Northwestern
INFORMATION TECHNOLOGY

Intro to Command Line

Northwestern IT Research Computing and Data Services

Matthew Gorby, Haley Carter, Julia Giannini

June 24/25, 2025

Overview

1. Introductions
2. Motivation for using the command line
3. Access to the exercises
4. Navigating the filesystem
5. Shell
6. Utility commands
7. Editing Files
8. Permissions
9. Best practices and tips

Overview

1. Introductions
2. <u>Motivation for using the command line</u>
3. Access to the exercises
4. Navigating the filesystem
5. Shell
6. Utility commands
7. Editing Files
8. Permissions
9. Best practices and tips

Command Line

The command line is where you can give your computer instructions. We use words (a programming language) instead of point-and-click.

The command line language is the language used to talk to your **operating system** - move files, install programs, etc. You can also run other programming languages and software programs from the command line, if you pass the right code to start those programs.

Benefits and drawbacks to using a command line

BENEFITS

- Explicit
- No application updates
- Necessary in some situations
- Automation
- Speed

DRAWBACKS

- Explicit
- Less user-friendly
- Limited display capabilities and other functionalities
- Most commands are permanent (no CTRL+Z or move something to trash)

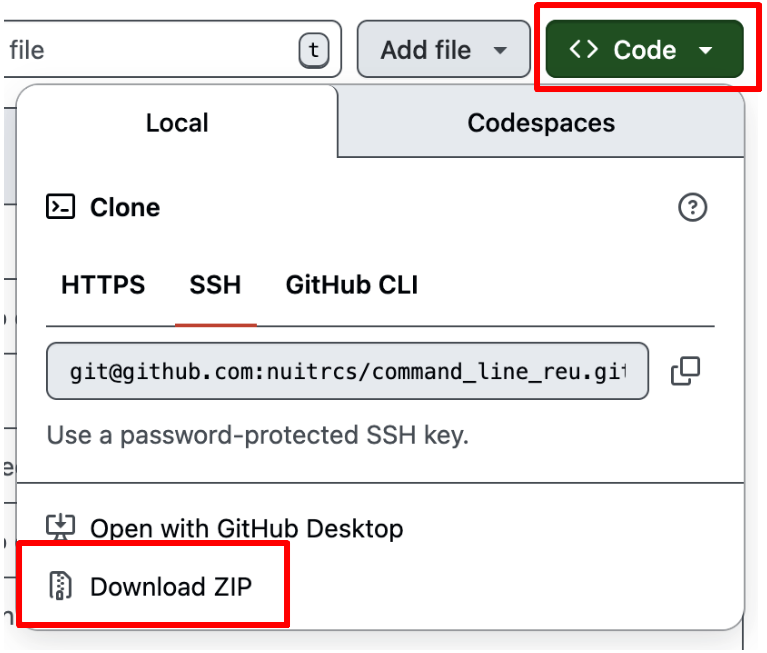
Overview

1. Introductions
2. Motivation for using the command line
3. Access to the exercises
4. Navigating the filesystem
5. Shell
6. Utility commands
7. Editing Files
8. Permissions
9. Best practices and tips

Brief pause to check for terminal applications

Materials for this workshop

- Navigate to https://github.com/nuitrcs/command_line_reu
- Download (and extract/unzip) the repository
- Or, if you're comfortable with it, clone the repository to your laptop
- Note: directory structure can differ with operating system

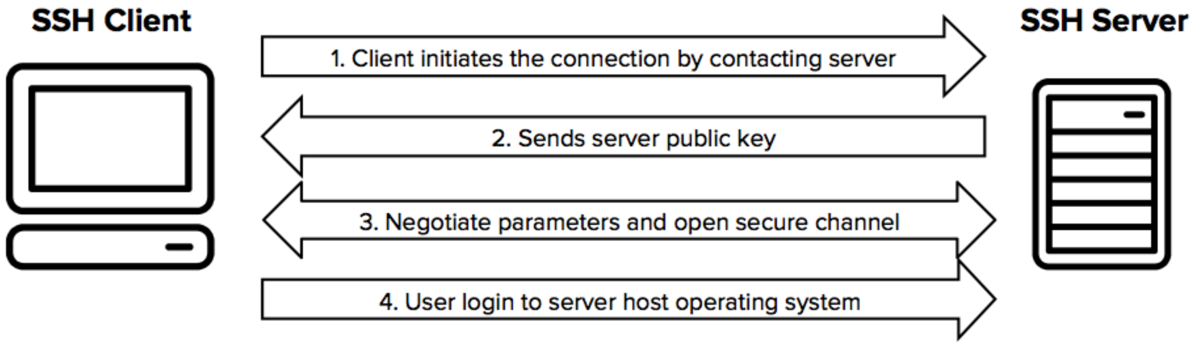


```
git@github.com:nuitrcs/command_line_reu.git
```

Use a password-protected SSH key.

