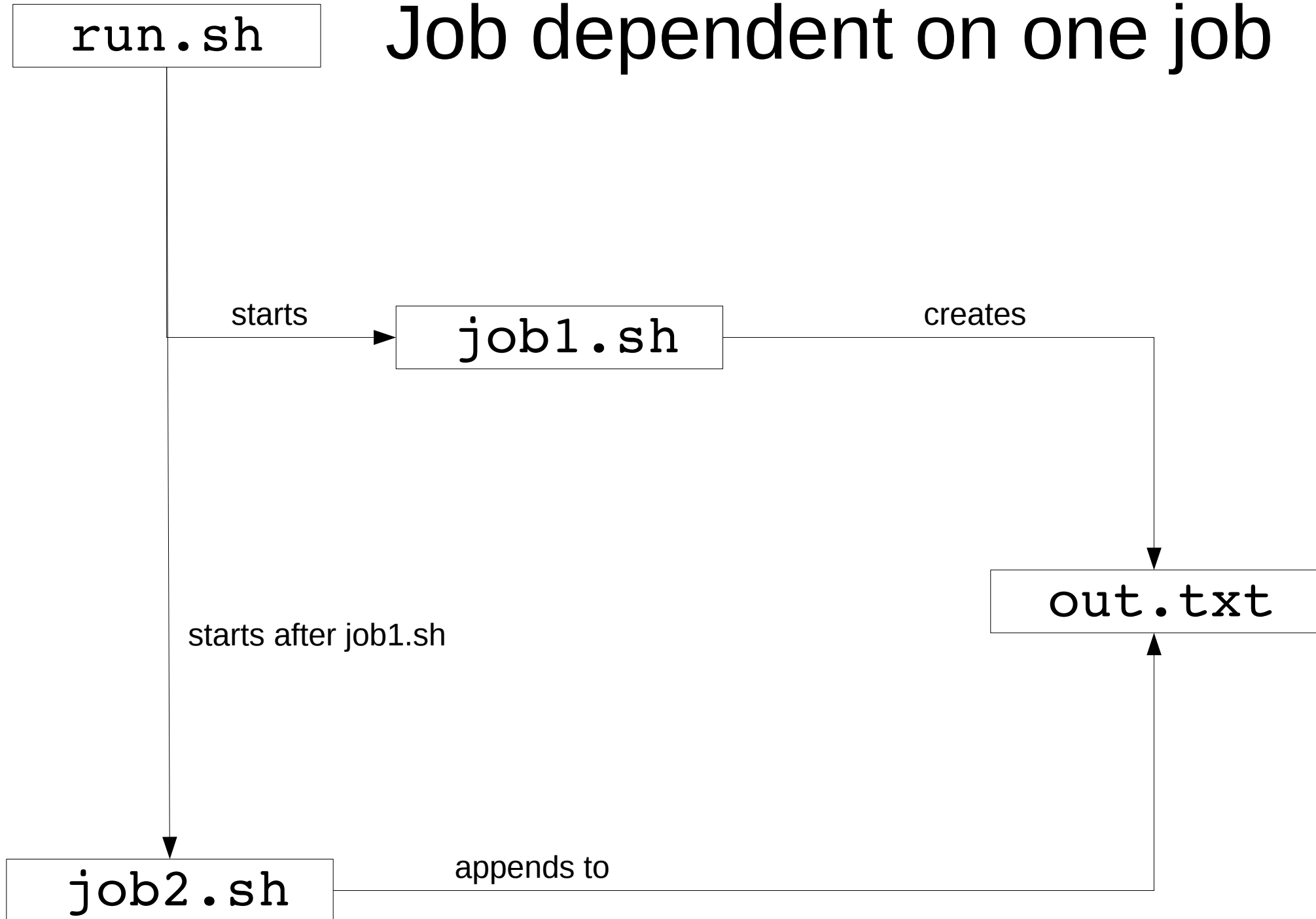
Pipe the words 'Hello world' to cut.
Extract, using space as a delimiter, the first field.
Store this result in the variable called 'hi'
Display the content of the variable 'hi'

