NGS Discussion Class - 12th

Using Linux & SeqPipe

Linlin Yan @ CBI, PKU
2012-12-19

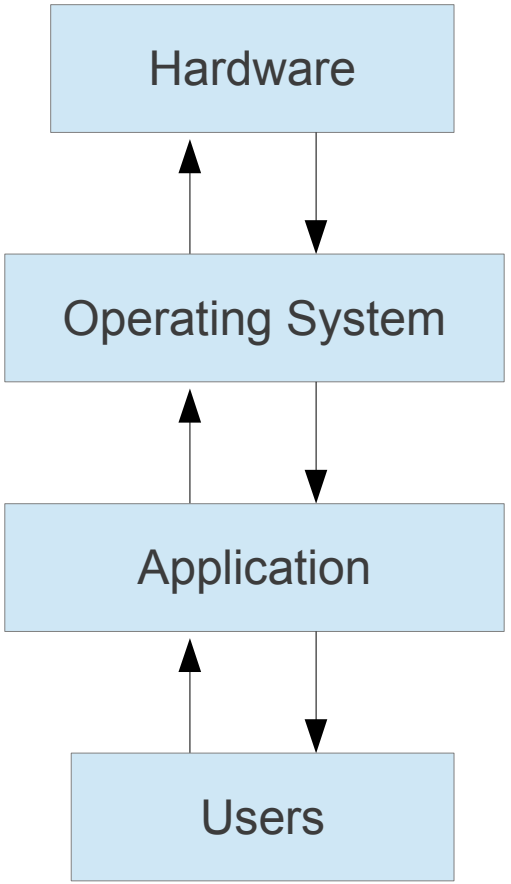
Outline

- Linux
  - Basic Knowledges
  - Some Using Skills
- SeqPipe
  - What and Why?
  - How to use
  - Perspective

Part I. Linux

Linux

- Operating System
  - Process management (CPU)
  - Memory management (memory)
  - File System (storage)
  - Device Drivers (other hardwares)
  - Security (users / groups)



Frequently Used Commands

- File System:
  - ls / cd / pwd / cp / mv / rm / ln / mkdir / rmdir / chown / chmod
  - cat / head / tail / cut / paste / grep / find / diff / sort / uniq
  - tar / gzip / gunzip / gcat / bzip2 / bunzip2 / bzcat
  - sed / awk / perl / du / df / mount / umount
- Process & Memory management:
  - top / free / ps / kill / killall / time / nice / renice / screen
- Others
  - id / passwd / w / who / uptime / uname / date
  - ping / netstat / ssh
  - man / info / (*-h* / *--help*)

File Permissions

- Ownership: user, group, other (& all)
- Permission: read, write, execute
- Commands: id, ls -l, chmod, chown
- Examples:

```
$ ls -l
total 8
drwxr-xr-x 2 yanll bio 4096 Dec 18 07:04 dir
-rw------- 1 yanll bio    0 Dec 18 07:04 file
-rwxr-x--- 1 yanll bio    6 Dec 18 07:04 otherfile

$ chmod 600 file
$ chmod +x file
$ chmod o-rw file
$ chmod o=g file
```

Streams / Pipes / Redirects

- Input(<) / Output(>) / Append (>>) / Execute (|)
- /dev/stdin, /dev/stdout, /dev/stderr
- >, 2>, 2>&1, |, |&, <(...)
- Examples:

```
$ ls > files.txt
$ ls a* 2> error.txt
$ cat files.txt | sort
$ wc <(cat files.txt)
$ cat files.txt | gzip -c > files.txt.gz
```

Compress / Uncompress

- Gzip compression: gzip, gunzip, zcat
- Bzip2 compression: bzip2, bunzip2, bzcat
- Other compression: zip, rar, 7z, …
- Package: tar
- Examples:

```
$ zcat 1.fq.gz | head
$ less 1.fq.gz
$ unzip 1.fq.gz
$ gzip -9 1.fq
$ cat 1.fq | gzip -c -9 > 1.fq.gz
```

Environment Variables

- Display environment variables: set
- Export (set) variables: export *NAME*=*VALUE*
- Some useful variables: PATH, PERL5LIB, PS1, …
- Use the variables in command: $*NAME* or ${*NAME*}

Bash Expansion

- Wildcard: *, ?
- Numbers or Letters: {0..5}, {08..12}, {a..g}
- Strings with minor differences: {D,R}NA.txt
- Expression expansion: $((1 + 3 / 2))
- Bash command expansion: $(ls), `pwd`

