Bash

Stefan Kemnitz[1]

[1]Department of distributed high performance computing
University of Rostock

Bash, 2019

The Bourne Again Shell

- works interactive
- can be run non-interactive from files
- is easy to use compared to python or c++
- mostly works as a helper for the real programs

Variables

```
# define variables
LOCAL_VARIABLE="value"
export GLOBAL_VARIABLE="global_value"

# use variables
echo $LOCAL_VARIABLE
```

- can be defined for the local environment
- can be exported so that other scripts can read them
- predefined variables already exist
  - PWD
  - HOME
  - TERM
  - SHELL
  - USER
- show them via "env"

Commands

- every program that is installed on the system
  - date
  - hostname
  - bc
  - sed
- everything that is build into the bash shell
  - cd
  - mkdir
  - time
  - echo

Example Job Script

Listing 1: create_run.sh

```
#!/bin/bash

RUN_ID=0
BASE_DIR=$PWD
RUN_DIR=$BASE_DIR/run_dir_$RUN_ID

mkdir $RUN_DIR
cd $RUN_DIR

cp $BASE_DIR/my_program_src/* $RUN_DIR/
cd $RUN_DIR

./my_program.sh
```

Listing 2: my_program.sh

```
#!/bin/bash
echo "running my_program"
time echo "scale=3000; 4*a(1)" | bc -l
echo "done running my_program"
```

Function

```bash
#!/bin/bash

#define function
function my_function {
    echo "called with argument 1: $1 argument 2: $2"
}

# call function
my_function test abc
```

- structures code
- easier to read
- hides complexity

Loops

```
#!/bin/bash

for (( i = 0; i < 100; i++ )); do
    my_function $i abc
done
```

- used to call functions with different arguments
-

Job Management

- running programs can be suspended with ctrl+z
- fg <id> gets them into foreground
- bg <id> continues execution of a suspended application
- jobs shows all program
- programs can be stared in background with &
- wait can be used to wait for a specific or all applications to finish

Job Script

```
#!/bin/bash

function make_run {

  BASE_DIR="$PWD"
  RUN_DIR=$BASE_DIR/run_dir_$1

  mkdir $RUN_DIR
  cd $RUN_DIR

  cp $BASE_DIR/my_program_src/* $RUN_DIR/
  ./my_program.sh
}

for (( i = 0; i < 10; i++ )); do
  make_run $i &
done
wait
```

Embarrassingly Parallel

Output Redirection

- normally output is written to the shell
- output can be redirected
  - ">" character writes to file
  - "»" appends to file
  - "|" pipes to the next program as input

Output redirection applied

```
#!/bin/bash

function make_run {

  BASE_DIR="$PWD"
  RUN_DIR=$BASE_DIR/run_dir_$1

  mkdir $RUN_DIR
  cd $RUN_DIR

  cp $BASE_DIR/my_program_src/* $RUN_DIR/
  ./my_program.sh > my_program.out
}

for (( i = 0; i < 10; i++ )); do
  make_run $i &
done
wait
```

```
#!/bin/bash
echo "running my_program"
time echo "scale=3000; 4*a(1)" | bc -l
echo "done running my_program"
```

Summary

- problems:
  - no way to control the maximum number of processes
  - load in-balance not handled
  - can not be spread across multiple systems
  - does not continue execution after user logged out
  - no way to share a system with multiple users
- solutions:
  - use a local job scheduler:
    - xargs (single system)
    - gnu parallel (can do multi-system with ssh)
  - user a resource manager:
    - slurm
    - pbs
    - ibm load leveler