Job dependent on one job

```
run.sh
```

```
job1.sh
```
starts

creates

```
out.txt
```

```
job2.sh
```
starts after job1.sh

appends to

Run and check it

```
p230198@pg-login:jobs_in_sequence ./run.sh

Submitted batch job 7144425
```

```
p230198@pg-login:jobs_in_sequence cat out.txt

Created by job1.sh

Appended by job2.sh
```

run.sh

```bash
#!/bin/bash


# Submit job1, store job ID

jobid=`sbatch job1.sh | cut -d ' ' -f 4`


# Submit job2 after job1

sbatch --dependency=afterok:$jobid job2.sh
```
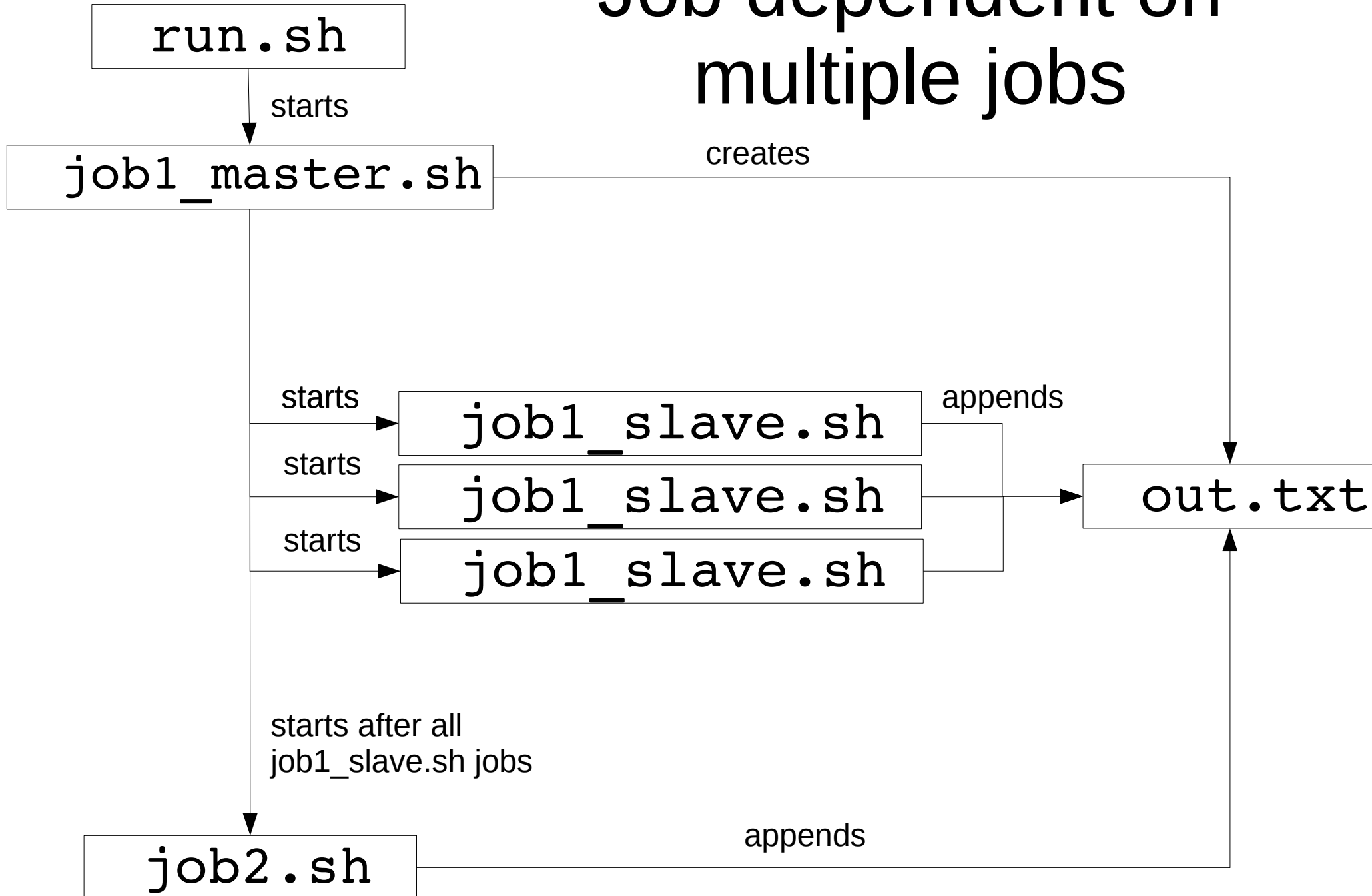
Submitted batch job 7144425