SSH

- `ssh` – secure shell
- Consists of two components
  - Ssh-server
  - Client


SSH Client
SSH Server
1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system

Connect to NU's computing cluster - Quest

- Connect to Quest via `ssh` (**S**ecure **Sh**ell protocol)
  - Method for executing commands on a remote machine
  - Authenticates and encrypts connections between devices
    - Use your Northwestern credentials (NetID and password) to login
- In your terminal, type:

```
ssh -X <your_net_id>@login.quest.northwestern.edu
```

'flag' = extra option passed to `ssh` command
`-X` allows graphics forwarding

Overview

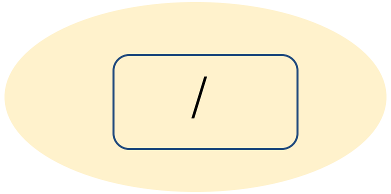
Filesystems

- Linux (and other OS's) organizes its file system in a *Hierarchical directory structure* (tree-like structure)

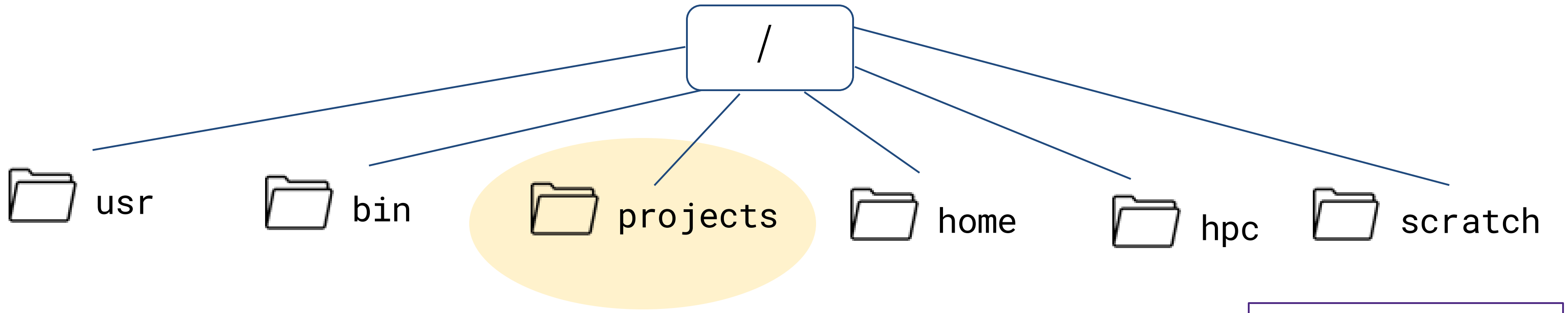
- Some directories are included in the file system (for/by the OS) by default:
  - `/bin` → binary or executable programs
  - `/etc` → system configuration files
  - `/home` → home directory
  - `/usr` → user related files and programs
  - others

- Be aware of *absolute/full pathnames* and *relative pathnames.*

- Differences between OS's
  - Root directory for Linux and Mac: `/`
  - Root directory for Windows (drives): `C:\`

