Unix miscellenous topics

Jon Chernus (adapted from Ryan Minster)

Department of Human Genetics
School of Public Health
University of Pittsburgh

Document created: September 29, 2024

Location

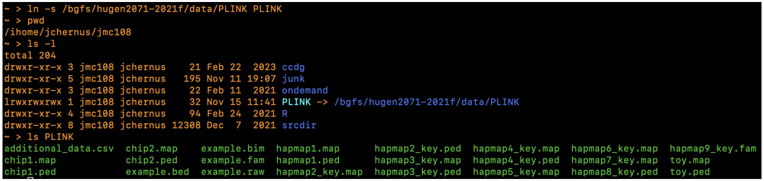
This slide set is called `unix_miscellaneous` and is located in the "20_unix_miscellaneous" folder of our Lectures repository.

Objectives

- To learn how to parallelize functions in Unix
- To learn a few other useful Unix commands

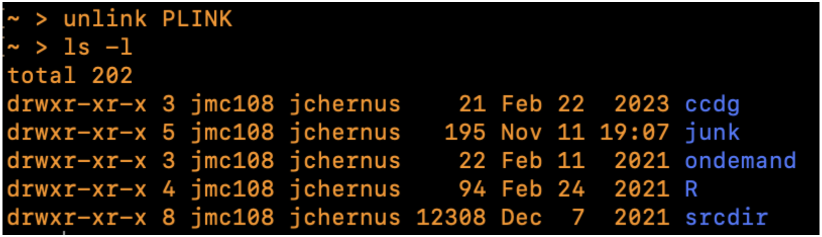
Making symbolic links with `ln -s`

- Symbolic links are shortcuts to files or directories
- Syntax: `ln -s path_to_target_file_or_folder shortcut_name`
- Makes a shortcut to the target file/folder in your working directory
- Suppose I get tired of typing out the path to the homework data folder, `/bgfs/hugen2071-2021f/data/PLINK/`



```
~ > ln -s /bgfs/hugen2071-2021f/data/PLINK PLINK
~ > pwd
/ihome/jchernus/jmc108
~ > ls -l
total 204
drwxr-xr-x 3 jmc108 jchernus    21 Feb 22  2023 ccdg
drwxr-xr-x 5 jmc108 jchernus   195 Nov 11 19:07 junk
drwxr-xr-x 3 jmc108 jchernus    22 Feb 11  2021 ondemand
lrwxrwxrwx 1 jmc108 jchernus    32 Nov 15 11:41 PLINK -> /bgfs/hugen2071-2021f/data/PLINK
drwxr-xr-x 4 jmc108 jchernus    94 Feb 24  2021 R
drwxr-xr-x 8 jmc108 jchernus 12308 Dec  7  2021 srcdir
~ > ls PLINK
additional_data.csv  chip2.map    example.bim  hapmap1.map      hapmap2_key.ped  hapmap4_key.map  hapmap6_key.map  hapmap9_key.fam
chip1.map            chip2.ped    example.fam  hapmap1.ped      hapmap3_key.map  hapmap4_key.ped  hapmap7_key.map  toy.map
chip1.ped            example.bed  example.raw  hapmap2_key.map  hapmap3_key.ped  hapmap5_key.map  hapmap8_key.ped  toy.ped
```

Removing symbolic links with `unlink`



```
~ > unlink PLINK
~ > ls -l
total 202
drwxr-xr-x 3 jmc108 jchernus    21 Feb 22  2023 ccdg
drwxr-xr-x 5 jmc108 jchernus   195 Nov 11 19:07 junk
drwxr-xr-x 3 jmc108 jchernus    22 Feb 11  2021 ondemand
drwxr-xr-x 4 jmc108 jchernus    94 Feb 24  2021 R
drwxr-xr-x 8 jmc108 jchernus 12308 Dec  7  2021 srcdir
```

