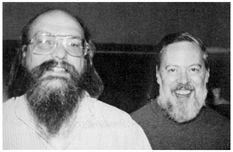
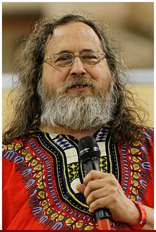
History

- UNIX operating system, the commercial ancestor of Linux, was developed by Ken Thompson and Dennis Ritchie at AT&T Bell Labs in 1969 and released in 1970



- Later several more commercial UNIX distributions appeared
- In 1983, Richard Stallman started the GNU project with the goal of creating a free UNIX-like operating system.



History

- By the early 1990s, there was almost enough available software to create a full operating system. However, the GNU kernel did not work out.
- In 1991, while studying computer science at University of Helsinki, Linus Torvalds, at the age of 21, began a project that later became the Linux kernel

- GNU/Linux provided free open source UNIX environment without expensive license fees
- Nowadays GNU/Linux is usually just called Linux

History

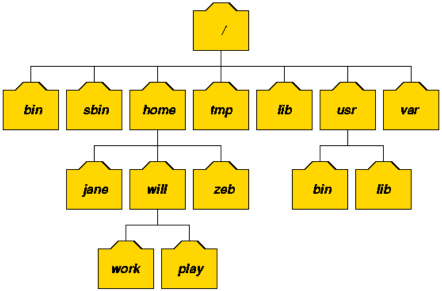
- It is free, open source and a lot of developers and companies from all around the world keep contributing to Linux kernel and application software
- When I started using Linux in 1994, it felt almost illegal:
  - your employer would typically be very suspicious if you are not using the blessed Windows
  - no vendors would talk to you if you say you need their hardware to behave under Linux
  - due to no vendor support one had to be very careful when buying hardware to make sure that all the components would work with Linux
  - it was quite an adventure to install it
  - many useful applications were not available under Linux and one had to use Windows for those
- These days Linux is the mainstream server platform, it is used on almost all HPC sites, embedded devices, etc.
- It is used by such companies as Google, Facebook, Oracle, banks from Wall Street, etc.

History

- Even Microsoft these days relies on Linux in their Azure cloud platform and is one of the major contributors to Linux
- There is a good vendor support now, especially for servers and HPC systems, although still not necessarily for consumer laptops/desktops
- There are free and commercial applications to do anything under Linux and since the beginning of the century I use Linux for everything and have no use for Windows
- It is easy to install Linux today
- There are hundreds distributions to select from: Debian, RedHat, Scientific Linux, Fedora, Slackware, Gentoo, etc.
- Two big families of distributions: Debian-based and RedHat-based
- For a typical user it is recommended to start with one of the most popular distributions: for example, Ubuntu or CentOS - huge user community, easy to find answers to your questions, more free and commercial software works out of the box, more frequent updates, security patches are available

Navigating the file system: directory tree

- Linux, like any other OS I have seen so far, organizes directories and



  files into a tree
- At the top of the tree there is root `/`
- The linux system commands, like ls, cp, mv, are typically in `/bin`
- Application programs, like emacs, vi, find, are typically in `/usr/bin`
- Commands for system administrators are usually in `/sbin`
- Libraries are usually in `/lib`, `/lib64`, `/usr/lib`
- User home directories are under `/home`
- Home directory of the root (system administrator user) is in `/root`

File editing: emacs

- `emacs [<filename>]` starts GUI version if X is forwarded, otherwise, it either starts terminal version or complains
- `emacs -nw [<filename>]` starts terminal version
- Just type your text
- To cut everything from the current character to the end of line: `Ctrl+k`
- To paste: `Ctrl+y`
- To save a file: `Ctrl+x Ctrl+s`
- To save and exit: `Ctrl+x Ctrl+c`
- To mark a big part of text, `Esc Shift+@`, move cursor, do something with the selected text, for example `Ctrl+w` to cut it.
- To search text forward for some string `Ctrl+s`, to search backward `Ctrl+r`
- To move cursor to the end of the line `Ctrl+e`, to move cursor to the beginning of the line `Ctrl+a`

File editing: emacs

- To replace a string starting from the current character till the end of file: `Esc+x replace-string`, type source and target.
- You can open several files inside emacs and keep switching between different buffers, copying and pasting between different files.
- To open an existing file or start a new one in emacs `Ctrl+x Ctrl+f`
- To list open files (buffers): `Ctrl+x Ctrl+b`
- To switch between different buffers: `Ctrl+x b`
- One can open multiple windows, each looking at the same or different buffers.
- To open a new window below: `Ctrl+x 2`
- To open a new window to the right: `Ctrl+x 3`
- To remove other windows: `Ctrl+x 1`
- To move to other window: `Ctrl+x o`
- One can cancel a command in construction with `Ctrl+g`

