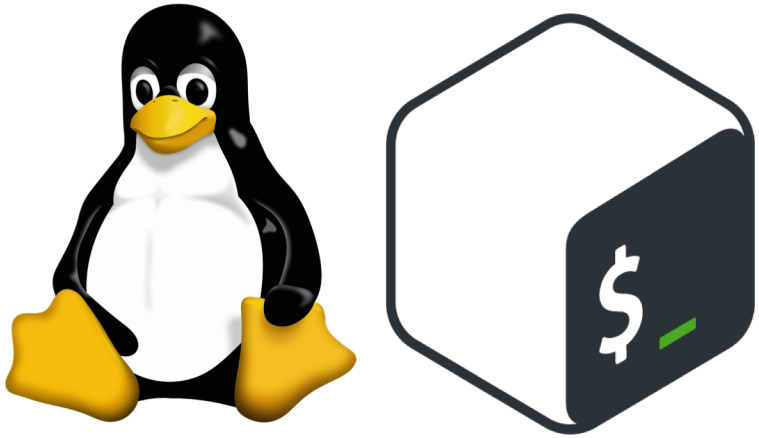


An introduction to Linux and the Unix Shell

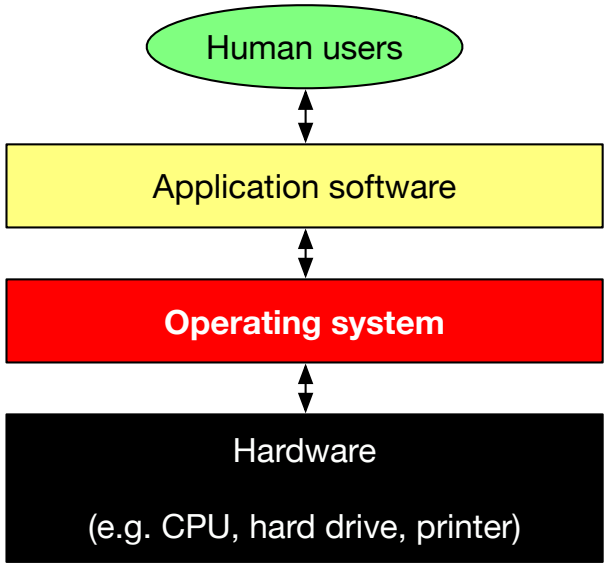
Rui Meireles

Assistant Prof. of Computer Science, Vassar College, USA

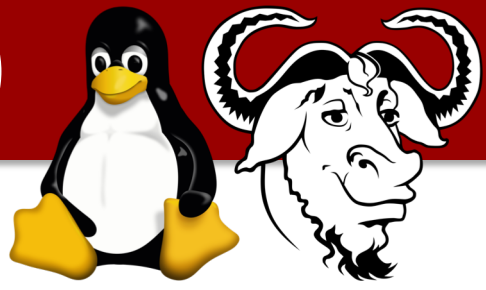
Introduction

What is Linux? (1/2)

- First of all, Linux is an Operating System (OS)
  - Software that manages the computer's hardware and provides common services for software
    - E.g. read keyboard input, draw window on screen
  - Include useful software utilities
    - E.g. compiler, text editor



What is Linux? (2/2)

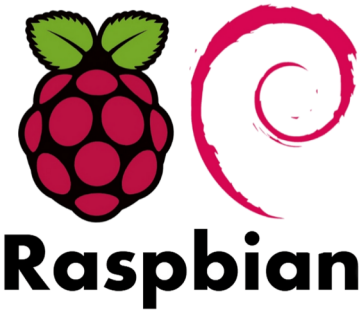


- The Linux OS is a combination of:
  - **Linux kernel**
    - Provides core OS functionality (e.g. process and hardware management)
    - Multitasking and multiuser, advanced security
    - Open source, created by Linus Torvalds at the University of Helsinki in 1991
  - **GNU's Not Unix (GNU) system software**
    - Open source software project started in 1983 by Richard Stallman at MIT
    - Intended to become its own OS but the kernel (HURD) isn't ready yet
    - Includes glibc C library, libstdc++ C++ library, gcc C/C++ compiler, gdb debugger, coreutils, binutils, bash shell, GNOME desktop env., Emacs text editor, etc
- Both the Linux kernel and the GNU utils are Unix-inspired
  - Highly influential OS developed at Bell Labs in the 1970s
    - Direct descendants include BSD and macOS/iOS
  - Can think of Linux as an open source version of Unix

Linux distributions (1/2)

- A distribution is a packaging of the Linux operating system
  - All run a version of the Linux Kernel, differ in included software
  - Permissive licensing allows for customization, leading to a lot of choice
  - Different distributions may target different uses (e.g. server vs desktop vs embedded device), or user types (e.g. beginner vs power user)
- Examples:

for desktops



for RaspberryPi single-board computer

for network routers

- A note regarding Android OS
  - Although it uses the Linux kernel is not considered a Linux distro because it lacks the GNU utilities, includes Google-developed utils instead

Linux distributions (2/2)