Checking storage space with du

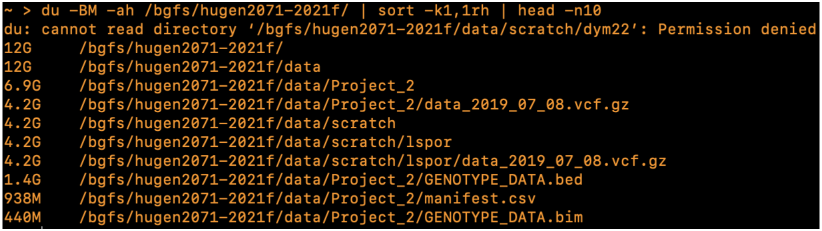
du recursively shows the size (in blocks) of all subdirectories in a directory

- -a to show all files
- -s to summarize each directory's size
- -h for human-readable output
- How big is my working directory? Use du -sh .
- How big are all the files and folders (nested) in my working directory? Use du -ah .

```
du -ah .
4.0K    ./header_pagenrs.tex
8.0K    ./.DS_Store
208K    ./images/symbolic_link_example.png
8.0K    ./images/.DS_Store
296K    ./images/du_sorted_example.png
 12K    ./images/image-1535965655.png
128K    ./images/unlink_example.png
652K    ./images
4.0K    ./scripts/example.R
4.0K    ./scripts
4.0K    ./data/letters.txt
4.0K    ./data/letters_shortened.txt
4.0K    ./data/letters_copy.txt
 12K    ./data
568K    ./unix_miscellaneous.pdf
8.0K    ./unix_miscellaneous.Rmd
1.2M    .
```

Making `du` more useful

- `-B` to change block size (K, M, G, T, P, etc.)
- pipe to `sort -h` and list biggest files first



```
~ > du -BM -ah /bgfs/hugen2071-2021f/ | sort -k1,1rh | head -n10
du: cannot read directory '/bgfs/hugen2071-2021f/data/scratch/dym22': Permission denied
12G     /bgfs/hugen2071-2021f/
12G     /bgfs/hugen2071-2021f/data
6.9G    /bgfs/hugen2071-2021f/data/Project_2
4.2G    /bgfs/hugen2071-2021f/data/Project_2/data_2019_07_08.vcf.gz
4.2G    /bgfs/hugen2071-2021f/data/scratch
4.2G    /bgfs/hugen2071-2021f/data/scratch/lspor
4.2G    /bgfs/hugen2071-2021f/data/scratch/lspor/data_2019_07_08.vcf.gz
1.4G    /bgfs/hugen2071-2021f/data/Project_2/GENOTYPE_DATA.bed
938M    /bgfs/hugen2071-2021f/data/Project_2/manifest.csv
440M    /bgfs/hugen2071-2021f/data/Project_2/GENOTYPE_DATA.bim
```

Running an R script with `Rscript`

- First, on htc, you need to load R (and a C/C++ compiler)
  `module load gcc/12.2.0 r/4.3.0`
- If you're using a Slurm script that calls your R script, you need to include that `module load` line in the Slurm script
- Run the R script with a command like `Rscript --vanilla your_R_script.R`
  (`--vanilla` avoids restoring workspaces, etc.)
- Printed output from the R script goes to the Slurm log file, unless you've redirected it
  (e.g., `Rscript your_R_script.R > my_r_script_log.txt`)
- Good idea: include `options(echo = TRUE)` at the top of your R script so that the output will include your R commands (helps for reading output and debugging)

Running an R script with command-line arguments

To pass variables to your R script, just put a space-separated list of the form `variable_name=value` in your `Rscript` command. Note that for character variables, quotes need to be escaped with \.