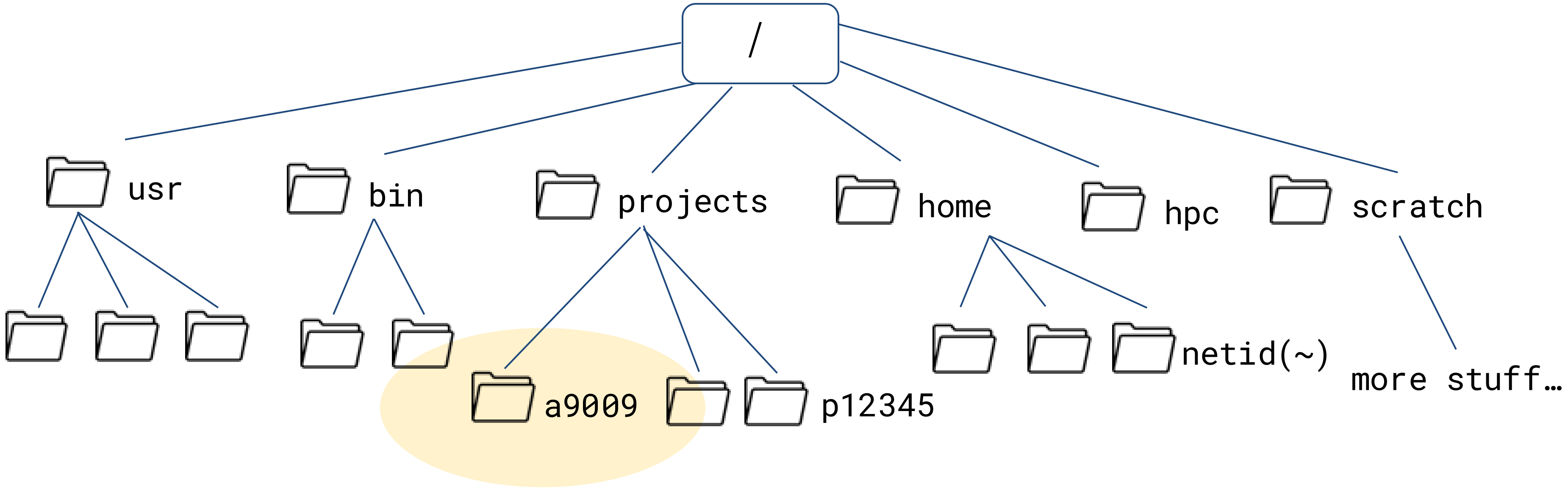


root directory

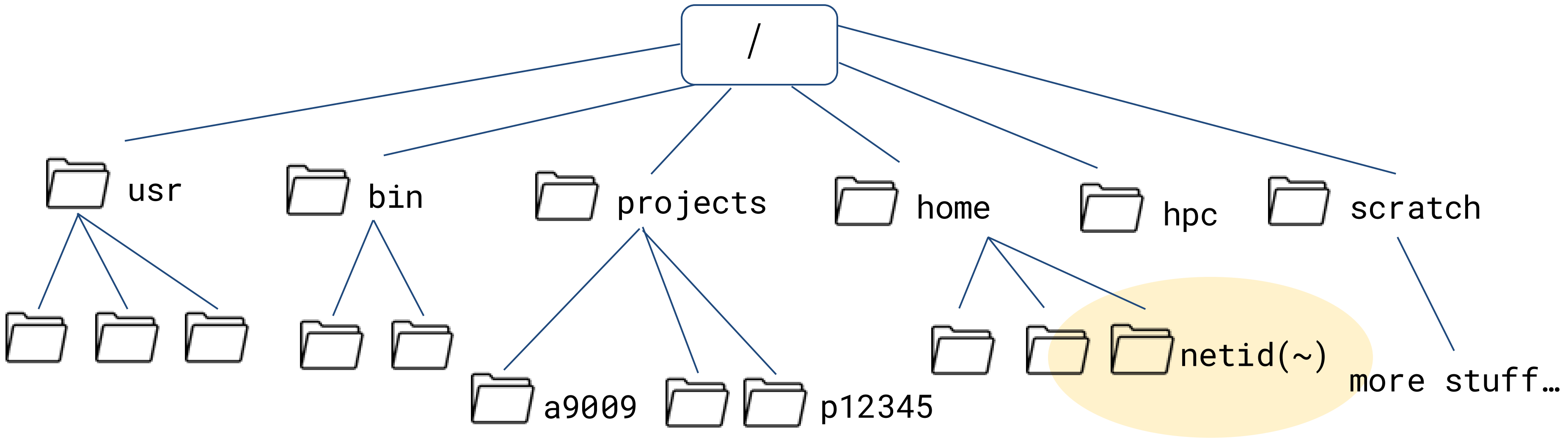


