SIRE511 : LINUX AND BIOINFORMATICS DATA SKILLS

Bioinformatics Data Skill I:

6th Week, 15/10/2024

Kwanrutai Mairiang, Ph.D

Outline

- Setting up and Managing a Bioinformatics Project
  - Project Directories and Directory structures
  - Naming files and folders
  - PATHs in project management
  - Project Documentation
  - Organizing data to automate file processing task
- Essential Shell Command Recap
  - Combining Pipes and Redirection
  - A tee in Your Pipe
  - Exit Status: How to Programmatically Tell Whether Your Command Worked
  - Background Processes
  - Maintaining Long-Running Jobs with nohup and tmux

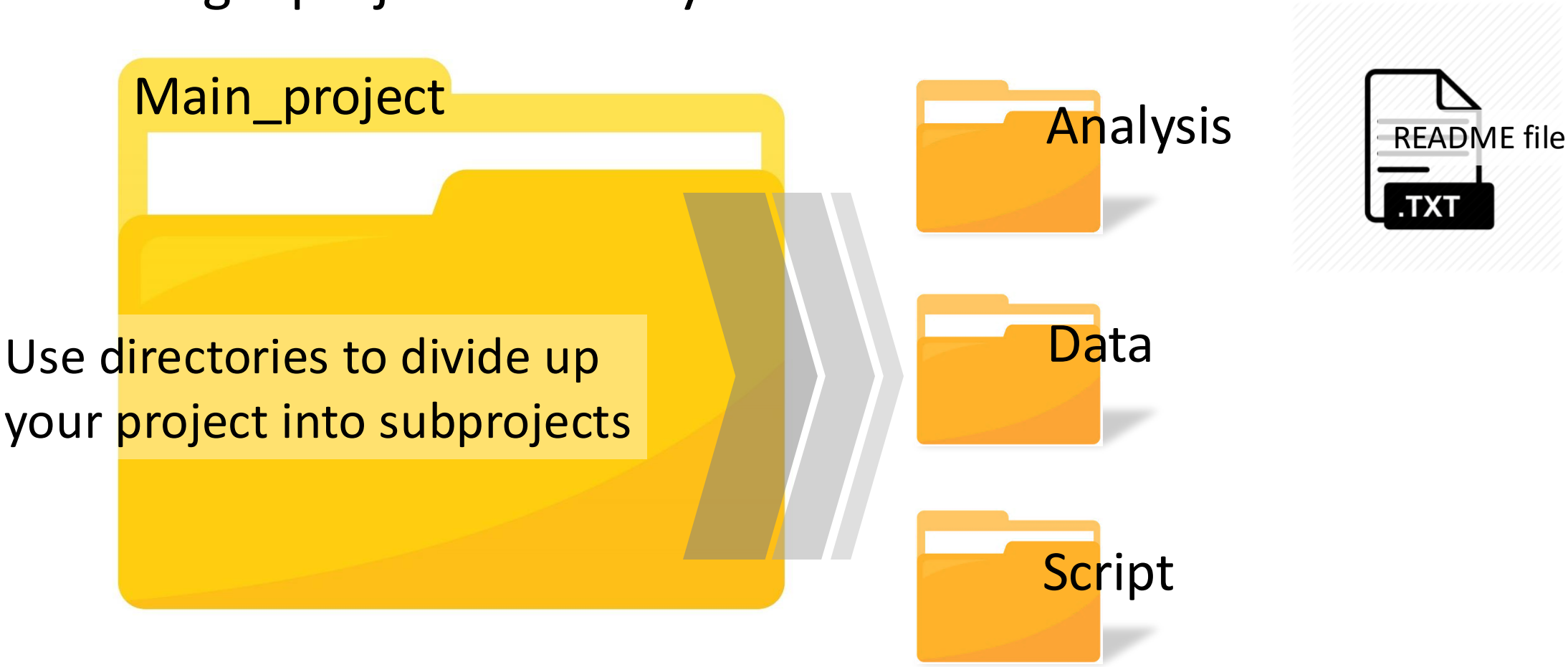
Setting up and Managing a Bioinformatics Project

Why is project management important in bioinformatics?