Example of running an R script with 3 command-line arguments:

```
Rscript my_r_script.R a=1 b=2 c=\"abc\"
```

To access the variables inside the body of the R script, you need to do this:

- Read in the arguments:
  ```
  args <- (commandArgs(T))
  ```
- And then assign them:
  ```
  for (i in 1:length(args)) { eval(parse(text =
  args[[i]])) }
  ```

Example R script (./scripts/example.R)

```r
# Make it so that commands are echoed in the output
options(echo = TRUE)

# Get the command-line arguments
args <- commandArgs(T)

# Notice: the values are NOT actually assigned!
# We just have the "ingredients" for making them
ls()
args

# Now take the text in args and execute it as commands
for (i in 1:length(args))
  {
  eval(parse(text = args[[i]]))
}

# The variables exist now:
ls()

# We can use them:
a + b
c
```

Running the example script

```
# module load gcc/12.2.0 r/4.3.0 would go here if this were in a Slurm script
Rscript --vanilla ./scripts/example.R a=1 b=3.14 c=\"Hello\"
```

```
>
> # Get the command-line arguments
> args <- commandArgs(T)
>
> # Notice: the values are NOT actually assigned!
> # We just have the "ingredients" for making them
> ls()
[1] "args"
> args
[1] "a=1"          "b=3.14"       "c=\"Hello\""
>
> # Now take the text in args and execute it as commands
> for (i in 1:length(args))
+   {
+   eval(parse(text = args[[i]]))
+ }
>
> # The variables exist now:
> ls()
[1] "a"    "args" "b"    "c"    "i"
>
> # We can use them:
> a + b
[1] 4.14
> c
[1] "Hello"
>
```

Parallel computing

- Often a larger computing task can be divided into smaller parts
- If the parts are independent, they can be run in parallel, simultaneously
  - E.g., each chromosome in a GWAS can be run separately and concurrently
  - E.g., aligning sequencing reads to a reference sequence
- Some programs offer to do this for you; sometimes you need to implement it yourself

Implementing parallelism with job arrays

Suppose you have a Slurm script that performs a task for one chromosome...

```
#!/bin/bash
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=username@pitt.edu
#SBATCH -t 1:00:00
set -euo pipefail

awk '$6 = 2 { print $1, $2, $3, $4, $5, $6, $7, $8 }' hapmap1.ped > ~/hapmap1_new.ped
```

... but it turns out there are 22 files: `hapmap1_1.ped`, `hapmap1_2.ped`, `hapmap1_3.ped`, ... , and `hapmap1_22.ped`

Implementing parallelism with job arrays

Use a job array, where the array index ranges from 1 to 22. Inside the script, you can use the variable ${SLURM_ARRAY_TASK_ID} to perform the tasks in parallel:

```
#!/bin/bash
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=username@pitt.edu
#SBATCH -t 5:00
#SBATCH -J newpeds
#SBATCH --output=newpeds-%A_%a.out
#SBATCH --array=1-22
set -euo pipefail

awk '$6 = 2 { print $1, $2, $3, $4, $5, $6, $7, $8 }' \ hapmap1_${SLURM_ARRAY_TASK_ID}.ped > \
~/hapmap1_${SLURM_ARRAY_TASK_ID}_new.ped
```

In `--output=newpeds-%A_%a.out`, `%A` gets replaced with the job ID nd %a gets replaced with the array index.

What if the tasks aren't naturally numbered?

Suppose we have `hapmap1_1.ped`, `hapmap1_2.ped`, `hapmap1_3.ped`, . . . , `hapmap1_22.ped`, and `hapmap1_X.ped`. The array indices have to be integers, and X is not an integer.

The solution is to use this trick:

```
#!/bin/bash
...
#SBATCH --array=1-23
set -euo pipefail

# For array index i, this grabs the name of the i-th file
file=`ls hapmap1_*.ped | head -n $SLURM_ARRAY_TASK_ID | tail -n 1`

# Now do the task to that file
awk '$6 = 2 { print $1, $2, $3, $4, $5, $6, $7, $8 }' \   $file > ~/${file}_new.ped
```