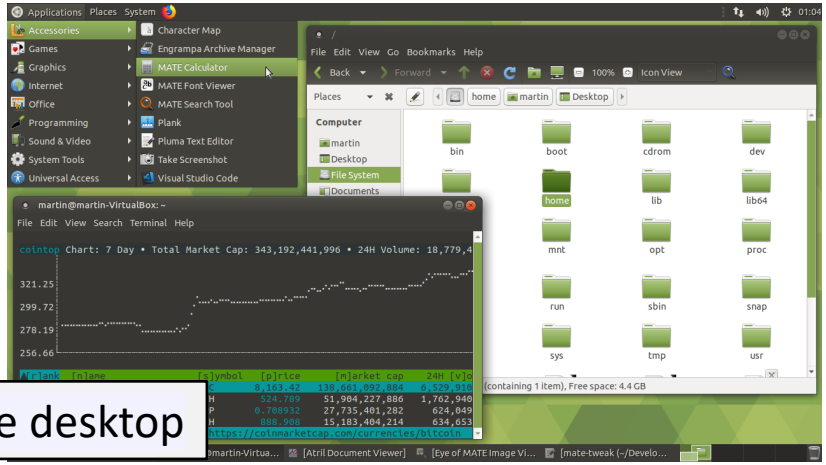
- Some general-use distributions of note:
  - **Debian**: focus on stability over novelty
    - Does not include non-free software by default
    - Includes synaptic package manager: easy to install new software
    - Many popular distros are Debian forks: e.g. Mint, Ubuntu
  - **Arch**: for power users
    - Rolling release, configuration more exposed, x86_64 only
    - Is the basis for Manjaro (currently #1 on distrowatch.com)
  - **Ubuntu MATE**
    - Combination of Ubuntu with MATE desktop environment
    - Debian-based but more up-to-date
    - Currently deployed in the CS department machines

debian

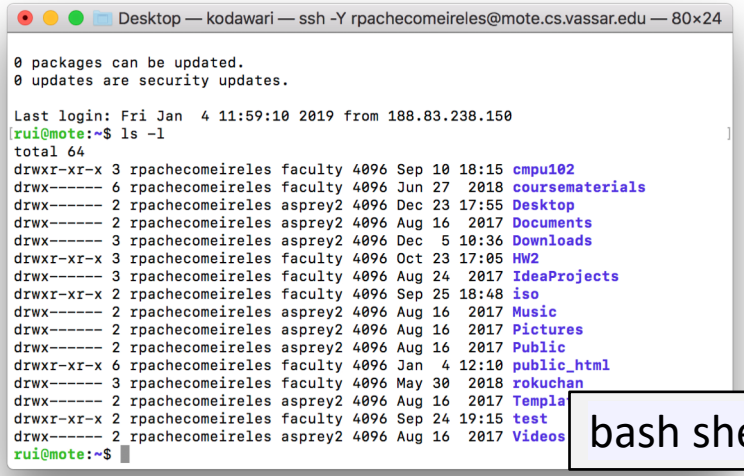
ubuntu MATE

Interacting with the operating system

- Graphically
  - Graphical primitives such as windows, icons and buttons
  - Common window managers: KDE, GNOME, **MATE** (GNOME fork), XFCE
- **Textually (our focus today)**
  - Through a command-line interpreter or shell
  - Very powerful, can actually be seen as a programming language
  - Can run inside a graphical window (terminal emulator)
  - Common Unix shells conforming to the POSIX standard: **bash**, dash, csh



Mate desktop

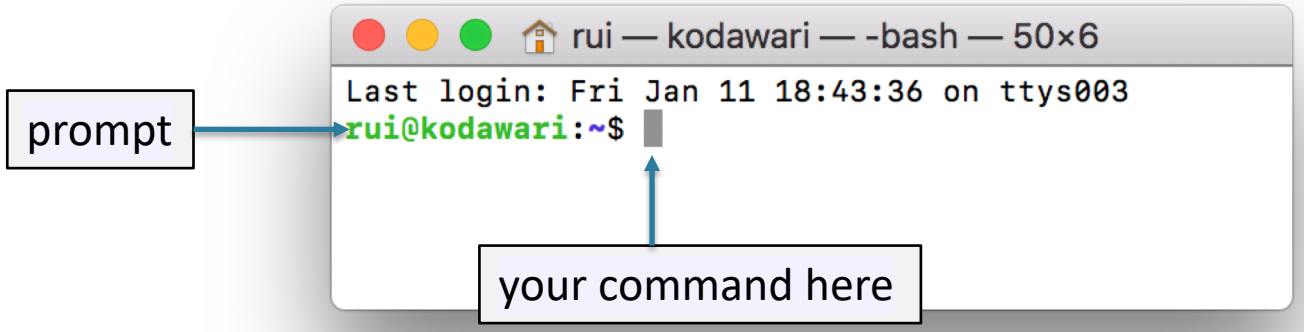


bash shell

The Unix shell

Our first shell command (1/2)

- Open up a bash terminal emulator by accessing:
  - Menu → System Tools → Mate Terminal



- The prompt is customizable
  - Typically shows username, computer name; ends in dollar sign $
- The shell is waiting for a command
  - Type `echo Hello World` and hit the Enter key
  - What happened?