Condition & Loop

- for / while / do / done
- if / then / else / elif / fi
- command1 && command2 / command1 || command2
- Example:

```
$ cat para_run.sh
#!/bin/bash

if [ -z "$1" ]; then
        echo "Usage: <STDIN> | prog INT_cpu"
        exit 1
fi
cpu=$1

while read c; do
        /bin/bash -c "$c" &
        r=`jobs | wc -l`
        while [ $r -ge $cpu ]; do
                sleep 0.001
                r=`jobs | wc -l`
        done
done
wait
exit 0
```

(Code of para_run.sh is from GaoG)

awk / sed / perl

To list all files that size is less than 1024 (bytes):

```
$ ls -l | awk '$5<1024'
```

To count how many bases have been sequenced:

```
$ zcat 1.fq.gz | sed -n '2~4p' | wc
    332     332   33532
$ echo $((33532 – 332))
33200
```

To calculate average length of reads:

```
$ zcat 1.fq.gz | sed -n '4~4p' | perl -e 'while(<>)
{ $l+=length($_)-1; $c++; } print $l / $c, "\n";'
100
```

Other Tips

- Run in background: nohup, screen
- Threads: lscpu, top, nice, renice
- Memory: free, top
- Disk quota: quota

Part II. SeqPipe

What is SeqPipe

- **Seq**uencing data analysis **Pipe**line framework
  - Hosted on http://seqpipe.googlecode.com/
  - Current version: 0.4.7 ($Rev: 231 $)
- SeqPipe:
  - Is text-based user interface
  - Runs work flow with logs
  - Integrates tools easily

Why SeqPipe

- Goals
  - Reproducibility
  - Expressiveness
  - Convenience

Why SeqPipe

- Goals
  - Reproducibility
  - Expressiveness
  - Convenience
- To achieve them
  - Record “everything” experienced
  - Hide “everything” unrelated
  - Enable “everything” tunable

Before SeqPipe

- Bad Practice
  - Run command directly
  - Change scripts frequently
  - Program version unrecorded
  - Messy code for checking & recording
  - …

Messy Code Example

```
date >>run.log 2>>run.err
echo “bwa aln hg19.fa 1.fq.gz >1.sai” >>run.log
bwa aln hg19.fa 1.fq.gz >1.sai 2>>run.err &
echo “bwa aln hg19.fa 2.fq.gz >2.sai” >>run.log
bwa aln hg19.fa 2.fq.gz >2.sai 2>>run.err &
wait
date >>run.log 2>>run.err
echo “bwa sampe hg19.fa 1.sai 2.sai 1.fq.gz 2.fq.gz >out.sam” >>run.log
bwa sampe hg19.fa 1.sai 2.sai 1.fq.gz 2.fq.gz >out.sam 2>>run.err
date >>run.log 2>>run.err
```

Other Solutions

- Makefile (with *-j <thread_num>*)
- Bpipe
  - http://code.google.com/p/bpipe/
- Snakemake
  - http://code.google.com/p/snakemake/
- GATK-Queue
  - http://www.broadinstitute.org/gatk/guide/article?id=1306
- Galaxy
  - https://main.g2.bx.psu.edu/
- WebLab
  - http://weblab.cbi.pku.edu.cn/
- …

Since SeqPipe

- SeqPipe Features:
  - Records (commands, versions, time, logs, …)
  - Checks every step if the command succeeded
  - Auto checks result files, skips already-finished steps
  - Tunable options
  - Parallel mode
  - …

How to Install SeqPipe

- Download the code
  - Download the .tar.gz file and uncompress
    - https://code.google.com/p/seqpipe/downloads/list
  - Checkout from Subversion
    - https://code.google.com/p/seqpipe/source/checkout
- Add /*path*/*to*/seqpipe/ to PATH
- Customize for bioseq.pipe (create bioseq.pipe.conf):

```
$ cat bioseq.pipe.conf
JAVA_MAX_MEM_SIZE=8G

REF_DIR=/rd/data/genomes/human/GRCh37
REF_NAME=human_g1k_v37

PICARD_ROOT=/rd/build/picard-tools
MAX_RECORDS_IN_RAM=2000000

GATK_ROOT=/rd/build/gatk
GATK_BUNDLE_ROOT=/rd/data/public/gatk_bundle/1.5/b37
```

- Run "seqpipe bioseq_syscheck" to check the configure

Basic Usage

- Run in Linux command console
- Type “seqpipe” to see help
  - `seqpipe [<options>] <procedure> [NAME=VALUE ...]`
- Save logs into .seqpipe/ of current directory
  - Index from history.log
- Write custom pipeline like bash script
  - Almost compatible with bash
  - See *demo.pipe*

Demo (1) – Hello, world!

