University of Zurich[UZH]

URPP Evolution

URPP tutorial

Basic bash scripting 2
&
Introduction to cluster submission system

Dr. Heidi E.L. Lischer
University of Zurich
Switzerland

21 January, 2016

What is a bash script?

- Mainly a set of commands that can be executed in the terminal
  - Text file with commands
  - Should begin with the shebang line (`#!/bin/bash`)
  - Executed line by line → new line = new command

- Example: script.sh

```
#!/bin/bash
#this is a comment
echo "Hello World"
```

→ shebang line
→ A comment (ignored)
→ Print something in terminal

```
chmod +x script.sh
./script.sh
Hello World
```

→ Give execution permission
→ Run script

Variables and arrays

Variables:

- Store data and configuration options
- Create: name followed by “=“ and the value
- Call: “$” in front of the name, enclose the name in {} if directly followed by something else

```
a=/home/user
echo "Data folder: ${a}/data"
```

Arrays:

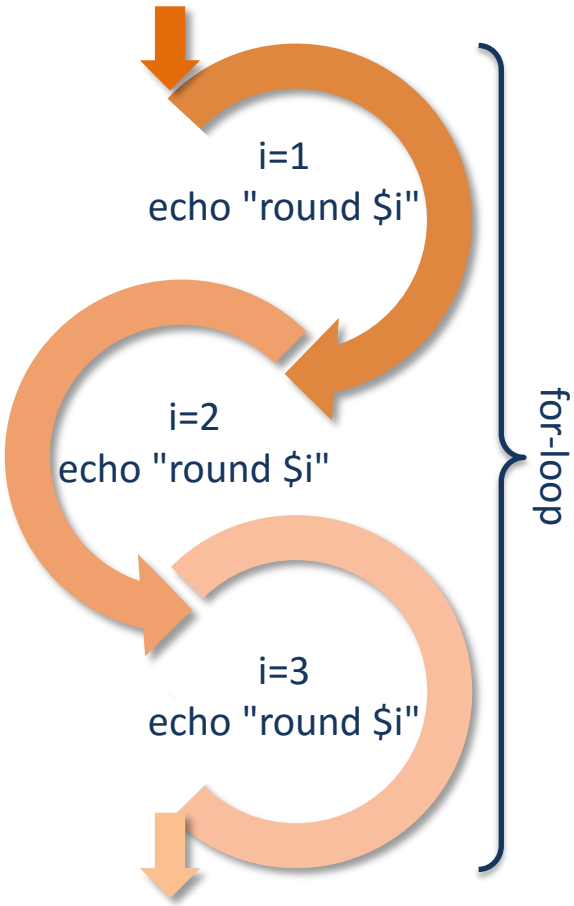
- Arrays are variables containing multiple values
- Create: name followed by “=()“, values are space separated
- Values can be accessed by their index (number) starting from 0

```
col=(red blue yellow)
echo "The sun is ${col[2]}"
```

For loop

```
for VARIABLE in 1 2 3
do
  command1
done
```

- Can be used to repeat certain tasks



- Examples:

```
#!/bin/bash
for i in 1 2 3
do
  echo "round $i"
done
```

```
#!/bin/bash
for i in {1..3}
do
  echo "round $i"
done
```

```
round 1
round 2
round 3
```