subdirectories

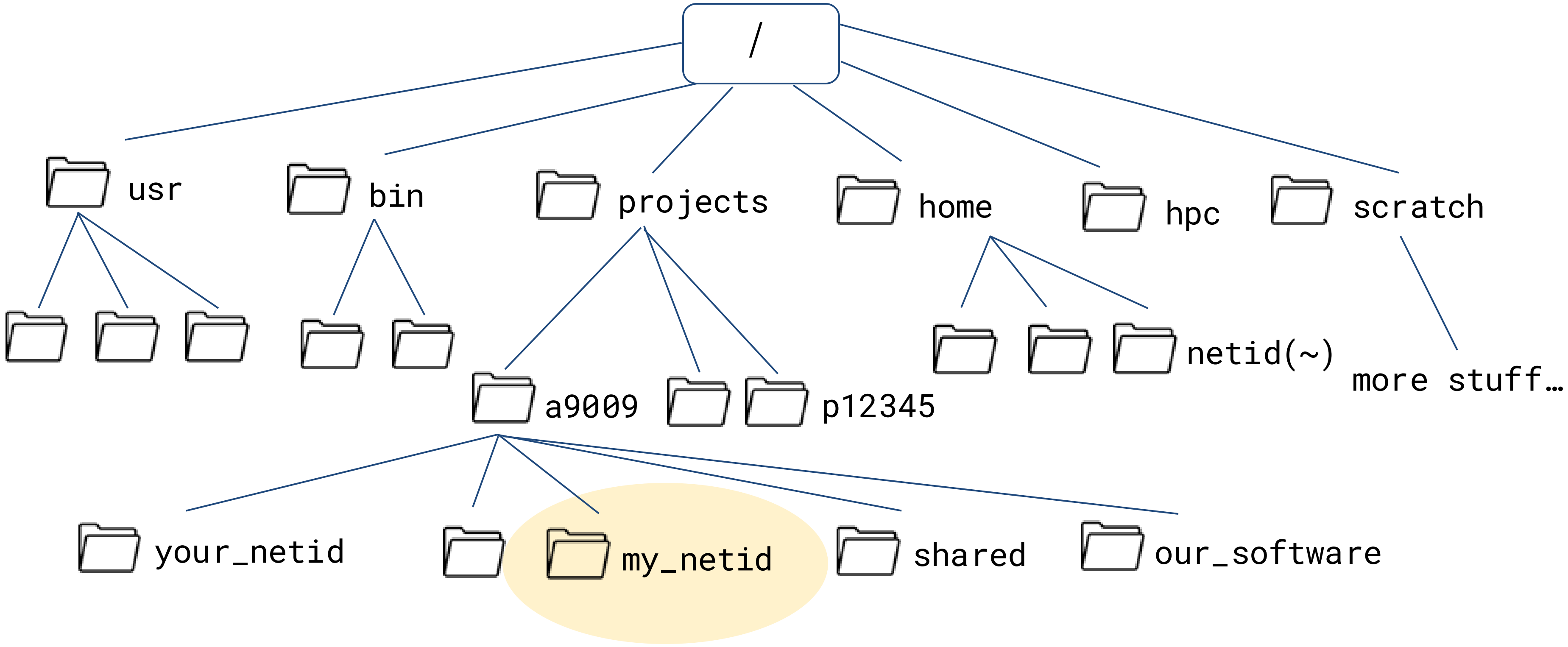
/projects



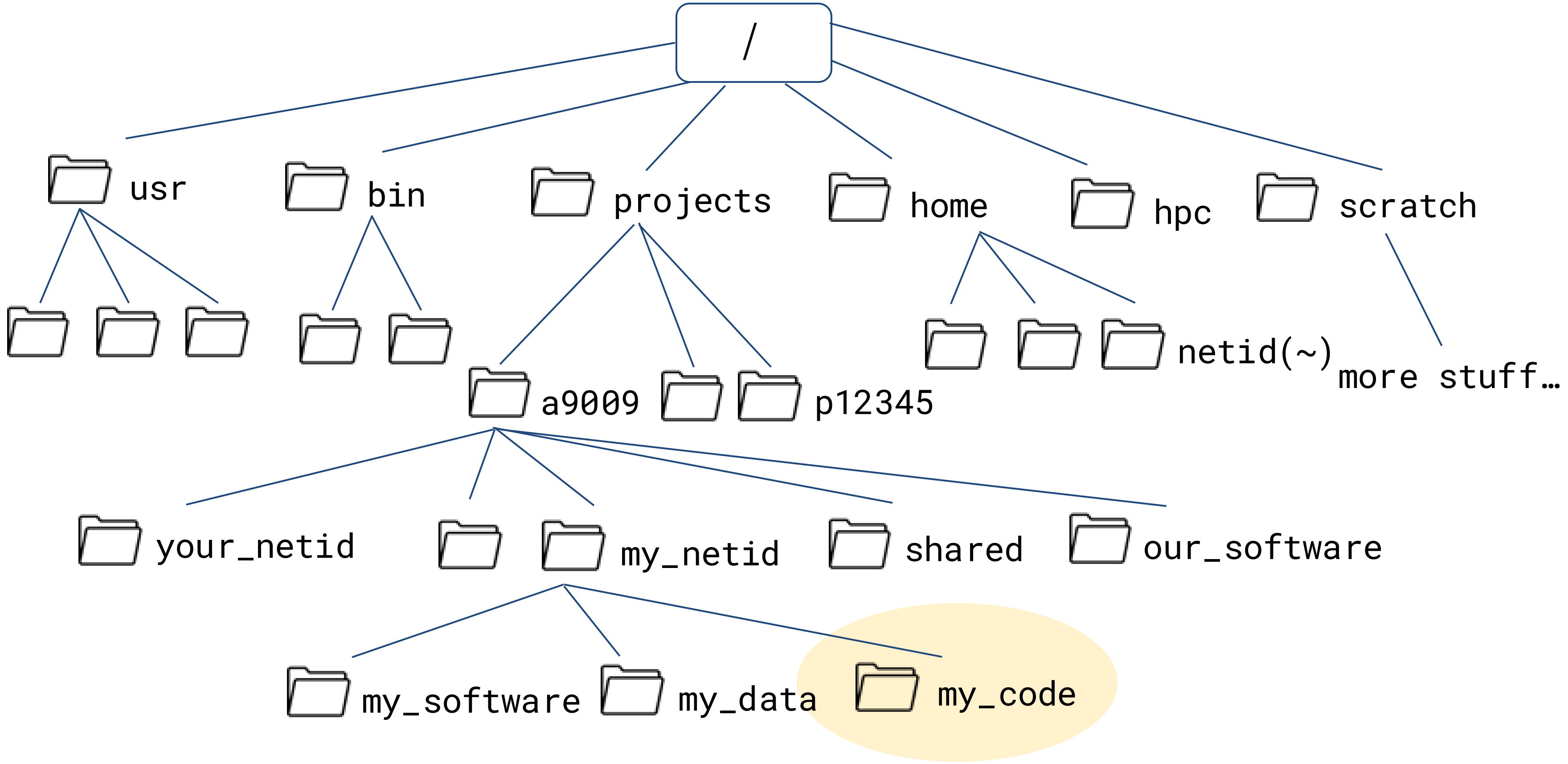