File editing: emacs

- By default emacs shows line number on the bottom bar. One can ask it to show also column number with `Esc+x column-number-mode`
- To undo, `Ctrl+x u`
- There are thousands of other commands and options but the above is typically enough for 99% of text editing
- You can define your own macros
- Emacs usually detects what language you are programming in and formats text accordingly. It can recognize C, C++, Fortran, Python, Java, HTML, LaTeX, etc. Language specific menu appears in GUI version of emacs.
- The other popular editor in Linux which we do not discuss here is `vi`.
- There are many other higher level and more user friendly editors like gedit, nano, lyx, libreoffice, atom, etc. but you better know either emacs or vi since those are available everywhere.

File editing: gedit

- If you do not want to spend a few hours to get used to the power of emacs, just use gedit
- It has similar intuitive GUI interface to Window's notebook, nothing to learn
- However, you need either to use ssh client that does X-forwarding or use ThinLinc

Copying and moving files and folders

- To copy files on a Linux system, use `cp`, to copy folders, use either `cp -r` or `rsync -av`
  - `cp <fromfile> <tofile>`
  - `cp -r <fromdirectory> <todirectory>`
  - `mv <fromfile> <tofile>`
  - `rsync -av <fromdirectory> <todirectory>`
- `rsync`, contrary to `cp`, is restartable, if it is interrupted in the middle and you start it again, it figures out what has already been done and picks up where it left.
- To copy file/directory to remote host, if you have ssh client, do

```
scp <fromfile> <yourusername>@midway2.rcc.uchicago.edu:<tofile>
scp -r <fromdirectory> <yourusername>@midway2.rcc.uchicago.edu:<todirectory>
rsync -av <fromdirectory> <yourusername>@midway2.rcc.uchicago.edu:<todirectory>
```

- To copy file/directory from remote host, if you have ssh client, do

```
scp <yourusername>@midway2.rcc.uchicago.edu:<fromfile> <tofile>
scp -r <yourusername>@midway2.rcc.uchicago.edu:<fromdirectory> <todirectory>
rsync -av  <yourusername>@midway2.rcc.uchicago.edu:<fromdirectory> <todirectory>
```

- `rsync` uses `ssh` underneath unless `rsync` server is running on the remote site

Copying and moving files and folders

- If you have a lot of data to copy between your laptop and midway or between midway and some other Globus end point, use Globus. For instructions, see

  `https://globus.rcc.uchicago.edu/globus-app`

- If ssh/scp still does not work for you, you can use JupyterHub to upload/download data to/from midway

  `https://jupyter.rcc.uchicago.edu`

- If you are transferring text files from Windows/Mac to Linux or the other way arround, you might need to run `dos2unix` or `unix2dos` on those files since the end of line character in Linux and Windows/Mac text files is different which might cause problems for some programs.

File and directory permissions

- For each file or directory you can set who can
  - r - read
  - w - write
  - x - execute (or browse for directory)

  it.
- For permissions purposes, there are four types of “who”:
  - u - user
  - g - group
  - o - others
  - a - all
- There is ACL (access control list) that gives finer control over permissions but I had never any reason to use it so far.
- There are permissions that each file and directory gets by default. The default is configurable.

File and directory permissions

- You can see permissions with ls -l:

  ```
  ls -l t8
  -rwxrw-r-x 2 ivy2 kicp 243640 Dec 11 09:38 t8
  ```

  The above says that a file `t8` was last modified on `Dec 11 09:38`, has `243640` bytes, belongs to `ivy2` user and `kicp` group (often default group is same as username), has only two hardware links to it, is a file (for directory the first character would be `d`, for symbolic links it is `l`), has `rwx` permissions for user, `rw-` permissions for the group, `r-x` permissions for others.
- The executable permission on a file means that it can be used as a program.
- The executable permission on a directory means that one can go there.

