UNIX basics

Jon Chernus (adapted from Ryan Minster)

Department of Human Genetics
School of Public Health
University of Pittsburgh

Document created: September 23, 2024

Location

This slide set is called `unix_basics.pdf` and is located in the "15_unix_basics" folder of our Lectures repository.

Objectives

- To learn basic UNIX commands
- To learn how to interact with running processes

Most of the content in this slide set is essential. You will need to become proficient with it to proceed in the course and to work at the command line in general.

Most of this is a review of what you read in the assigned Active Reading.

Warnings

- `rm` and `rmdir` are forever
- overwriting is forever (so be careful with `cp`, `mv`, etc.)
- a single whitespace matters a lot
    - `rm -rf a*`
        - recursively delete any file/folder starting with an a
    - `rm -rf a *`
        - recursively delete any file/folder starting with an a
        - then do the same for *everything* (because `*` matches anything)
- these are all different, so be careful when copy-pasting
    - ",", and " (straight vs. curly double-quotes)
    - ',',', and ' (straight vs. curly single-quotes vs. back-tick)
    - -, –, and — (hyphen, en dash, em dash)
- don't use spaces and special characters in file names
- don't work on the login node of htc (use `srun -M teach -A hugen2071-2024f --pty bash`)

What is a shell?

- User interface for interacting with the UNIX operating system
- A program that accepts commands that are in turn other, more basic programs/utilities
- Important for scripting, making pipelines, calling other programs, organizing files, managing resource-intensive processes/jobs
- There are lots of versions
- We use `bash` here

The bash shell

- Very common
- Supports history (use the `up arrow` to save time!)
- Supports autocompletion (use `tab` to save time!)
- Supports wildcards (more on that later)
- Comments start with #

Files are organized hierarchically

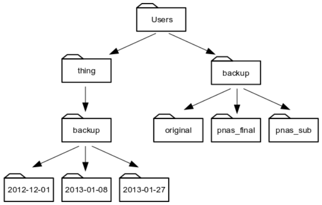


Figure 1: Example of a file tree (https://swcarpentry.github.io/shell-novice/02-filedir.html)

You can designate a file/folder with

- an absolute path (starting at the top of the file tree, root, /)
    - e.g., `/Users/thing/backup/2013-01-08/`
- a relative file path (starting at your working directory, .)
    - e.g., `../2013-01-08/` if you're currently in `/Users/thing/backup/2012-12-01/`

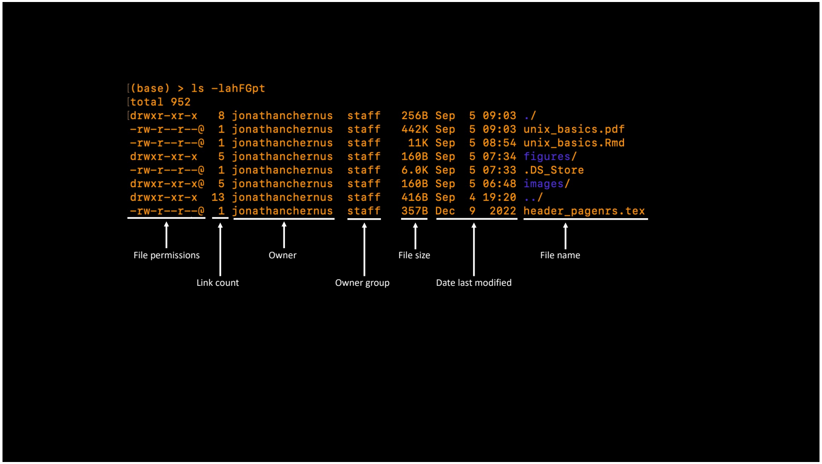
Shortcuts for navigating

Shortcut	Meaning
`~`	home directory
`.`	current/working directory
`..`	parent directory (up one from where you are now)
`-`	your previous working directory
`/`	• By itself: the root directory (top of the file tree) • Between names of folders: just a divider

Commands for navigating

Command	Meaning	Syntax	Useful options
`pwd`	print working directory	`pwd`	
`cd`	change directory	`cd directory`	
`ls`	list the files in a directory	`ls directory`	`-lahFGpt`

What does the output of `ls` mean?