/projects/a9009



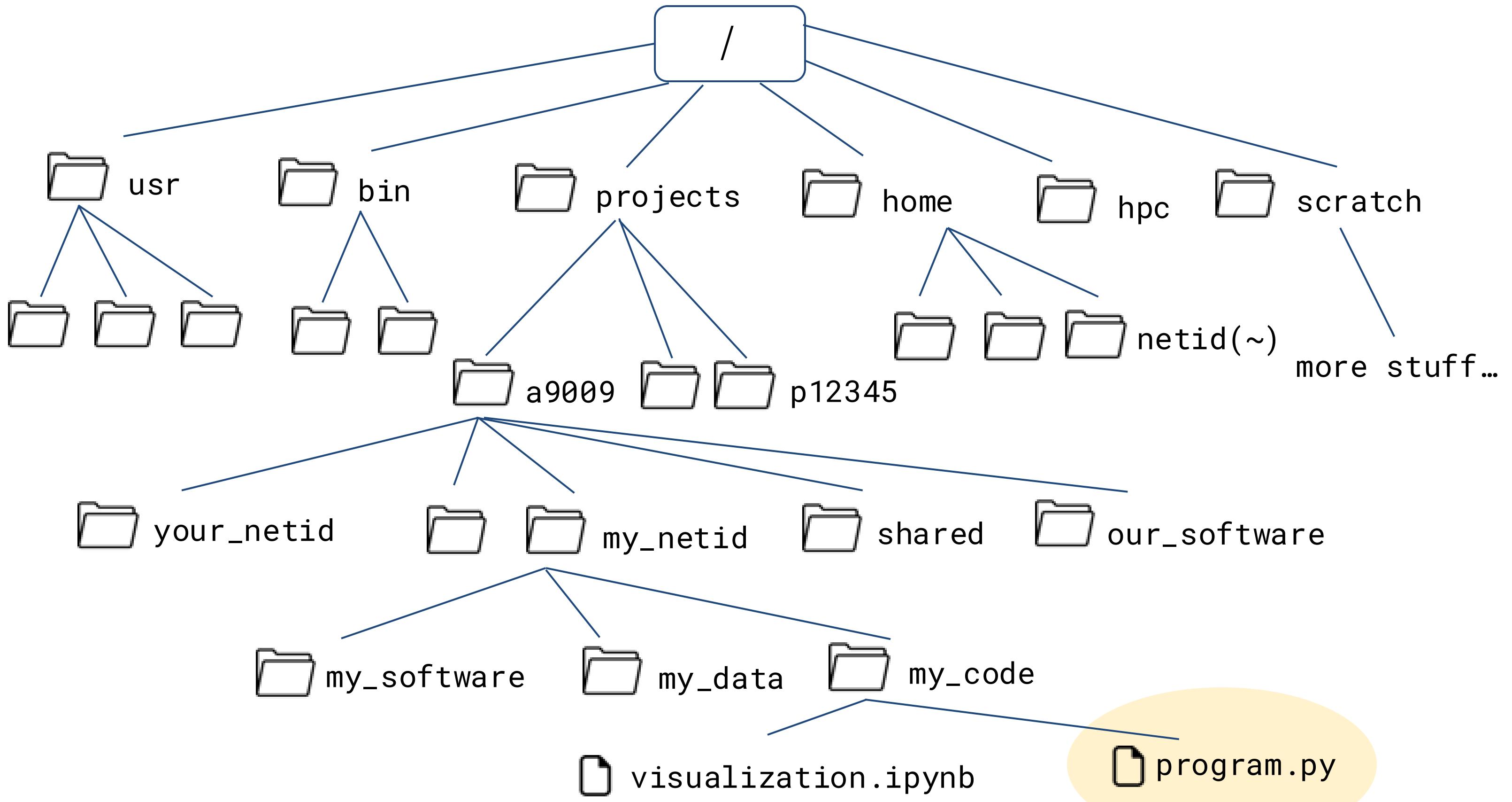
Your home directory == `/home/netid` == `~`