- Easy to work with a team or collaborater.
  - Sharing the data:
    - Input
    - Protocol
    - Observation
    - Finding
    - The reason of experimental fail
    - Output
  - Tracking the analysis version
- Project reproducible
- Easier to automate task when files are organized and clearly named.

Project Directories and Directory structures

- All files and directories used in your project should live in a single project directory with a clear name.



Project Directories and Directory structures

- All Raw and intermediate data
- Data-processing steps are treated as separate subdirectory in this *data/* directory.

- Keep general project-wide scripts
- If scripts have lots of module files, put them in their own folder.

- Store each analysis step in a separate subdirectory.

What's in a name?

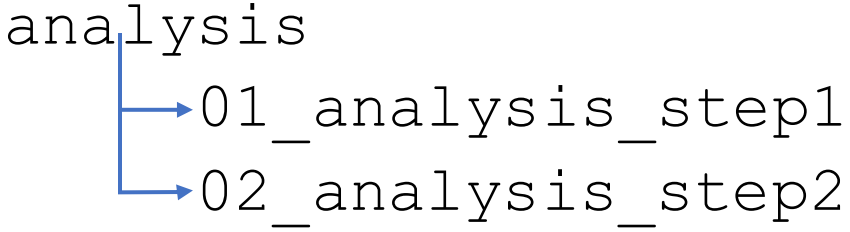
- Avoid space!
- It’s best to use only letters, numbers, underscores, and dashes in file and directory names.
- What happen if want to remove folder “raw sequence” with the following command?

```
rm -rf raw sequence
```

- The file extension is not require in Unix. However, file extension help to indicate the type of each file.

Folder Naming

- Number the folders in order of their creation
  - Numbers starting with 00, 01, 02, … , 99 will keep the files/folders in numerical order when listed using the `ls -l` command.



- The project's folder name should clearly describe the project.
  - Year of starting project
  - Name of researcher
  - Short project description

```
2023_Kwanrutai_DenvRNASeq
2024_Kwanrutai_DenvThGWAS
```

PATHs in project management

- Scripts and analyses often have to point to other files in your project hierarchy, like data. In such situations, it's crucial to use relative paths instead of absolute paths.
- As long as your project's internal directory structure doesn't change, these relative paths will consistently function.

Project Documentation

- A good bioinformatician will always document everything extensively and use clear filenames that can be parsed by a computer.
- Poor documentation can lead to irreproducibility and serious errors.

What exactly should you document?

Project Documentation

1. Document your methods and workflows

- Document all steps of a method or draw a workflow.
- Document full command lines (copied and pasted) that are run through the shell.
  - All the command lines that are generated data or intermediate result.
- Document all values in options and arguments that are used with software, even if you used the default.
- Document any command-line options used to run the script.
- Version control software, such as GitHub and Conda, can be used for managing the versions of your code.

Project Documentation

2. Document the origin of all data in your project directory

- Keep track of where data form:
  - Downloaded data: External data's source
  - Provided by collaborator: who gave it to you, and any other relevant information.
  - Create metadata

Project Documentation

3. Document when you downloaded data

- It is important to document when downloading data from an external data source because the source might change or be updated in the future.
  - Website or server URL
  - Download date
  - Record data version information including minor version number
  - Describe how you downloaded the data, such as:
    - Script used for download
    - Genome Browser