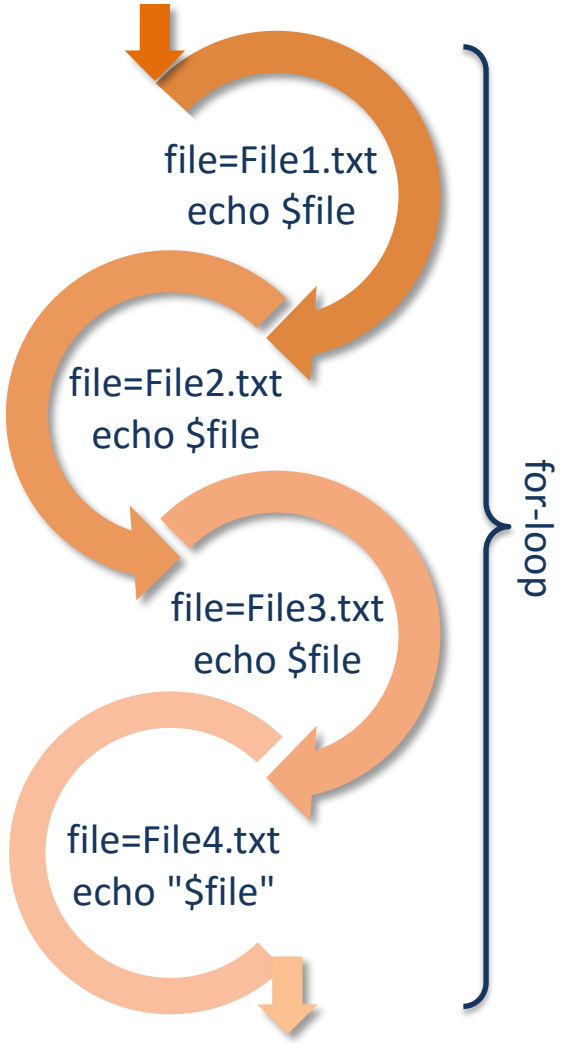
For loop

- Examples:

Go through all files in a folder:

```
#!/bin/bash
for file in /home/user/*
do
  echo $file
done
```

```
File1.txt
File2.txt
File3.txt
File4.txt
```



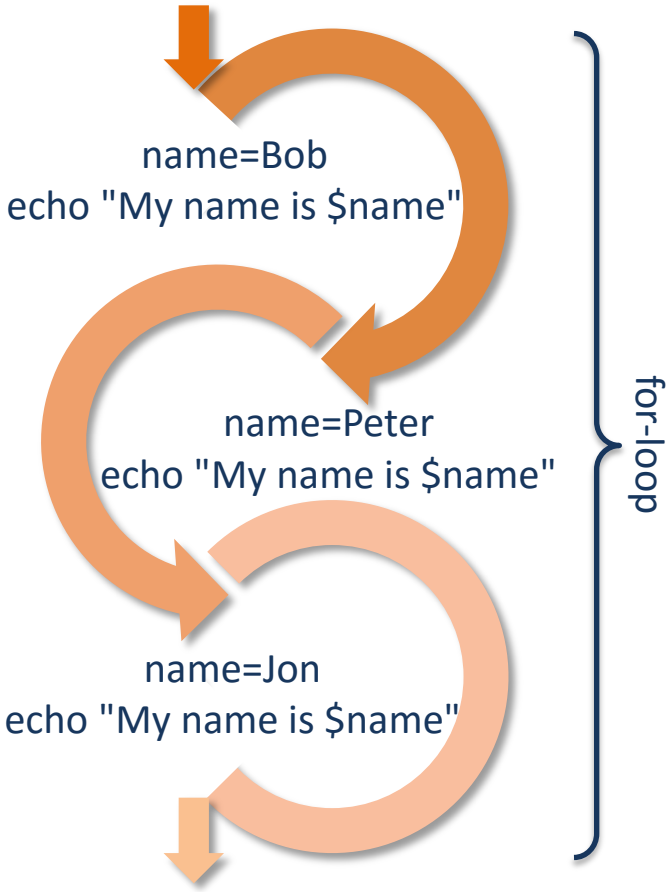
For loop

- Examples:

Go through all items of an array:

```
#!/bin/bash
names=(Bob Peter Jon)
for name in ${names[*]}
do
  echo "My name is $name"
done
```

```
My name is Bob
My name is Peter
My name is Jon
```



While loop

- Can also be used to repeat certain tasks:

```
while [[ something ]]
do
  command
done
```

- Example:

```
#!/bin/bash
count=0
while [[ $count -lt 4 ]]
do
  echo $count
  let count+=1
done
echo "done"
```

```
0
1
2
3
done
```

If statement

- If statements can be used to check for something and do something else depending on the outcome of the check

```
if [[ someting ]]
then
  command1
elif [[ something ]]
then
  command2
else
  command3
fi
```

If statement

Boolean operators:

- **-e FILE**: True if file exists

```
#!/bin/bash
if [[ -e log.txt ]]
then
  echo "log file exist"
else
  echo "log file doesn't exist"
fi
```

If statement

Boolean operators:

- **-e FILE**: True if file exists
- **STRING = STRING**: True if first string is identical to the second
- **STRING != STRING**: True if first string is not identical to the second
- **STRING < STRING**: True if first string sorts before the second
- **STRING > STRING**: True if first string sorts after the second

```
#!/bin/bash
name=Bob
if [[ $name = "Rod" ]]
then
  echo "Your name is Rod"
else
  echo "Your name is not Rod"
fi
```

If statement

```
#!/bin/bash
a=10
if [ $a -eq 7 ]
then
  echo "Not equal to 7"
else
  echo "You guessed $a"
fi
```