Our first shell command (2/2)

- What happened?
  1. The command was executed, yielding the writing of `Hello World` onto the terminal window
  2. A prompt is displayed, allowing us to enter a new command

```
rui@mote:~$ echo Hello World
Hello World
rui@mote:~$
```

- The echo program writes its arguments to the standard output (which is, by default, the terminal) and a new line
- General shell command format:

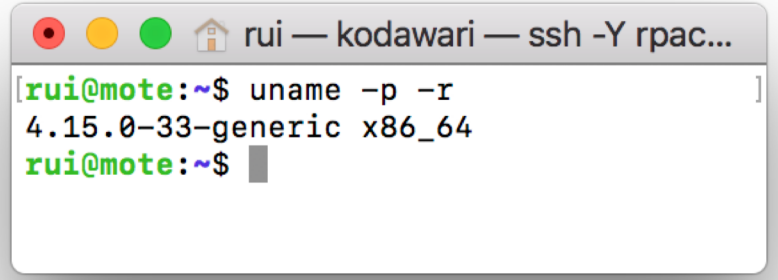
  `<program-name> [arg1] [arg2] … [argn]`

- Different programs support different arguments
  - E.g. the `echo` program is variadic (takes any number of arguments): it just writes them all in order to the standard output

Learning about commands: `man`

- Every GNU program has an associated manual page
- The command `man <program-name>` lets us access it
- Navigating a manual page
  - We can use the navigation keys (e.g. arrow keys) to move around
  - We can search by hitting `/`, typing in your query and hitting entry
    - To navigate multiple hits we can use `n` for next and Shift+`n` for previous
  - Finally, we can exit the manual by hitting `q`
- Exercise
  - The `uname` program can be used to obtain system information
  - Use `man` to figure out how to use `uname` to obtain:
    1. The processor type of the computer you are working on
    2. The kernel release of your Linux OS
  - Use `uname` for the aforementioned purposes

man exercise solution



- Dash and letter combinations are commonly used as arguments to specify program options
- Often, multiple letters can follow a single dash for brevity:
  - E.g. the command `uname -pr` is equivalent to `uname -p -r`
- Long-form options are preceded by two dashes, can't be combined
  - E.g. the `uname --processor --kernel-version`

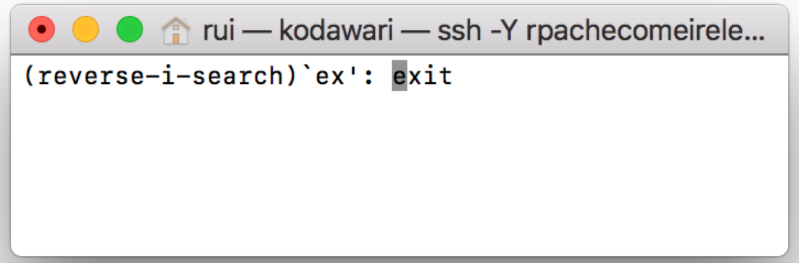
Searching for programs

- We can search `man` pages to try and find programs to do things
- `apropos <query>` searches the entirety of man pages
  - Can limit search to shell programs by providing option `-s 1` (section 1)
- Exercise
  - Try to find a program to display the current date using `apropos -s 1`
- Alternative:
  - Type a prefix and then hit tab to list all programs starting by that prefix
  - E.g. typing `d` and hitting tab will list all programs starting with `d`
- Also, sometimes an online search yields the best results
- If we know a program's name it's easy to learn what it does
  - `whatis -s 1 <progam>` gives us a one-line description
  - Example:

```
rui@mote:~$ whatis uname
uname (1)            - print system information
```

Navigating command history

- Retyping entire commands can be cumbersome
- We can use the up and down keys or Ctrl-`p` Ctrl-`n` to recall previous commands
- We can also search command history by hitting Ctrl-`r`
  - Cycle through hits by pressing Ctrl-`r` over and over
- Further, we can navigate within a command:
  - Ctrl+Left, Ctrl+Right to skip words (Alt+Left, Alt+Right on macOS)
  - Ctrl-`a` to move to the beginning, Ctrl-`e` to move to the end
  - Press Tab for auto-complete



- Exercise
  - Experiment with your command history, search for the `echo Hello World` command

Files

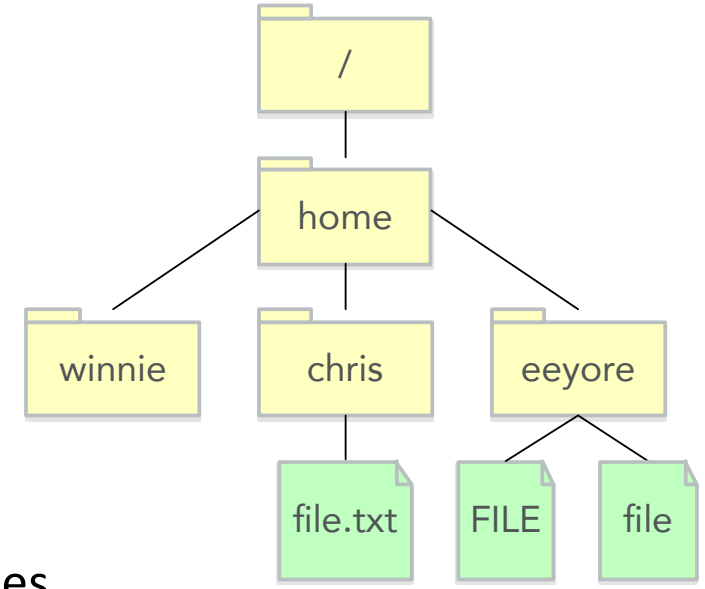
Files

- "In Unix, everything is a file"
- A file is a unified abstraction representing an input/output resource
  - Data container you can read from and/or write to
  - They are named entities

- Resources represented by files can be:
  - A stream of data located in persistent memory (e.g. a document)
  - A stream of data located in volatile memory (used for inter-process communication)
  - A link to another file
  - A folder/directory containing other files
  - An input/output device, e.g. disk, keyboard, network card, printer, etc
- How do we know what kind of file we're looking at?
  - Filesystems store file metadata
  - Standardization