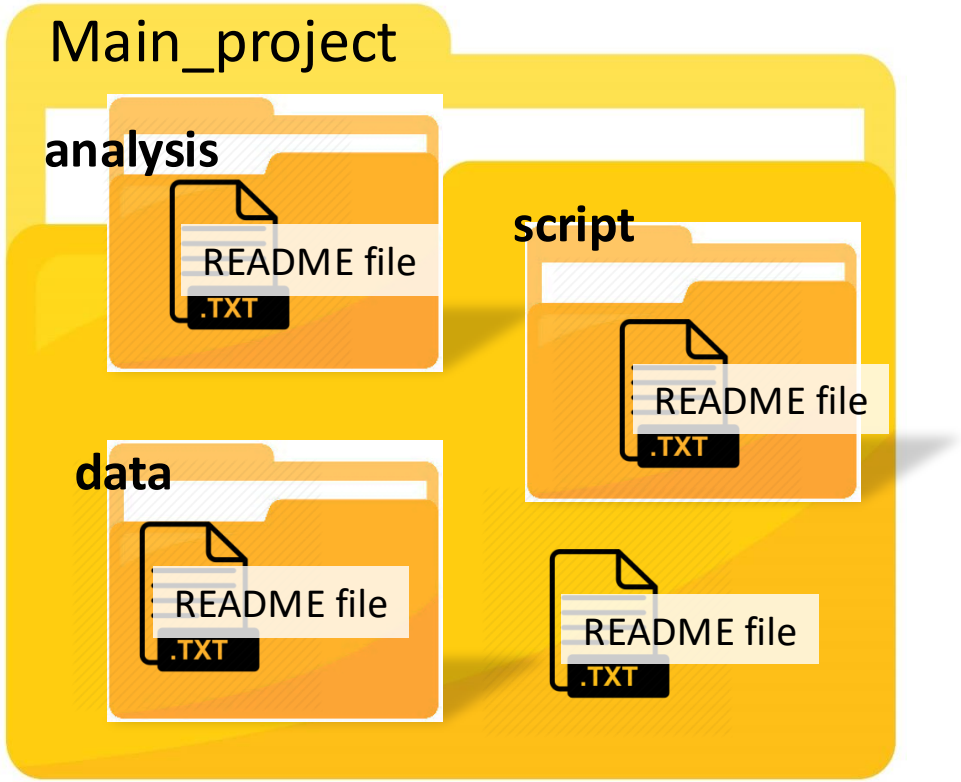
Project Documentation

4. Document the versions of the software that you ran

- The analysis results from different versions of the software may yield differing results.
- Good bioinformatics software usually has a command-line option to return the current version.
- Software management systems like Git provide explicit identifiers for every version, which can be used to document the software's version.
- If no version information is available, a release date, link to the software, and download date will suffice.

Project Documentation

- All the document is best stored in plain-text "README" files.
  - Plain-text can easily be read, searched, and edited directly from the command line. It's
  - Plain text can be accessed on all computer systems.
  - Plain text also lacks complex formatting, which can create issues when copying and pasting commands.
- A good approach is to keep README files in each of your project's main directories.
  - For example, a data/README file would contain metadata about your data files in the data/ directory.




```
$touch README data/README
```

Use Directories to Divide Up Your Project into Subprojects

- Bioinformatics projects involve many subprojects and subanalyses.
- Creating directories to logically separate subprojects can simplify complex projects and help keep files organized.