Boolean operators:

- **-e FILE**: True if file exists
- **STRING = STRING**: True if first string is identical to the second
- **STRING != STRING**: True if first string is not identical to the second
- **STRING < STRING**: True if first string sorts before the second
- **STRING > STRING**: True if first string sorts after the second
- **INT -eq INT**: True if both integers are identical
- **INT -ne INT**: True if integers are not identical
- **INT -lt INT**: True if first integer is less than the second
- **INT -gt INT**: True if first integer is greater than the second
- **INT -le INT**: True if first integer is less than or equal to the second
- **INT -ge INT**: True if first integer is greater than or equal to the second

If statement

```
#!/bin/bash
a=Bob
if [[ $a = "Rob" || $a = "Tod" ]]
then
  echo "Your name is Rod or Tod"
else
  echo "Your name is not Rod or Tod"
fi
```

Boolean operators:

- **-e FILE**: True if fil
- **STRING = STRING**: True if first string is identical to the second
- **STRING != STRING**: True if first string is not identical to the second
- **STRING < STRING**: True if first string sorts before the second
- **STRING > STRING**: True if first string sorts after the second
- **INT -eq INT**: True if both integers are identical
- **INT -ne INT**: True if integers are not identical
- **INT -lt INT**: True if first integer is less than the second
- **INT -gt INT**: True if first integer is greater than the second
- **INT -le INT**: True if first integer is less than or equal to the second
- **INT -ge INT**: True if first integer is greater than or equal to the second
- **! EXPR**: Inverts the result of the expression (logical NOT)
- **EXPR && EXPR**: True if both expressions are true (logical AND)
- **EXPR || EXPR**: True if either expression is true (logical OR)

Functions

- Functions: blocks of commands
- Code that you may call multiple times within your script
- Arguments can be passed to functions
  and be accessed by $1 (first argument), $2 (second argument),…
- Example:

```
#!/bin/bash
sum() {
  echo "$1 + $2 = $(($1 + $2))"
}

sum 1 4
sum 8 7
```

→ Set up function

→ Call function

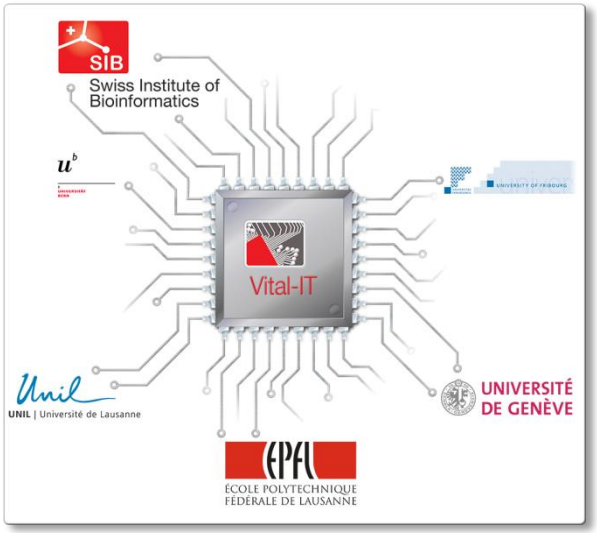
→ Call function

```
1 + 4 = 5
8 + 7 = 15
```

Cluster submission systems

- Usually a cluster consist of
  - Front-end machine (where user log in)
    → where you submit jobs
  - Compute nodes
    → where computations take place

- Specialized software for submitting jobs to compute nodes
  - LSF (Load Sharing Facility):
    Vital-IT (SIB)
  - SGE (Sun Grid Engine):
    Wagner-cluster (UZH)
  - Slurm workload manager:
    Hydra/ Piz Dora (S3IT UZH)



When should I use a cluster?

- Application
  - runs too long on a single computer
  - takes too much CPU

- The problem (input data) can be split into pieces which can be executed in parallel
  Examples:
  - BLAST 1 million sequences
  - simulate model 1 million times

- Every program (pipeline) to be executed on a cluster needs to be defined as a "job" with
  - Executable , input data, output data
  - Job characteristics (CPU, memory, run time,…)

Basics

- Get info about job status:
  - LSF: **bjobs**
  - SGE: **qstat**
  - Slurm: **squeue**

```
squeue
JOBID PARTITION   NAME   USER ST   TIME NODES NODELIST(REASON)
65646     batch   chem   mike  R 24:19     2 adev[7-8]
65647     batch    bio   joan  R  0:09     1 adev14
65648     batch   math   phil PD  0:00     6 (Resources)
```