/projects/a9009/my_netid



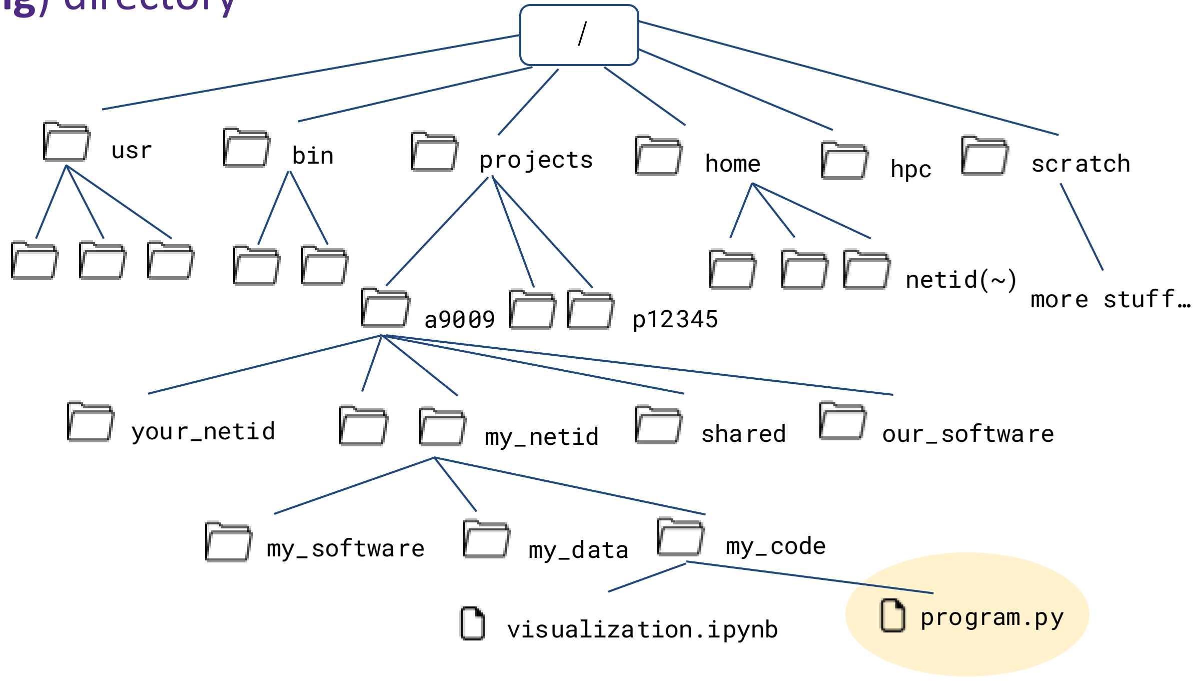
/projects/a9009/my_netid/my_code



/projects/a9009/my_netid/my_code/program.py full path: `/projects/a9009/my_netid/my_code/program.py`