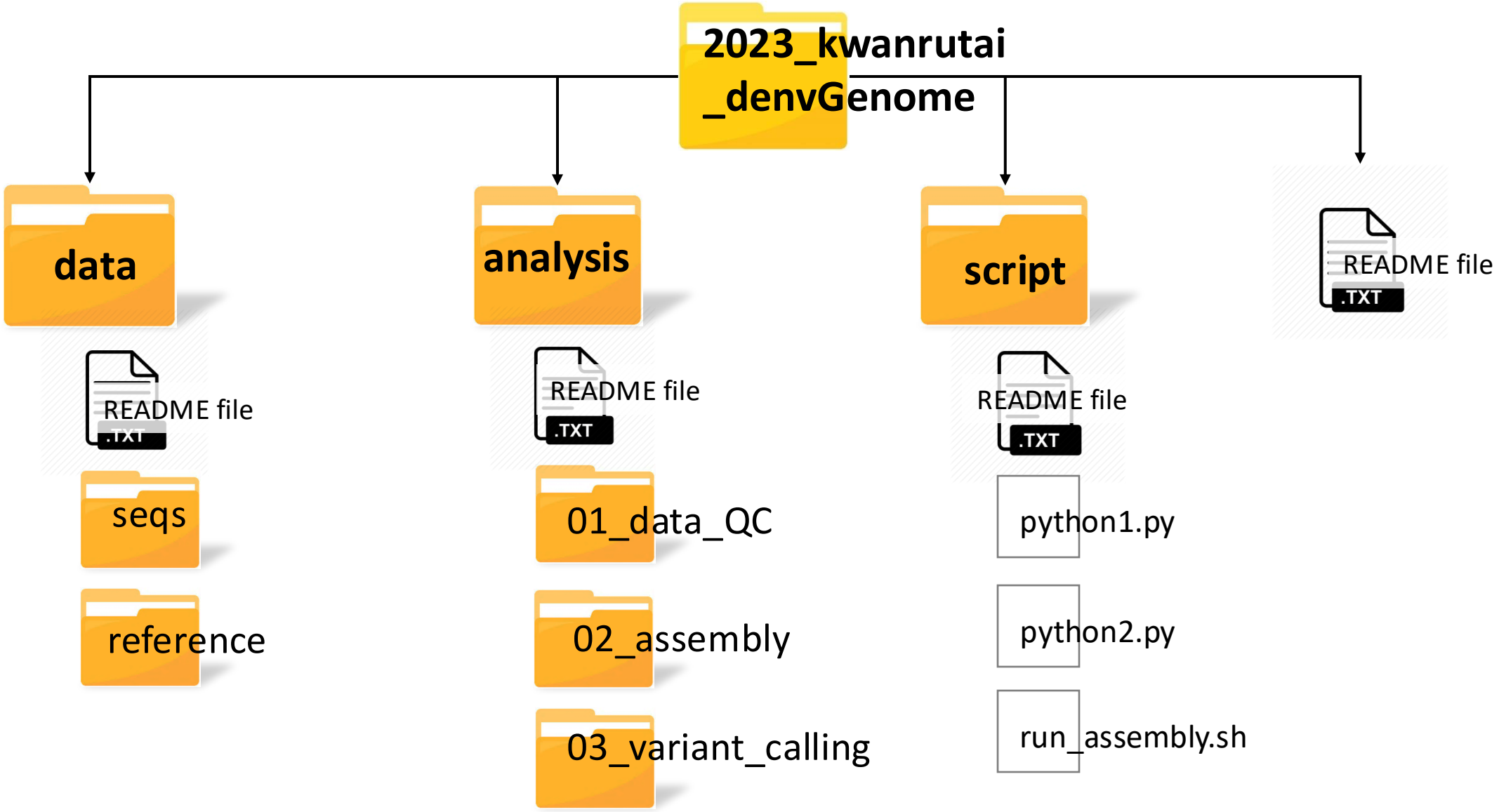
analysis/01_data_QC

analysis/02_ref_alignment

analysis/03_assembly

analysis/04_variant_calling

Example of directories structure



Organizing data to automate file processing task

- Automating file processing tasking is an integral part of bioinformatics.
- Organizing data into subdirectories and using clear and consistent file naming schemes allow us to programmatically refers to files.
- Programmatically referring to multiple files is easier and safer than typing them all out (because it’s less error prone).

How does consistent file naming help automate file processing?

Shell expansion Tips

1. Brace expansion

Create folders using this command

```
$ mkdir -p denv/{data/seqs,scripts,analysis}
```

Try running the following commands:

```
$ echo dog-{gone,bowl,bark}
$ touch data/seqs/denv{1,2,3,4}_R{1,2}.fastq
$ ls data/seqs/denv[1-4]_R[1-2].fastq
$ touch data/seqs/zmays{A..C}_R{1..2}.fastq
```

How does consistent file naming help automate file processing?

Shell expansion Tips

2. Common Unix filename wildcards

Wildcard	What it matches
*	Zero or more characters (but ignores hidden files starting with a period).
?	One character (also ignores hidden files).
[A-Z]	Any character between the supplied alphanumeric range (in this case, any character between A and Z); this works for any alphanumeric character range (e.g., [0-9] matches any character between 0 and 9).

How does consistent file naming help automate file processing?

Shell expansion Tips

3. Leading zeros and sorting

Naming files with leading zeros ensures the correct order when using the 'ls' command to list files.

```
$ touch test_{1,2,11,12,13}.txt
$ ls -l

$ touch test_{001,002,011,012,013}.txt
$ ls -l
```

Shell expansion Tips

2. Common Unix filename wildcards

Running the following commands:

```
$ ls data/seqs/denv*
```

```
$ ls data/seqs/zmays?_R1.fastq

$ ls data/seqs/zmays[A-C]_R1.fastq

$ ls data/seqs/zmays[A-C]_R?.fastq
```

Practical:

Write a shell script to create directories and move fastq files to their corresponding directories in data/seqs.