- Command
  - `seqpipe -m demo.pipe demo_001`
- To learn
  - Options (`-m`, `-l`)
  - Procedure definition
  - Single command
  - Check log files

Demo (2) – More Commands

- Command
  - `seqpipe -m demo.pipe demo_002`
- To learn
  - *stderr* & pipe
  - Multiple commands
  - Multiple lines (ended with '\')
  - Option (-T)

Demo (3) – Return Value

- Command
  - `seqpipe -m demo.pipe demo_003`
- To learn
  - Return value checking
  - It fails when return non-zero

Demo (4) – Logical Operations

- Command
  - `seqpipe -m demo.pipe demo_004`
- To learn
  - Logical operations in bash
  - Only final return value is used for checking

Demo (5) – Complex Bash

- Command
  - `seqpipe -m demo.pipe demo_005`
- To learn
  - Use '\' to merge lines to bash
  - Notice the tailing ';'
  - Notice the variable names

Demo (6) – A Real “Pipeline”

- Command
  - `seqpipe -m demo.pipe demo_006`
- To learn
  - A simple example to start learning SeqPipe
  - We will try improve it better and better...

Demo (7) – Run in Parallel

- Command
  - `seqpipe -m demo.pipe demo_007`
- To learn
  - Parallel mode ({{, }})

Demo (8) – Complex Blocks

- Command
  - `seqpipe -m demo.pipe demo_008`
  - `seqpipe -m demo.pipe demo_008 -t 3`
- To learn
  - Nested blocks (parallel or sequential)
  - Option (-t)

Demo (8.5) – Inline Mode

- Command
  - `seqpipe -e "rm -fv 1.fq.gz.sai"`
- To learn
  - It is a good habit to keep everything **traceable**!

Demo (9) – Inputs & Outputs

- Command
  - `seqpipe -m demo.pipe demo_009`
- To learn
  - Set attributes (require/input/output) for commands & procedures
  - Options (`-h`)

Workflow Attributes

- **require**: File should exist
- **input**: File should exist and not newer than *output*
- **output**: Run only if file does not exist or older than any *input*
- **output.temp**: Same as *output*, will be removed when procedure finished

Demo (10) – Improve Workflow

- Command
  - `seqpipe -m demo.pipe demo_010`
- To learn
  - Process attributes (require/input/output) could be inferred from commands
  - Options (`-h, -H`)

Demo (11) – Variables

- Command
  - `seqpipe -m demo.pipe demo_011 REF=MT.fa FQ_1=1.fq.gz FQ_2=2.fq.gz OUTPUT=out`
- To learn
  - Options (`-h`)
  - Make things tunable!
  - Make pipeline re-usable!

Demo (12) – Sub Procedure

- Command
  - `seqpipe -m demo.pipe demo_012 REF=MT.fa FQ_1=1.fq.gz FQ_2=2.fq.gz OUTPUT=out`
- To learn
  - `SP_run <procedure> [NAME=VALUE ...]`

Demo (13) – Default Value

- Command
  - `seqpipe -m demo.pipe demo_013 OUTPUT=out2`
- To learn
  - SP_set NAME=value
  - Global variables
  - Options (-h, -H)

Demo (14) – Condition

- Command
  - `seqpipe -m demo.pipe demo_014`
  - `seqpipe -m demo.pipe bwa_index_ex -h`
- To learn
  - SP_if
  - Variable name which starts with '_'

Further Usage

- `SP_set X=value`
  - Value: strings / wildcard / number serial / letter serial / expression
  - Example:
    - `SP_set X=abc *.gz {1..5} $((3+4))`
- `SP_if` *cond* / `SP_else_if` *cond* / `SP_else`
  - Condition: value / (*bash*) / !(*bash*)
- `SP_for _X=value`
- `SP_for_parallel _X=value`
- `SP_while` (*bash*)

For NGS

- Type “seqpipe -l”
- For example:
  - DNAseq_analysis
  - fastqc_check
  - bowtie2_map_pe
  - bwa_map_pe
  - convert_fastq_33to64
  - convert_fastq_64to33
  - gatk_call_variants
  - pindel_call_structure_variants
  - mkdup_bam
  - …

Summary

- SeqPipe provides a convenient way to represent and track analysis procedure, therefore, to ensure the reproducibility
- SeqPipe will also include more regular pipelines for NGS data analysis (ongoing)

Thank You!

Questions or Suggestions?

http://seqpipe.googlecode.com/

yanll@mail.cbi.pku.edu.cn