job1.sh

```bash
#!/bin/bash

#SBATCH --time=0:01:00

#SBATCH --nodes=1

#SBATCH --ntasks-per-node=1

#SBATCH --ntasks=1

#SBATCH --mem=1M

#SBATCH --job-name=job1

#SBATCH --output=job1.log

echo "Created by job1.sh" > out.txt
```

job2.sh

```bash
#!/bin/bash
#SBATCH --time=0:01:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=1M
#SBATCH --job-name=job2
#SBATCH --output=job2.log
echo "Appended by job2.sh" >> out.txt
```

Job dependent on multiple jobs

```
run.sh
```

starts

```
job1_master.sh
```

creates

starts

```
job1_slave.sh
```

starts

```
job1_slave.sh
```

starts

```
job1_slave.sh
```

appends

```
out.txt
```

starts after all
job1_slave.sh jobs

```
job2.sh
```

appends

Run and check it

```
p230198@pg-login:jobs_in_parallel ./run.sh
Submitted batch job 7144527
```

```
p230198@pg-login:jobs_in_parallel cat out.txt
Created by job1 master
Added by job1 slave number 2
Added by job1 slave number 1
Added by job1 slave number 3
Appended by job2
```
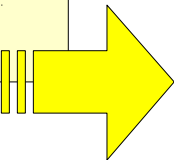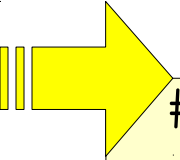
run.sh

```bash
#!/bin/bash
sbatch job1_master.sh
```

```bash
#!/bin/bash

#SBATCH --time=0:01:00

#SBATCH --nodes=1

#SBATCH --ntasks-per-node=1

#SBATCH --ntasks=1

#SBATCH --mem=1M

#SBATCH --job-name=job1_master

#SBATCH --output=job1_master.log


# Start file creation

# (may create race condition)

echo "Created by job1 master" > out.txt
```

```
# Submit all jobs, while collecting the job IDs
jobids=()
for i in {1..3}
do
  cmd="sbatch job1_slave.sh $i"
  jobids+=(`$cmd | cut -d ' ' -f 4`)
done

# Convert array of job IDs to colon-seperated string
txt=$(printf ":%s" "${jobids[@]}")
txt=${txt:1}

# Submit job2 after all previous jobs have finished
sbatch --dependency=afterok:$txt job2.sh
```

job1_slave.sh

```bash
#!/bin/bash
#SBATCH --time=0:01:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=1M
#SBATCH --job-name=job1_slave
#SBATCH --output=job1_slave_%j.log
echo "Added by job1 slave number "$1 >> out.txt


# To be sure the output file is created before the
next step
sleep 1
```

job2.sh

```bash
#!/bin/bash
#SBATCH --time=0:01:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=1M
#SBATCH --job-name=job2
#SBATCH --output=job2.log
echo "Appended by job2" >> out.txt
```

bash #3

```
egrep -R "error"
```

Searches all files in folders and subfolders for the regex 'error'

```
egrep -iR "error"
```

Searches all files in folders and subfolders for the regex 'error', case insensitively

```
egrep -iR "error" --include=*.log
```

Searches all files with a .log file extension in folders and subfolders for the regex 'error', case insensitively

```
scp p230198@peregrine.hpc.rug.nl:/home/p230198/any_folder/*.*
~/any_other_folder
```

Securely copy all files from a Peregrine folder to a folder in your LWP home

```
echo "alias q='squeue -u $USER'" >> ~/.bashrc
```

Use the command 'q' to show your jobs in the queue (after a restart)

Conclusion

- Setting up linear and parallel jobs is doable

- Some knowledge of bash is useful
  - especially when something goes wrong