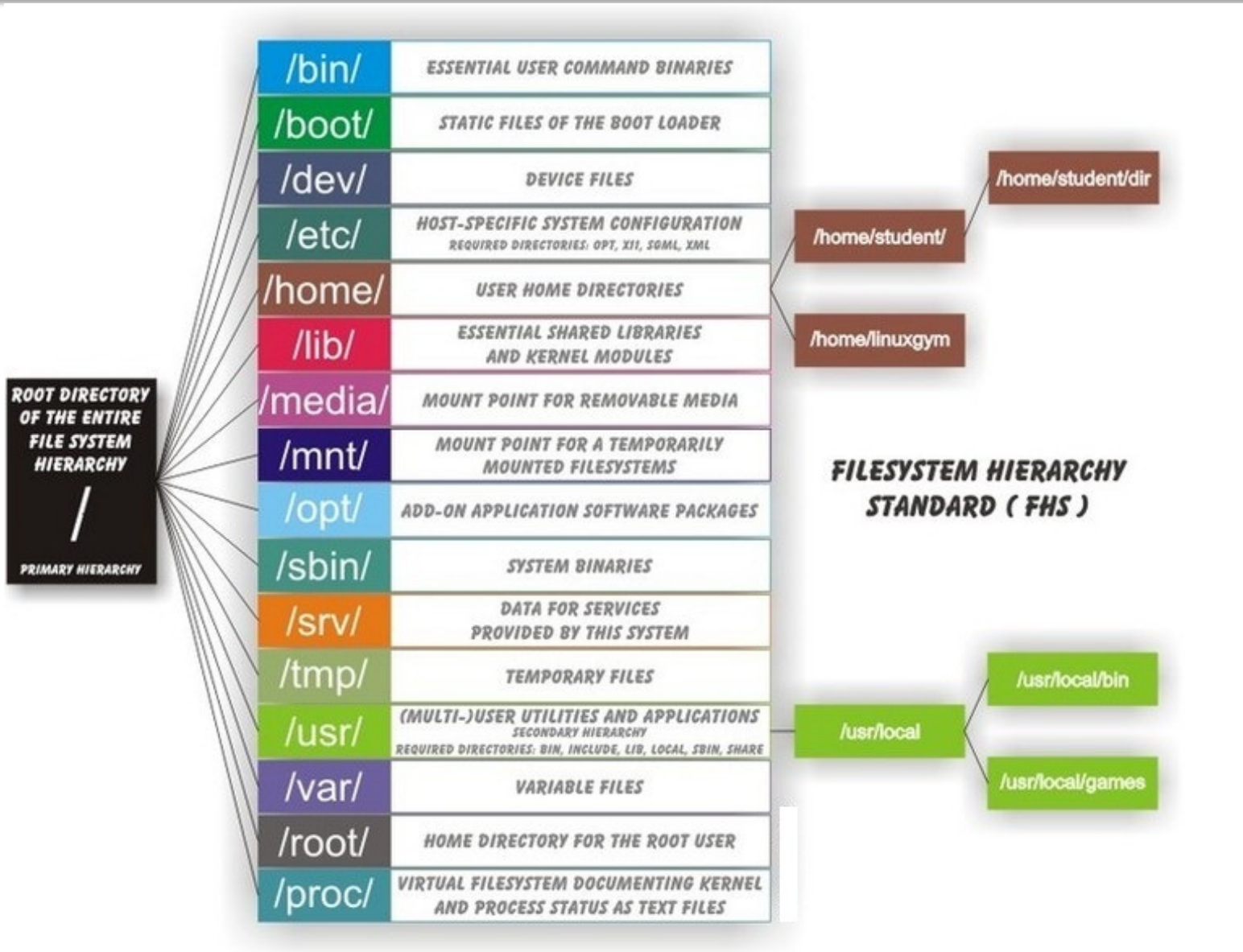
The Unix filesystem

- All files reside in a unified namespace rooted at the / folder
- Folders used to organize files
  - Can contain files or other folders
  - Results in tree-like hierarchy
- Files are identified by a unique path
  - Concatenation of:
    - Folders from the root, separated by /
    - File name
  - Files in same folder must have different names
  - Unix paths are case-sensitive



- File extensions:
  - File name suffixes used to convey type of file contents
  - Start with .
  - E.g. `.txt` is used for plain text files
- Example paths:
  - `/home/chris/file.txt`
  - `/home/eeyore/FILE`
  - `/home/eeyore/file`

Filesystem Hierarchy Standard (FHS)



Listing files using `ls` (1/4)

- Let's use the shell to interact with the file system
- Shell commands are executed in the context of a folder
  - The current working directory
    - It is often part of the prompt, or we can use `pwd` to learn what it is
  - Upon initialization this will typically be your home folder
    - Typically `/home/<username>`
- `ls` command format: `ls [options] [path1] … [pathn]`
- Examples:
  - `ls` lists files in current working directory
  - `ls /home` lists files in `/home` folder
- Exercise: what files are in your home folder?