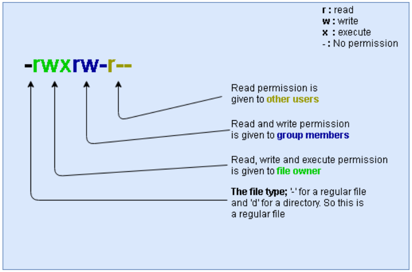
```
[(base) > ls -lahFGpt
[total 952
[drwxr-xr-x   8 jonathanchernus  staff   256B Sep  5 09:03 ./
 -rw-r--r--@  1 jonathanchernus  staff   442K Sep  5 09:03 unix_basics.pdf
 -rw-r--r--@  1 jonathanchernus  staff    11K Sep  5 08:54 unix_basics.Rmd
 drwxr-xr-x   5 jonathanchernus  staff   160B Sep  5 07:34 figures/
 -rw-r--r--@  1 jonathanchernus  staff   6.0K Sep  5 07:33 .DS_Store
 drwxr-xr-x@  5 jonathanchernus  staff   160B Sep  5 06:48 images/
 drwxr-xr-x  13 jonathanchernus  staff   416B Sep  4 19:20 ../
 -rw-r--r--@  1 jonathanchernus  staff   357B Dec  9  2022 header_pagenrs.tex
```

Figure 2: Columns of ls output

Commands for manipulating files

Command	Syntax	Meaning
`cp`	• `cp file1 file2 ... file5 dir` • `cp file1 file2`	• Copy one or more files into the folder • Copy file1 as file2
`mv`	• `mv file1 file2 ... file5 dir` • `mv file1 file2`	• Move one or more files into the folder • Rename file1 as file2
`rm -i`	`rm file1 file2 ... file5`	Permanently delete one or more files
`mkdir`	`mkdir dir1 dir2 ... dir5`	Create one or more folders
`rmdir`	`rmdir dir1 dir2 ... dir5`	Permanently delete one or more empty folders

Who can do what with a file?



r : read
w : write
x : execute
- : No permission
-rwxrw-r--
Read permission is given to other users
Read and write permission is given to group members
Read, write and execute permission is given to file owner
The file type; '-' for a regular file and 'd' for a directory. So this is a regular file

Figure 3: How to read file permissions (https://teaching.healthtech.dtu.dk/unix/index.php/File_permissions)

Permission code	Meaning
`r`	read file/list directory
`w`	edit file/make files in directory
`x`	execute file/cd into directory

Use `chmod` to control file permissions

Example command	Meaning
`chmod -v a=rwx file`	Give everyone (all) permission to read, write, and execute
`chmod -v ug=rw file`	Give yourself (the user/owner, u) and your group permission to read and write only
`chmod -v o=r file`	Give others read-only permission
`chmod -v a+x file`	Add (+) execute permission for all (without changing the `r` or `w` permissions)

`-v` means "verbose" and prints a useful message.

Commands for looking at contents of files

Command	Meaning	Syntax	Useful options
head	show first lines of file	head file	-n (first n lines)
tail	show last lines of file	tail file	-n (last n lines) or -n+ (start at line n)
more	read through file	more file	(Navigate with spacebar and q)
less	read through file	less file	(See documentation)
wc	get count of words, lines, and bytes	wc file	-l (get only word count)

More important commands: `cat`, `echo`, `grep`

Command	Use	Example	Meaning
`cat`	Prints contents of a file	`cat file1.txt file2.txt`	Print one or more files
`echo`	Prints out whatever follows	`echo "Hi there"`	Print "Hi there"
`grep`	Search for pattern in file • `-i` case-insensitive • `-v` invert (find non-matches) • `--color` highlight matches	`grep gene file.txt`	Print lines of file.txt containing "gene"

Use wildcards (?, *, and [...]) for pattern-matching

These are powerful - so be careful with them!