- Kill jobs:
  - LSF: **bkill** jobID
  - SGE: **qdel** jobID
  - Slurm: **squeue** jobID

Submit job

Example command we want to execute:

```
blastall -p blastp -d "swiss" -i p123456.seq
```

- LSF:
  - Write shell script with job description: **`blast.sh`**

```
#!/bin/bash
#BSUB -o blast-output.txt      → Standard output is redirected to this file
#BSUB -e blast-error.txt       → Standard error is redirected to this file
#BSUB -J blastp                → Job name
#BSUB -R "mem > 4000"          → Minimal memory in MB
#BSUB -n 4                     → CPUs per process (multithreading)
#BSUB -R "span[ptile=4]"

blastall -p blastp -d "swiss" -i p123456.seq
```

```
bsub < ./blast.sh
```

Submit job

- SGE:
  - Write shell script with job description: **sge-blast.sh**

```
#!/bin/bash
#$ -l h_cpu=01:00:00        → Max. runtime
#$ -l h_vmem=1G             → Max. memory (M=MB, G=GB)
#$ -cwd
#$ -o blast-output.txt      → Standard output is redirected to this file
#$ -e blast-error.txt       → Standard error is redirected to this file
#$ -N blastp                → Job name
#$ -pe smp 4                → CPUs per process (multithreading)

blastall -p blastp -d "swiss" -i p123456.seq
```

```
qsub ./sge-blast.sh
```

Submit job

- Slurm:
  - Write shell script with job description: **slurm-blast.sh**

```
#!/bin/bash
#SBATCH --time=1:0:0                    → Max. runtime
#SBATCH --mem=16g                       → Max. memory (m=MB, g=GB)
#SBATCH --output=blast-output.txt       → Standard output is redirected
#SBATCH --error=blast-error.txt            to this file
#SBATCH --job-name=blastp               → Job name
#SBATCH --cpu-per-task=4                → CPUs per process
                                           (multithreading)


srun blastall -p blastp -d "swiss" -i p123456.seq
```

```
sbatch ./slurm-blast.sh
```

Array job

- Array job: submit the same job multiple times
  - Simulation has to run 20 times
    → submit a single job rather than 20 individual ones

- LSF:
  - mysim.sh

```
#!/bin/bash
#BSUB -J array[1-20]
#BSUB -o output-%I.txt
#BSUB -e error-%I.txt

run-mysim --seed=$LSB_JOBINDEX
```

→ Will start 20 subjobs

→ %I: get index of job array in header

→ get index of job array

```
bsub < ./mysim.sh
```

Array job

- SGE:
  - mysim.sh

```
#!/bin/bash
#$ -t 1-20
#$ -l h_cpu=01:00:00
#$ -l h_vmem=1G
#$ -cwd
#$ -o output-$TASK_ID.txt
#$ -e error-$TASK_ID.txt
#$ -N array-job

run-mysim --seed=$SGE_TASK_ID
```

→ Will start 20 subjobs

→ get index of job array in header

→ get index of job array

```
qsub ./mysim.sh
```

Array job

- Slurm:
  - mysim.sh

```
#!/bin/bash
#SBATCH --array=1-20
#SBATCH --time=1:0:0
#SBATCH --mem=16g
#SBATCH --output=output-%a.txt
#SBATCH --error=error-%a.txt
#SBATCH --job-name=array-job

srun run-mysim --seed=$SLURM_ARRAY_TASK_ID
```

→ Will start 20 subjobs

→ get index of job array in header

get index of job array

```
sbatch ./mysim.sh
```

Acknowledgment

- Bash scripting:
  - http://www.allaboutlinux.eu/bash-script-for-beginners/
  - http://www.howtogeek.com/67469/the-beginners-guide-to-shell-scripting-the-basics/
  - http://bash.cyberciti.biz/guide/Main_Page
  - http://www.arachnoid.com/linux/shell_programming.html
  - http://www.tldp.org/LDP/Bash-Beginners-Guide/html/
  - http://mywiki.wooledge.org/BashGuide
  - http://ryanstutorials.net/bash-scripting-tutorial/bash-functions.php
  - There are a lot of other online tutorials available and forums discussing diverse kinds of topics!
- Cluster:
  - SIB: HPC in Life Science workshop
  - http://www.ceci-hpc.be/slurm_tutorial.html
  - http://www.s3it.uzh.ch/infrastructure/hydra/usage/