Listing files using `ls` (2/4)

- Common `ls` options:
  - `-l`: long listing format shows file metadata: permissions, number of hard links, owner, group, size (in bytes), last modification time
  - `-a`: show all files, including hidden ones, the ones that start with `.`
- Exercise:
  1. What file in your home folder was the last to be modified?
  2. Does your home folder contain any hidden files?

```
rui@mote:test$ ls -la
total 32
drwxr-xr-x  2 rpachecomeireles faculty 4096 Jan 11 17:54 .
drwx--x--x 43 rpachecomeireles faculty 4096 Jan 11 13:43 ..
-rw-r--r--  1 rpachecomeireles faculty   72 Sep 24 19:14 package.bluej
-rw-r--r--  1 rpachecomeireles faculty  471 Sep 24 19:14 README.TXT
-rw-r--r--  1 rpachecomeireles faculty    3 Jan  9 15:26 .secret
-rw-r--r--  1 rpachecomeireles faculty  375 Sep 24 19:15 Test.class
-rw-r--r--  1 rpachecomeireles faculty  546 Sep 24 19:15 Test.ctxt
-rw-r--r--  1 rpachecomeireles faculty  711 Sep 24 19:15 Test.java
rui@mote:test$
```

Listing files using `ls` (3/4)

- Relative paths
  - If the path doesn't start with `/`, it will be appended to the current directory
    - E.g. if current dir is `/home` then `ls rui` ↔ `ls /home/rui`
  - `~` can be used as shorthand for the home directory
    - E.g. if home dir is `/home/rui` then `ls ~/dir` ↔ `ls /home/rui/dir`
  - Every folder contains files . and .. that point to the current and parent folders, respectively
    - E.g. `ls .` ↔ `ls`
    - E.g. `ls /home/..` ↔ `ls /`
- Exercise
  - List all the files in your home folder's parent directory, using a command that would work regardless of the current directory, your username, or home folder location

Listing files using `ls` (4/4)

- Wildcards allow us to match multiple names (expanded bef. exec.):
  - `?`: can represent any single character
    - E.g. `ls hd?` would list `hda`, `hdb`, `hdc`, etc..
  - `*`: can represent any number of characters (including zero)
    - E.g. `ls *.txt` would list all files ending in `.txt`
  - `[range]`: matches a character in `range`
    - E.g. `ls d[a,o]d` would list `dad`, `dod`, `ls d[a-z]d` lists `dad`, `dbd`, …, `dzd`
  - `[!range]`: matches a character not in `range`
    - E.g. `ls d[!a-z]d` would list `d8d` but not `dod`
  - `{t1, …, tn}`: matches at least one of the terms in comma-separated list
    - E.g. `ls {*.txt, *.jpg}` matches everything ending in `.txt` or `.jpg`
- Wildcards are usable pretty much everywhere, not just with `ls`
  - Learn more at https://devdocs.io/bash/html_node/shell-expansions
- Exercise: list all the files ending in `conf` from folder `/etc/`

Navigating the file system

- `cd` (change directory) can be used to change the current directory
- Format: `cd [path]`
  - The path can be absolute or relative
  - Not specifying a path (i.e. just `cd`) will change to the home directory
  - `cd -` can be used to return to return to the previous directory
- Typically, the directory change will be reflected on the prompt
  - Not guaranteed though, depends on prompt configuration
- Examples:
  - Absolute path: `cd /var/tmp`
  - Relative path: `cd ~ && cd rui && cd .. && cd -`
    - `&&` is used to have multiple commands in one line, with the following command only executing after successful completion of the previous
      - `;` can be used to run multiple commands in parallel, e.g. `uname; ls`
    - Exercise: where did we end up after this sequence?

Creating files and folders

- `touch <path>` will create an empty file `<path>`
  - Can use absolute or relative path
  - E.g. `touch test` creates an empty file named `test` in current directory
- Editing text files:
  - `nano` is a user-friendly terminal-based text editor
    - Open file with `nano <path>`
  - Later on you can try more sophisticated editors such as `vim` and `emacs`
- Creating folders
  - `mkdir <path>` creates new empty folder `<path>`
    - E.g. `mkdir testdir` creates folder `testdir` inside current working directory

Viewing file contents

- Viewing file contents:
  - `cat <path>` prints entire to the standard output, bad for large files
  - `less <path>` prints a single screen's worth, then pauses
    - Let's you browse the file using navigation keys, `q` to quit
    - Let's you search using `/` like `man`
    - E.g. `less /proc/cpuinfo` and `less /proc/meminfo`
- Exercise:
  1. Create a folder named `scripts` inside your home folder
  2. Create a file named `hello.sh` inside the `scripts` folder
  3. Edit it using a text editor so that it contains the following two lines:

```
#!/bin/bash
echo "Hello world!"
```

  4. Print `hello.sh`'s contents to the standard output

File permissions (1/2)

- Unix has built-in file security. Every file has these permissions:

	Owner	Group	Others
Read (r)	yes or no	yes or no	yes or no
Write (w)	yes or no	yes or no	yes or no
Execute (x)	yes or no	yes or no	yes or no

- Groups useful to manage permissions when there are many users
  - `less /etc/group` will show all groups in the system
  - `groups` will show the groups you are a part of
- We can list the permissions of a given file using `ls -l`:

```
rui@mote:scripts$ ls -l
total 4
-rw-r--r-- 1 rpachecomeireles faculty 33 Jan 11 12:01 hello.sh
```

File permissions (2/2)

- Permissions can be changed by file owner and `root` user
  - We'll talk about `root` in a little bit
- `chmod <opt> <perm> <p1> … <pn>` changes permissions
  - `-R` for recursive, `-v` for verbose and `-f` for force are common `opts`
  - `perm` is the permissions string, is highly flexible
    - E.g. `chmod u=rwx,g=rx,o= hello.sh` gives, on file `hello.sh`, all permissions to owner, read and execute permissions to group and none to others
    - E.g. `chmod a=rwx hello.sh` gives all permission to everyone
    - E.g. `chmod +x hello.sh` gives execute permission to everyone
    - E.g. `chmod -x hello.sh` removes execute permission for everyone
- `chown <opt> <nown> <p1> … <pn>` to change owner
  - `nown` is the new owner
  - E.g. `chown martha hello.sh`
- `chgrp <opt> <ngrp> <p1> … <pn>` to change group
  - E.g. `chgrp students hello.sh`