Hint: edit from “movefile.sh”

Essential Shell Command Recap

Pipes in Action: Creating Simple Programs with Grep and Pipes

- This is an example of how to use the pipe along with the 'grep' command to create a small program for identifying non-'ATCG' nucleotides in a fasta sequence.

```
$ grep -v "^>" tb1.fasta | \
grep --color -i "[^ATCG]"
```

Combining Pipes and Redirection

- Use the pipe (|) and redirection symbols to create a pipeline that passes the output from the previous process to be the input of the next one, and also collects the error messages in a file.

```
$ program1 input.txt 2> program1.stderr | \
  program2 2> program2.stderr > results.txt
```

- Redirect standard error to become the standard output using the '2>&1' symbol.

```
$ program1 2>&1 | grep "error"
```

Even More Redirection: A tee in Your Pipe

- The 'tee' command is used to divert a copy of the standard output to an intermediate file while still passing it to the next process as input.

```
$ program1 input.txt | tee intermediate-file.txt \
| program2 > results.txt
```

Background Processes

- To run a program in the background, simply append '&' to the end of the command.

```
$ program1 input.txt > results.txt &
[1] 26577     ## job IDs and PID
```

- To check what processes are running in the background using ‘jobs’ command.

```
$ jobs
[1]+  Running program1 input.txt >
results.txt    ## Return the running jobs
```

Background Processes

- To bring a background process into the foreground using ‘fg’ command.

```
$ fg
program1 input.txt > results.txt
```

  - ‘fg’ will bring the most recent process to foreground. To return the specific background job using `$ fg %Job_IDs`

- To place a process running on foreground to background using ‘bg’ command

```
$ program1 input.txt > results.txt # forgot to append ampersand
$ # enter control-z to stop the process
[1]+  Stopped                 program1 input.txt > results.txt
$ bg
[1]+ program1 input.txt > results.txt
```

  - To place the specific jobs to background using `$ bg %Job_IDs`

Exit Status: How to Programmatically Tell Whether Your Command Worked

- Shell will set the exit status value to a variable `$?`
- To check the exit status using 'echo' command

```
$ program1 input.txt > results.txt
$ echo $?
0
```

  - A zero exit status means the process has finished without any errors.
- Even when a zero exit status is returned, it's better to always check your intermediate data to ensure that your job has completed without errors.

Naming directories with current date (today)

- Adding the date to directory names is another way to manage your results when you can generate more than one version in a project.

```
$ mkdir results-$(date +%F)
$ ls results-2023-10-16
```

- Use the 'alias' command to create shorter names for long sets of frequently used commands.
  - Set alias ‘today’ for ‘date +%F’

```
$ alias today="date +%F”
```

  - Now, make directory with date should be:

```
$  mkdir result-$(today)
```

  - To permanently set the alias, add the command to the ~/.bashrc file.

Maintaining Long-Running Jobs with nohup and tmux

nohup

- Insatll nohub

```
$ apt install nohub
```

- Running nohub

```
$ nohup program1 > output.txt &
[1] 10900  ## Return Job ID and PID
```

Working with Remote Machines Through Tmux

- Install Tmux
  - $ apt install tmux
- Create new Tmux session
  - $ tmux new-session -s session_name
- Detach the session use *Control-a d*
- List all sessions in tmux
  - $ tmux list-sessions
- Reattach the session
  - $ tmux attach-session -t session_name

Common Tmux key sequences

Key sequence	Action
Control-a d	Detach
Control-a c	Create new window
Control-a n	Go to next window
Control-a p	Go to previous window
Control-a &	Kill current window (exit in shell also works)
Control-a ,	Rename current window
Control-a ?	List all key sequences

Common Tmux subcommands

Subcommand	Action
tmux list-sessions	List all sessions.
tmux new-session -s session-name	Create a new session named “session-name”.
tmux attach-session -t session-name	Attach a session named “session-name”.
tmux attach-session -d -t session-name	Attach a session named “session-name”, detaching it first.