relative path from `/projects/a9009/my_netid/my_code` == `./program.py`

`.` == **current** (or **working**) directory

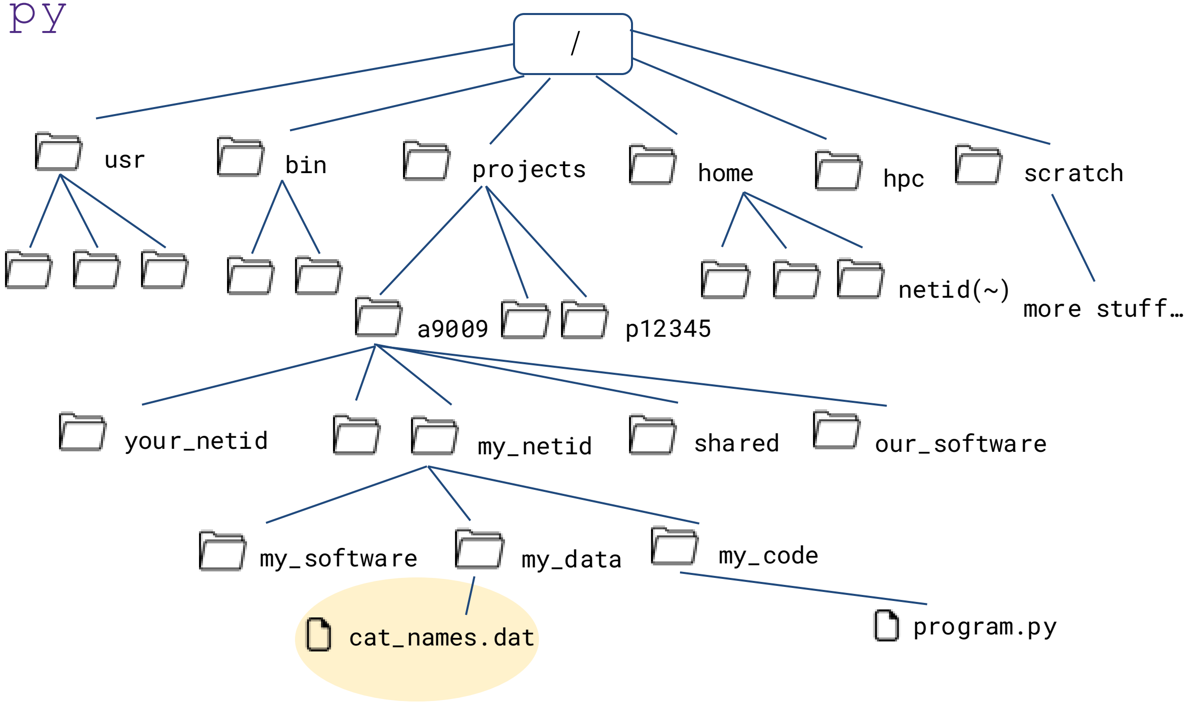


working in: `/projects/a9009/my_netid/my_code`

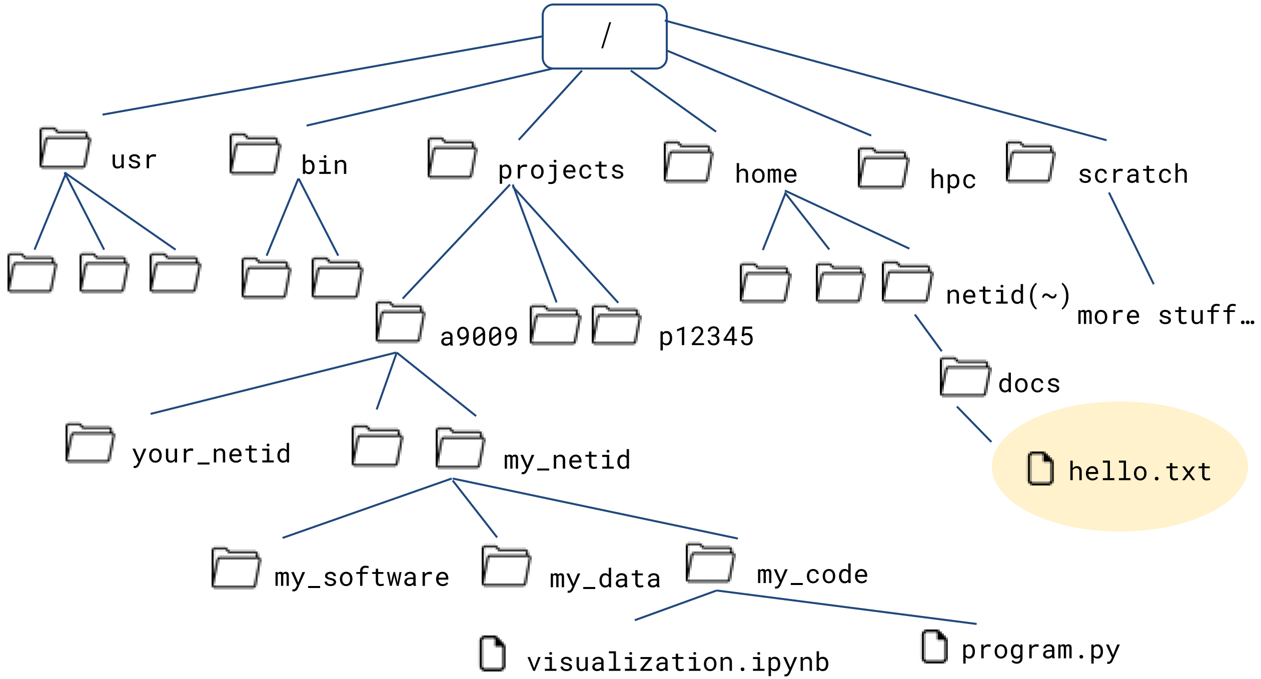
`/projects/a9009/my_netid/ == ..`

relative path to `/projects/a9009/my_netid/my_data/cat_names.dat ==`
`../my_data/cat_names.py`

`..` == parent directory



two ways to specify full path from home: `/home/netid/docs/hello.txt` == `~/docs/hello.txt`



Commands for navigating the filesystem

- `ls` – list the contents of the specified (or current) directory
- `cd <directory>` – change directory to the specified location
- `pwd` – print working directory (where am I?)
- `mkdir <directory>` – create the specified directory

Practice with navigating the filesystem

Find `exercise_0.md` in `command_line_reu`

To display it in your command line, run `cat exercise_0.md`

You can also view it in your web browser on Github

Overview

What is a ‘shell’?

- A shell is a command line interpreter that defines the syntax and environment in which users interface with their computer (run commands, start other programs, etc.)
  - https://linuxcommand.org/index.php
  - User and system defined profiles and *environmental variables*

- There are a variety of shell programs/shell scripting languages
  - Most Linux systems use `bash` (stands for **B**ourne **A**gain **Sh**ell)
  - Others: `ksh`, `tcsh`, `zsh`

- Writing shell scripts
  - A series of commands that automates processes in the command line

Overview

1. Introductions
2. Motivation for using the command line
3. Access to the exercises
4. Navigating the filesystem
5. Shell
6. <u>Utility commands</u>
7. Editing Files
8. Permissions
9. Best practices and tips

Utility commands

- `Ctrl + C` – stop current command execution
- `find <start_dir> -maxdepth <num> -name <filename>` – recursively search the filesystem at specified level of recursion for a file
- `grep <pattern> <filename>` - searches the specified file for a pattern
- **`man <command>` – display the manual for the specified command**
- `top` – display all running processes
- `history` – print your command history
- `Ctrl + R <phrase>` - reverse-i-search your command history for a phrase

Commands for managing files

- `cp <source> <destination>` - copy a file from the source to the destination
- `mv <source> <destination>` - move a file from the source to the destination (also rename a file)
- `rm <file>` - PERMANENTLY delete a file (be very careful with this) ☐
- `touch <file>` – create a file or change timestamp of existing file

Commands for reading and writing files

- `head <file>` – output the first 10 lines of specified file
- `tail <file>` – output the last 10 lines of specified file
- `cat <file>` – output the entire specified file
- `less <file>` – scroll through the contents of specified file
- Command line text editors: `vim <file>`, `nano <file>`, etc..

General tips for utility commands

- Every character is important
  - Watch out for case-sensitivity and typos
- Use `Tab` to auto-complete paths
- Use the ↑ key to scroll through command history
- Reference the documentation for commands/programs you are using
- Bonus:
  - Use the `;` character to execute several commands in series
  - Use the `|` character to pipe the output of one command into another
  - Use the `>` character to pipe the output (`stdout`) of a command into a file

Practice with utility commands

Find `exercise_1.md` in `command_line_reu`
Hint: it is in a subdirectory of `hobbit_house`, which is a subdirectory of `wooded_lane`

Overview

1. Introductions
2. Motivation for using the command line
3. Access to the exercises
4. Navigating the filesystem
5. Shell
6. Utility commands
7. <u>Editing Files</u>
8. Permissions
9. Best practices and tips

Command line text editors

- Run within a terminal or command-line interface (CLI)
- Create and edit plain text files without a graphical interface or needing to repetitively upload/download files
  - Prototype and debug code
  - View progam output
  - Read documentation
- Several different command line text editors available:
  - Nano
    - More user friendly
  - Vim
    - More powerful
  - Emacs
    - More capabilities

```
hello
goodbye

       _.---._     /\\
    ./'       "--`\//
  ./              o \
 /./\  )______   \__ \
./  / /\ \   | \ \  \ \
   / /  \ \  | |\ \  \7
    "     "    "  "
```