File execution

- Let's give our hello script execution permissions and then run it:

```
rui@mote:scripts$ chmod u=rwx,g=,o= hello.sh
rui@mote:scripts$ ls -l
total 4
-rwx------ 1 rpachecomeireles faculty 33 Jan 11 12:01 hello.sh
rui@mote:scripts$ ./hello.sh
Hello world!
rui@mote:scripts$
```

- Why do we need the `./` prefix when executing `hello.sh`?
  - The shell only looks for executables in the folders specified in the `PATH` environment variable

```
rui@mote:scripts$ echo $PATH
/usr/csapps/berkeley_upc/bin:/opt/swift/usr/bin:/usr/local/MATLAB/R2018a/bin:/opt/go/bin:/usr/
csapps/games:/usr/csapps/bin:/usr/csapps/sbin:/usr/csapps/acl:/opt/adg:/opt/adg32:/usr/local/s
bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/li
b/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin
```

  - For executables located elsewhere we need a complete (absolute or relative) path

The super user

- In a Unix system, `root` is the superuser
  - Has permission to do anything they please
- During normal use you should use a regular user, for security
- Regular users can be made into **sudoers**
  - A sudoer is a user that can temporarily become `root`
    - Is someone that is part of the `sudo` group
  - `sudo <command>` executes supplied command as `root`
  - E.g. `sudo ls /root/` lists the content's of `root`'s home folder
  - Note: you are not a sudoer on the CS machines, you can be on your own PC

Deleting files

- `rm <path1> … <pathn>` removes regular files
  - Can use relative paths, wildcards, etc
  - Useful options:
    - `-R` recursive deletion, useful to delete folders
    - `-i` ask for confirmation before deleting each file
    - `-f` delete without asking for confirmation
  - Be careful, a simple `rm -Rf /` can wipe out an entire system
  - Examples:
    - `rm a*` removes all files started by `a` in current directory
    - `rm -R /tmp/bye` recursively removes folder `/tmp/bye`
- `rmdir <path>` let's us remove directories, but only empty ones
  - E.g. rmdir ~/test deletes the test subfolder inside the home folder
- Exercise
  1. Create files `a.txt` and `b.txt`
  2. Delete both files using a single command

Moving files

- `mv <path1> … <pathn> <dst>` moves files
  - `<path1> … <pathn>` are the files to be moved
  - `dst` is the destination path, there can be only one
    - Corollary #1: if we're moving multiple files, `dst` has to be a folder
    - Corollary #2: if we're moving a single file we can use `mv` for renaming
      - In fact there is no dedicated rename program
    - If the destination is a folder, it has to preexist
- Examples:
  - `mv *.txt /tmp/` moves all files ended in `.txt` to folder `/tmp`
  - `mv a b` renames file `a` to `b`
- Exercise
  1. Create files `a` and `b`
  2. Create folder `test`
  3. Move `a` and `b` to folder `test` with a single command
  4. Move `a` and `b` back to their original locations with a single command

Copying files

- `cp` works just like `mv`
- General format: `cp <path1> … <pathn> <dst>`
  - `<path1> … <pathn>` are the files to be copied
  - `dst` is the destination path, there can be only one
    - Corollary : if we're copying multiple files, `dst` has to be a folder
    - If the destination is a folder, it has to preexist
- Examples:
  - `cp *.txt /tmp/` copies all files ended in `.txt` to folder `/tmp`
  - `cp a b` creates a copy of file `a` named `b`

Archive files

- An archive is a file whose contents are the concatenation of one or more other files
  - Useful for long term storage (e.g. data backup) and data transmission
- Archives are often compressed to save space/bandwidth/time
  - Compression is performed by identifying repeated patterns and including them only once
- Most common archiving tool: `tar`
  - Stands for (t)ape (ar)chive, a remnant of the days when tapes were the most cost-effective way to store archival data
  - Depending on the specified options, `tar` can be used to both create archives and extract files from archives (also known as deflating and inflating)

Creating archives using `tar`

- Format: `tar -cf <opt> <opath> <inpath1> … [inpathn]`
  - `-c` or `--create` specifies creation, `-f` or `--file` specifies file mode
  - Input paths can include wildcards, can be folders
  - E.g. `tar -c arch.tar *.txt *.sh`
    - Creates archive containing all files ending in `.txt` or `.sh`
    - `.tar` is the standard file extension for tar archives
- Other common options
  - `-v` or `--verbose` lists files as they're processed
  - `-z` or `--gzip` compresses archive using `gzip` (`.tar.gz` file extension)
- Compressed archive example
  - `tar -czvf out.tar.gz ~/.bash_profile ~/scripts/a.sh`
- Exercise
  - Create a gzip-compressed tar archive named `allinfo.tar.gz` containing all files ended in `info` that are part of the `/proc/` folder