Wildcard	Matches	Example pattern	Example matches
`*`	Anything (zero or more characters)	`*.txt`	Any .txt file
`?`	Any single character	`chr1?.txt`	chr11.txt, chr12.txt, chr1a.txt (but not chr1.txt, chr111.txt, chr.txt)
`[...]`	Anything in the set of characters in the brackets	`chr[0-9][0-9].txt`	chrMN.txt where M and N are any integers 0-9
		`chr[XY].txt`	chrX.txt or chrY.txt

cat, echo, grep, wildcard examples

```
# echo example
echo "Echoing a comment to the screen. Then using a cat command!"
Echoing a comment to the screen. Then using a cat command!

# cat example
# print out some short text files
echo "Below are the contents of my chr*.txt files"
Below are the contents of my chr*.txt files
cat data/chr*.txt
chr1:36926582
chr1:66782904
chr1:77840389
chr2:60318540
chr2:85739014


# grep example
echo "Here are all the lines of chr1.txt without a 3"
Here are all the lines of chr1.txt without a 3
grep -v 3 data/chr1.txt
chr1:66782904
```

Using grep with regular expressions

Some more advanced pattern-matching options:

Regular expression	Use	Example	Meaning
`^`	Matches beginning of line	`grep "^chr1" file.txt`	Print lines of file that start with “chr1” (this would also match “chr12”, etc.)
`$`	Matches end of line	`grep "0$" file.txt`	Print lines of file that end with the digit 0
`[...]`	Matches lines containing characters listed inside the brackets	`grep "[ACGT]" file.txt`	Print lines of file that contain A, C, G or T
`[^...]`	Matches lines containing characters **not** listed inside the brackets	`grep "[^ACGT]" file.txt`	Print any lines of the file containing characters **other than** A, C, G, or T

grep examples with regular expressions

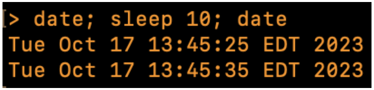
```
# Look at top of the file
head -n3 data3/tb1.fasta
>gi|385663969|gb|JQ900508.1| Zea mays subsp. mexicana isolate IS9 teosinte branched 1 (tb1) gene, complet
GCCAGGACCTAGAGAGGGGAGCGTGGAGAGGGCATCAGGGGGCCTTGGAGTCCCATCAGTAAAGCACATG
TTTCCTTTCTGTGATTCCTCAAGCCCCATGGACTTACCGCTTTACCAACAACTGCAGCTAAGCCCGTCTT

# Pull out just the header
grep "^>" data3/tb1.fasta
>gi|385663969|gb|JQ900508.1| Zea mays subsp. mexicana isolate IS9 teosinte branched 1 (tb1) gene, complet

# Pull out any line containing a non-ATGC character
grep --color -i "[^ACGT]" data3/tb1.fasta
>gi|385663969|gb|JQ900508.1| Zea mays subsp. mexicana isolate IS9 teosinte branched 1 (tb1) gene, complet
CCCCAAAGACGGACCAATCCAGCAGCTTCTACTGCTAYCCATGCTCCCCTCCCTTCGCCGCCGCCGACGC
```

Processes

- A command you run is run as a "process" and assigned a process ID number (pid)
- By default a process runs in the **foreground** (you wait for it to finish before starting another)
- To keep interacting with the terminal, you can run a process in (or move it to) the **background**
  - Run it in the background by putting & after the command (this prints job id and pid)
  - To move a process from foreground to background
    - `ctrl` + `z` suspends the process
    - then `bg` to restart it in the background
  - Use `jobs` to see what jobs are running in the background (shows the commands)
  - Use `ps` to see process status (including pid)
  - Terminating a process
    - `ctrl` + `z` for a foreground process
    - `kill` followed by the pid for a background process

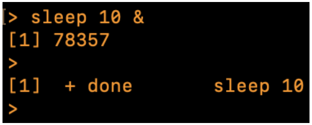
Processes - example

The sleep command suspends operations for a specified time. Here it runs for 10 seconds in the foreground:



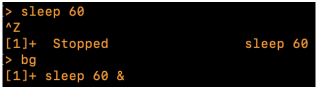
```
> date; sleep 10; date
Tue Oct 17 13:45:25 EDT 2023
Tue Oct 17 13:45:35 EDT 2023
```

Here it runs in the background, getting a job id (1) in addition to a process id (78357). Then a message when it finishes 10 seconds later:



```
> sleep 10 &
[1] 78357
>
[1]  + done       sleep 10
>
```

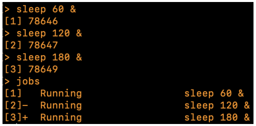
Or we can start in the foreground and then background it:



```
> sleep 60
^Z
[1]+  Stopped                 sleep 60
> bg
[1]+ sleep 60 &
```

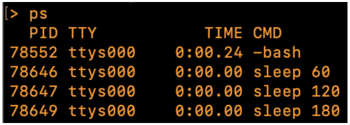
Processes - example (continued)

To see what jobs are running in the background, use `jobs`. You can foreground a job with `fg` followed by its job id. (If you don't specify a job id, `fg` operates on the job with the +; and the job with – would be next after the + job finishes.)



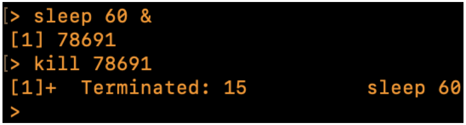
```
> sleep 60 &
[1] 78646
> sleep 120 &
[2] 78647
> sleep 180 &
[3] 78649
> jobs
[1]   Running                 sleep 60 &
[2]-  Running                 sleep 120 &
[3]+  Running                 sleep 180 &
```

To check the status of all processes, use `ps`:



```
[> ps
  PID TTY           TIME CMD
78552 ttys000    0:00.24 -bash
78646 ttys000    0:00.00 sleep 60
78647 ttys000    0:00.00 sleep 120
78649 ttys000    0:00.00 sleep 180
```

To terminate a process, use `kill`:



```
[> sleep 60 &
[1] 78691
[> kill 78691
[1]+  Terminated: 15          sleep 60
>
```

Exit status

- When it ends, a process returns an **exit status** stored in the variable $?
- Exit code 0 means no error
- Any other exit code means error/failure for some reason

Exit status example

```
cat data/chr1.txt
chr1:36926582
chr1:66782904
chr1:77840389

echo $?
0

ehco "Hi"
bash: line 6: ehco: command not found

echo $?
127

grep "369" data/chr1.txt
chr1:36926582

echo $?
0

grep "CHR" data/chr1.txt

echo $?
1

grep "CHR" data/chr1.txt | cat

echo $?
0
```

The computing cluster and htc

- htc (high throughout computing) is the computing cluster we'll use for this class
- the CRC (Center for Research Computing) administers it
- don't work on the login node (always ‘srun --pty bash‘ when you first log in via the terminal)
- you can also **submit** larger jobs using the workload manager, Slurm (we'll cover that later)

Logging on to htc

First, log into the VPN with GlobalProtect.

There are two ways to log on to the cluster

- Via the web: ondemand.htc.crc.pitt.edu
- Via a terminal window
    - `ssh <your_user_name>@htc.crc.pitt.edu`

For details, see https://crc.pitt.edu/getting-started/accessing-cluster.

Next, always start an interactive job: `srun --pty bash` .

Don't work on the login node!

- When you first `ssh` onto the cluster, you're on the **login node**, which is only for logging into and not for working
- Always do `srun --pty` bash to start an "interactive job" when you log in via the terminal (default: 1 hour)
- If you work on the login node
  - you will slow down the cluster and inconvenience the many people using it
  - you will get yourself and me in trouble

Moving files between your computer and htc

Two main options

- the in-browser interface at ondemand.htc.crc.pitt.edu in the Files menu (recommended)
- a free FTP program like Cyberduck (instructions: https://crc.pitt.edu/managingdata)

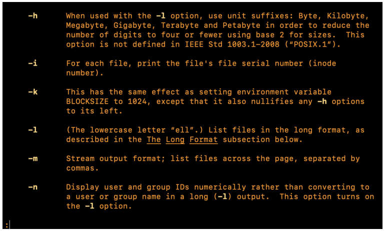
You can also use the more cumbersome `scp` command (again, see https://crc.pitt.edu/managingdata).