Tips for command line text editors

- There is some learning curve, but once you internalize the key bindings, they become second-nature
- Use your arrow keys to navigate (not your mouse)
- Save often!
- Consult the user guide/manual
  - https://www.nano-editor.org/dist/latest/cheatsheet.html
- If you are trying something new, back up important files before drastically editing them

Practice with nano

Find `exercise_2.md` in `command_line_reu`

Hint: It is in a hidden directory. You can reveal it by passing extra options to `ls`, or by using the `find` command

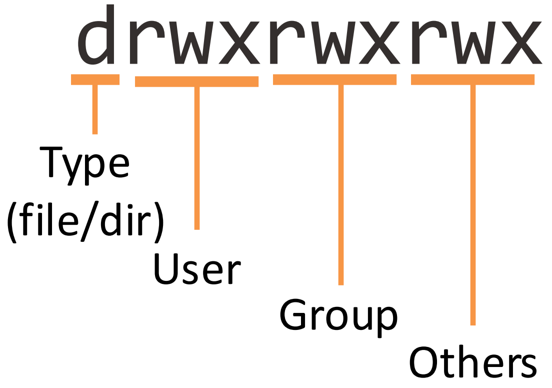
```
find . -maxdepth 6 -name exercise_2.md
```

Overview

1. Introductions
2. Motivation for using the command line
3. Access to the exercises
4. Navigating the filesystem
5. Shell
6. Utility commands
7. Editing Files
8. Permissions
9. Best practices and tips

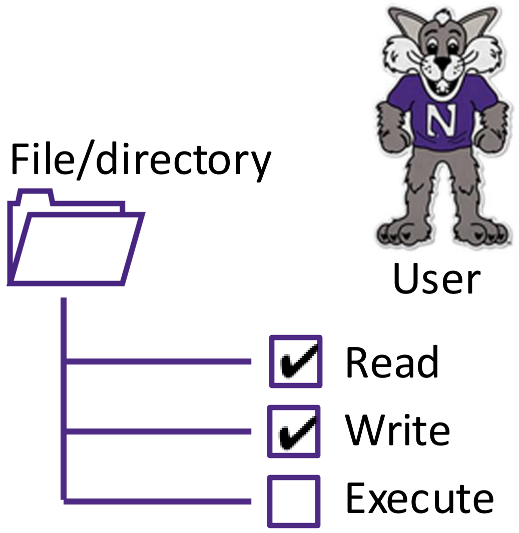
Permission strings

```
[abc1234@quser21 p12345]$ ls -l
total 2
drwxrwxr-x 1 abc1234 p12345 4096 Apr 15 11:00 codes
drwxrwxr-x 1 abc1234 p12345 4096 Apr 15 10:00 data
-rw-rw-r-- 1 abc1234 p12345  259 Apr 15 09:00 test.txt
```



A bit more on permissions:
https://kb.northwestern.edu/70712

What are “permissions”?



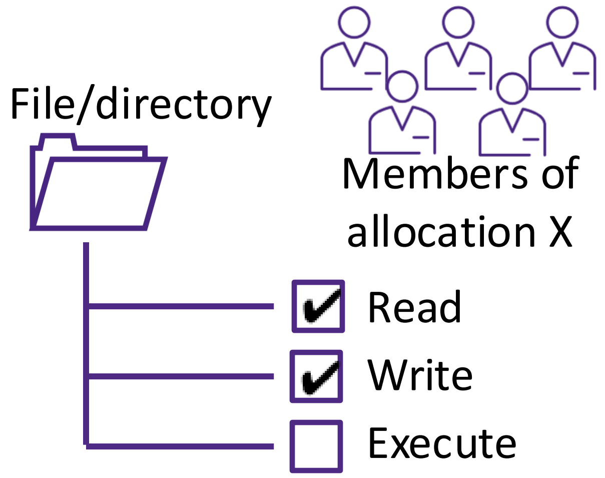
Files you created:

- You will always have full read/write permissions.

Files you did not create:

- You may not have read/write permissions.

Group-level permissions



Group permissions

- Permission are also defined at the “user group” level.
- To see which user groups you are a part of, run “groups”

```
[netid@quser21 ~]$ groups
netid     p30XXX     b10XX
```

Who will have access to my files?

Files created in /projects

- Read/write permissions to all allocation members by default

Files created in /home

- Read/Write permissions only you by default

Overview

1. Introductions
2. Motivation for using the command line
3. Access to the exercises
4. Navigating the filesystem
5. Shell
6. Utility commands
7. Editing Files
8. Permissions
9. Best practices and tips

Best practices and tips for the command line

- Every character is important
  - Watch out for case-sensitivity and typos
- Permanency of commands (deleted files are often gone forever)
- Use `Tab` to auto-complete paths
- Use the ↑ key to scroll through command history
- Use `CTRL+R` to search through command history
- Use `CTRL+C` to stop/cancel the execution of a command
- Avoid spaces and special characters in file and folder names when possible
  - Or type `\` before each space to specify the path
- Consult the man pages of commands you’re using
- Be cautious, but don’t be afraid to try things out!

Thank You!

Questions?

quest-help@northwestern.edu

Request a Consultation