Extracting archives using `tar`

- Format: `tar -xf <opt> <ipath> [p1] … [pn]`
  - `-x` or `--extract` specifies extraction, `-f` or `--file` specifies file mode
  - The `-v` and `-z` options still apply
  - `ipath` is the input archive
  - `[p1] … [pn]` represent the files from inside the archive that you want to be extracted. If none are specified, everything will be extracted.
    - We can list the contents using `tar -tvf <ipath>` or `tar -tzvf`
- Examples
  - `tar -tzvf out.tar.gz` (lists the archive's contents)
  - `tar -xzvf out.tar.gz ~/.bash_profile`
  - `tar -xzvf out.tar.gz`
- Exercise
  1. List the contents of the current working directory
  2. Extract the file `allinfo.tar.gz` you just created and list again

Redirection

Redirection

- Most programs read from the standard input (keyboard) and write to the standard output (terminal)
  - We can use redirection to alter that behavior
- Redirecting output with `>` and `>>`
  - E.g. redirect echo's output to a file, overwriting existing contents (if any)
    - `echo "Hello hello hello" > file.txt`
  - E.g. append echo's output to a file (preserves existing contents)
    - `echo "how low" >> file.txt`
- Redirecting input with `<`
  - E.g. sorting the lines of a file alphabetically
    - Background: running `sort` will take in lines from keyboard until Ctrl-d is pressed and then output them sorted alphabetically
    - `sort < file.txt` will print the lines of sorted alphabetically
- Consider the command: `sort < file.txt > file2.txt`
  - What will this command accomplish?

Pipes

- We can use redirection and a file to connect a program's output to another's input
  - E.g. count files: `ls > file && wc -l < file && rm file`
    - `wc -l` counts the number of lines in the input
      - `wc` without options counts number of characters, words and lines
    - Cumbersome and inefficient (and need to remember to subtract 1)
- A `|` (pipe) allows us connect one program's output to another's input very easily
  - E.g. we can do the same as above with `ls | wc -l`
    - Cleaner and more efficient since we're not creating any superfluous files!
- We can use multiple pipes to create a pipeline that works in a assembly line-like fashion
- Exercise:
  - Write one command that'll create, in your home folder, a file `netc.txt` containing the number of files in folder `/etc/`

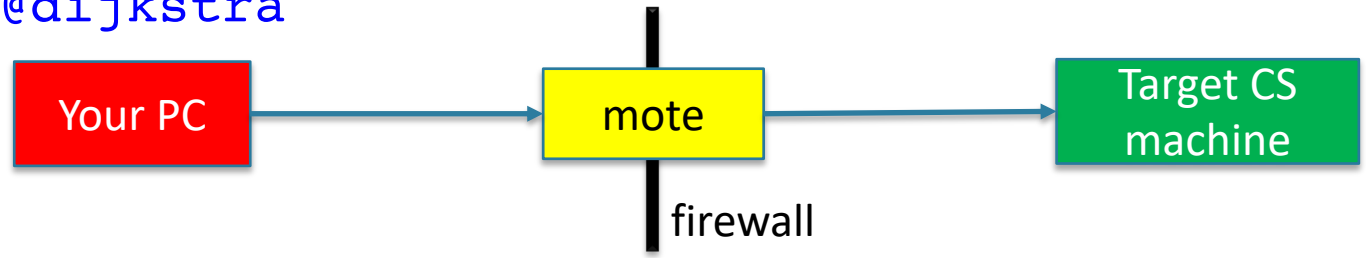
Networked computing

Remote computing with `ssh` (1/3)

- `ssh` (Secure Shell) allows us to work on a remote computer
  - Useful when some resource (e.g. data or software) is only available remotely
  - E.g. work on one of the CS machines from home
- Simplified command format:
  - `ssh [-p <port>] <user>@<hostname>`
  - `hostname` is the name of the computer you're trying to connect to
  - Note this is a simplified version, there are a myriad of options
- Example
  - `ssh -p 443 rui@mote.cs.vassar.edu`
    - The `cs.vassar.edu` suffix is only needed if we're outside the CS network
- Remote graphical session
  - Option `-Y` can be used to allow remote execution of graphical programs
  - E.g. `ssh -Y <user>@<hostname>`, then run e.g. `firefox`
  - For this you'll need X-Windows support on the client machine. Linux has it, and you can install XQuartz on macOS and Xming on Windows.

Remote computing with `ssh` (2/3)

- Exercise
  1. Use `ssh` to login into `mote.cs.vassar.edu`. Be sure to use the `-Y` option.
  2. Run `who` to see who else is logged on to `mote`
  3. Run `firefox` to browse the web. The browser will display locally but all computation will occur remotely.
- SSH tunneling
  - You can use `mote` as a proxy to connect to any CS machine from outside the CS network (let's you work on assignments from anywhere in the world!)
  - Command: `ssh -t -p 443 <user>@mote.cs.vassar.edu "ssh <user>@<target_hostname>"`
  - E.g. `ssh -t -p 443 rui@mote.cs.vassar.edu "ssh rui@dijkstra"`