Editing text files with `nano`

- `nano` is a simple text command-line text editor we'll use
- others include `vim` and `pico`
- In an interactive job on htc, first load `nano` with `module load nano`
- Entering `nano` will open the editor
  - Just `nano` opens a new blank file
  - `nano fileame` opens a new/existing file (called `filename`)
- Along the bottom are commands, where `^X` means `CTRL + x` and so on
- To exit: `^X`, then type `y` to save, and hit `RETURN`

Use `man` to learn how commands work

Suppose you want to quickly see how a command (e.g., `ls`) works. You can do a web search, use an LLM like ChatGPT, or consult the man page. I entered `man ls` and paged down with `spacebar`:



```
     -h      When used with the -l option, use unit suffixes: Byte, Kilobyte,
             Megabyte, Gigabyte, Terabyte and Petabyte in order to reduce the
             number of digits to four or fewer using base 2 for sizes.  This
             option is not defined in IEEE Std 1003.1-2008 (“POSIX.1”).

     -i      For each file, print the file's file serial number (inode
             number).

     -k      This has the same effect as setting environment variable
             BLOCKSIZE to 1024, except that it also nullifies any -h options
             to its left.

     -l      (The lowercase letter “ell”.) List files in the long format, as
             described in the The Long Format subsection below.

     -m      Stream output format; list files across the page, separated by
             commas.

     -n      Display user and group IDs numerically rather than converting to
             a user or group name in a long (-l) output.  This option turns on
             the -l option.

:
```

Figure 4: Part of the man page for ls

```
total 1736
drwxrwxrwx   4 jonathanchernus  staff   128B Oct 17  2023 data
drwxr-xr-x   3 jonathanchernus  staff    96B Oct 17  2023 data2
drwxr-xr-x   3 jonathanchernus  staff    96B Oct 17  2023 data3
drwxr-xr-x   4 jonathanchernus  staff   128B Oct 17  2023 data4
drwxr-xr-x  15 jonathanchernus  staff   480B Oct 17  2023 figures
-rw-r--r--@  1 jonathanchernus  staff   357B Dec  9  2022 header_pagenrs.tex
drwxr-xr-x@  5 jonathanchernus  staff   160B Sep  5  2023 images
-rw-r--r--@  1 jonathanchernus  staff    25K Sep 23 11:09 unix_basics.Rmd
-rw-r--r--@  1 jonathanchernus  staff   803K Sep 23 11:09 unix_basics.pdf
```

Using git at the command line - make a personal access token first

In order to clone your repository (and push changes) to your home directory on `htc`, you will need to create a personal access token.

- log in to your GitHub account
- click on your profile picture (top right) and select Settings
- click Developer Settings at the bottom left and then Personal access tokens - Tokens (classic)
- click Generate new token - Generate new token (classic)
- enter something like `hugen2071_htc_token` for the Note
- set the expiration date as 90 days
- under Select scopes click the repo box to automatically check the 5 boxes below it
- now click the Generate token button
- **copy the token and paste/store it somewhere safe**
- **you won't be able to look it up again without resetting it**

Using git at the command line - cloning a repository

You only need to this step once:

- copy the ssh `link` for the repository you want to clone
- `cd` to the directory you want
- enter `git clone link` (paste your link in place of the word `link`)
- provide your username and personal access token at the prompt

Using git at the command line - adding, committing, pushing

- First, if you create or change any files, enter `git add -A`
  - make sure all of the changes are staged for the next commit (otherwise git won't keep track of the changes)
  - or do `git add file1 file2` if there are files you don't want under git control
  - **do this every time before you commit/push**
- Second, commit
  - enter `git commit -m "message"` (where "message" is a short, useful description of what changes you've made
  - **do this every time before you push**
- Finally, push
  - enter `git push`
  - this ensures your commit is saved remotely on GitHub