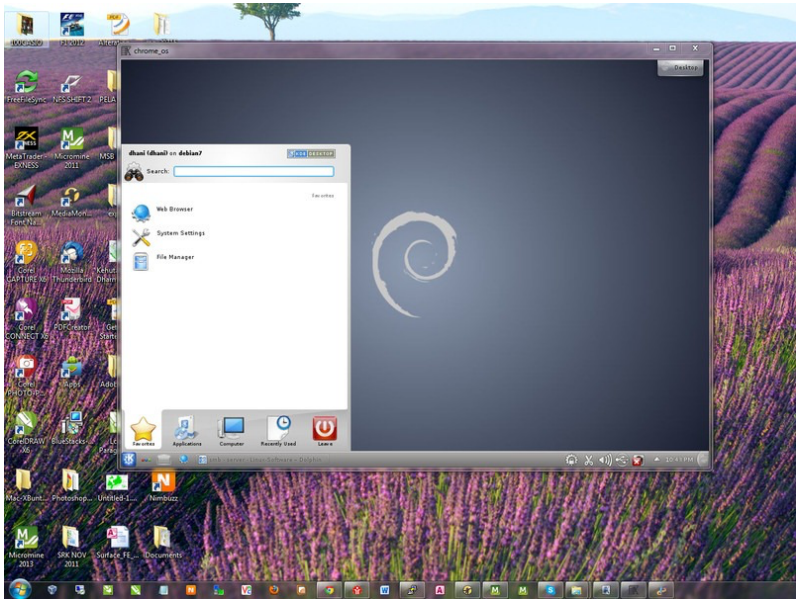


Remote computing with `ssh` (3/3)

- Session maintenance with `screen`
  - Will prevent disconnections from terminating running processes
  - Basic instructions:
    1. Once logged on through `ssh`, run `screen` to start a new session
    2. Detach by hitting Ctrl-`a` `d` or by disconnecting from `ssh`
    3. Reconnect using `screen -r`
- Using RSA keys to enable password-free `ssh` authentication
  - Pair of keys: one public, one private
  - Authentication works by using public key to encode a challenge that only the private key's holder can solve to
  - Basic instructions:
    1. Run `ssh-keygen -t rsa` to generate key pair
       - You'll be prompted for options, defaults are OK
    2. Run `ssh-copy-id [-p <port>] <username>@<server>` to copy public key to server
       - E.g. `ssh-copy-id -p 443 rui@mote.cs.vassar.edu`

Remote desktop with X2Go

- Allows you to access a full graphical environment remotely
- Available for Linux, macOS and Windows
- Compared with plain SSH
  - \+ More user-friendly
  - – Consumes more bandwidth, can lag if connection isn't great
- Learn more at https://www.cs.vassar.edu/help/general_linux/x2go



Remote file copying (1/2)

- `scp` let's us use the SSH protocol to copy files between computers
- Copy files from local machine to remote computer
  - `scp [-P <port>] <p1> … [pn] <username>@<hostname>:<dst>`
    - `p1 … pn` are the files to be copied (need at least one)
    - `dst` is the destination file or folder (there can be only one destination)
  - Examples
    - `scp a rui@dijkstra.cs.vassar.edu:/var/tmp/newa`
      - Copy file `a` to `/var/tmp/newa` at `dijkstra`. When we're copying a single file we can rename it.
    - `scp a b rui@dijkstra.cs.vassar.edu:~/`
      - Copy files `a` and `b` to `rui`'s home folder at `dijkstra`. Renaming isn't possible here.
  - `-r` option used to copy folders recursively
    - E.g. `scp -P 443 -r testdir rui@mote.cs.vassar.edu:~/`
  - Existing files of same path are overwritten

Remote file copying (2/2)

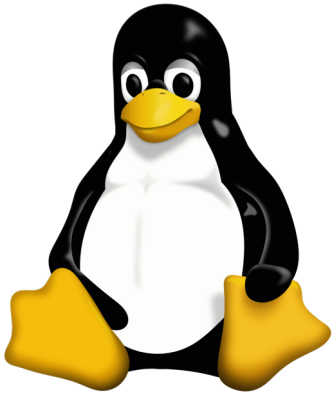
- Copy files from remote computer to local machine
  - `scp [-P <port>] <username>@<hostname>:<src> <dst>`
    - `src` can be a single path or multiple; if it's multiple they need to be enclosed in "" and separated by spaces
    - `dst` is the destination file or folder (there can be only one destination)
  - Again, `-r` is used to copy folders recursively and existing files are overwritten
  - Examples
    - `scp rui@dijkstra:"/etc/resolv.conf ~/a"  .`
      - Copy files ~/a and `/etc/resolv.conf` from `dijkstra` to current working dir
    - `scp -r rui@dijkstra.cs.vassar.edu:~/test ~/`
      - Recursively copy folder `/tmp/` from `dijkstra` to home folder on local machine
- Exercise
  1. Copy `/etc/resolv.conf` from `mote.cs.vassar.edu` to your machine
  2. List the contents of the file you just copied

Web publishing

- The CS department runs a Linux web server that you can use to create your own personal site, share files, etc
- Your home folder contains a `public_html` subfolder
- Files in that subfolder are web-accessible at `https://www.cs.vassar.edu/~<yourusername>/`
  - As long as they're set as readable to everyone
- Home folder is remote-mounted so you can edit in any CS machine
- Exercise:
  1. Create a text file named `hello.txt` with the contents "`Hello web world!`" inside of your `public_html` folder
  2. Make sure your `hello.txt` file is readable by everyone
  3. Use your favorite web browser to navigate to `https://www.cs.vassar.edu/~<yourusername>/hello.txt`

Conclusion

That's all folks!



- Today was only a brief introduction
  - Working with Linux and the Unix shell
  - Working with files
  - Remote computing and network file copy
- You can learn a lot more online:
  - Linux tutorial on CS wiki: https://www.cs.vassar.edu/help/top
  - Distribution-specific help forums, e.g. https://askubuntu.com
  - General online searching (Google, Bing, DuckDuckGo)
- I definitely recommend trying it out on your computer
  - You can dual boot Linux or install it on a virtual machine (Virtual Box is good & free virtualization software)
  - Also remember macOS has bash pre-installed and you can activate it on Windows 10 as well