File and directory permissions

- To change permissions:

  ```
  chmod u-w t8
  chmod -R a+X dir1
  ```

- The above would remove write permissions for the user and makes all directories under dir1 browsable by everybody. If one used 'x' instead of 'X', all the files under dir1 in addition would become executable.
- The ownership of the file/directory is changed with chown but as a user you have rather limited freedom to do it. Only root can change user ownership. You can change group ownership to one of the groups of which you are a member:

  ```
  chown ivy2:rcc-hadoop t8
  ```

- You can find to which groups you belong by executing groups (standard Linux command) or `rcchelp user <username>` (local RCC script)

Bash shell

- When you log into Linux, you get into shell - command line interface (CLI) to Linux - where you can type commands and the shell would interpret and execute them.
- By default it is `bash`.
- There are other shells that you can select: `csh`, `tcsh`, `zsh`, etc. They mostly differ in syntax but have similar functionality.
- bash configuration is defined in the following files in your home directory: `.bash_profile`, `.bashrc`, `.bash_aliases`
- There you can set environmental variables, aliases, define, for example, how your shell prompt looks like, etc.
- If you start typing a command or a path in shell and press TAB, the shell tries to autocomplete
- `history` shows the previous commands, arrow-up returns the previous command if you want to repeat it
- bash understands many of emacs key combinations: `Ctrl+A`, `Ctrl+E`
- Shell can be used as a programming language to write scripts

Environment

- When you type the name of the executable program in a shell, it is looking for it among the paths listed in `$PATH` environmental variable. For example:

```
$ echo $PATH
/bin:/software/slurm-current-el7-x86_64/bin:\
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:\
```

- If you have a program in some non-standard location `/yet/another/path/bin/myprogram`, you need to prepend/append another path to `PATH` or use the full path to the executable:

```
$ /yet/another/path/bin/myprogram
$ export PATH=/yet/another/path/bin:$PATH
$ myprogram
```

- To find out where your program is, if its location is in `$PATH`:

```
$ which gcc
/usr/bin/gcc
```

Environment

- Notice that bash would go over the list of paths in order specified and will pick the first available executable with the given name it can find and stop looking after that. Therefore, it might make difference if you prepend or append paths.
- Most of the programs in Linux are dynamically compiled, in the sense that they load the necessary libraries at runtime rather then including them into the executable. If the library you are using is not at some standard location like `/lib64`, `/usr/lib`, `/usr/lib64`, `/usr/local/lib`, etc, you need to prepend/append the corresponding path to `$LD_LIBRARY_PATH` environmental variable.
- To see what libraries are loaded or not found for a particular executable:

```
$ ldd ‘which gcc‘
linux-vdso.so.1 =>  (0x00007ffdbcba1000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff4b95a6000)
/lib64/ld-linux-x86-64.so.2 (0x00007ff4b9970000)
```

Environment

- The program behavior might be modified by various environmental variables specific to the program.
- For example, programs that use OpenMP multithreading would take the number of threads to use from `$OMP_NUM_THREADS`
- You can learn the path to your home directory, username, current directory:

```
$ echo $HOME
$ echo $USER
$ echo $PWD
```

- This is useful for writing portable programs that do not hardcode such information but learn it on the fly from the environment.
- To see the values of all the current evironmental variables use `printenv`
- If you use python and have some python libraries installed separately from python distribution, add the corresponding directory to `$PYTHONPATH`

Modules

- If you are running Linux on a desktop or laptop, you would typically have one version of each program installed in some standard directory
- In HPC environment, like midway, this is unrealistic: different users want different versions of the same program.
- As a result, applications are typically installed in a non-standard location and one must set environmental variables, at least `PATH` and `LD_LIBRARY_PATH` to select a particular version of a particular program
- To simplify this task we use `modules`
- You can see which modules are available by running `module av`
- You can load the module with `module load <name>/<version>`.
- You can unload the module with `module unload <name>/<version>`.
- If you do not specify version, default is used
- To unload everything, use `module purge`.

Input/Output redirection and pipes

- You can redirect output from one program, standard output, to a file:

```
ls -l > /tmp/my.out; pwd >> /tmp/my.out; cat /tmp/my.out
```

- Or you can use the output from one program as an input to another program with pipes. Most of Linux commands are written as filters that can take input either from a file or standard input, transform it, and print the results to standard output. For example:

```
ls -l | sort
cat /tmp/ls.out | grep test | grep -v something | tail -2
grep test < /tmp/ls.out > /tmp/ls_1.out
```

- If a program prints out to a screen an error or a warning, it is often done to a different stream, standard error, that can be treated separately from standard output and can be redirected to a separate file or merged with standard output. If you do not redirect it somewhere, it will be printed to the screen.

```
nohup ./myprogram > myoutput 2>myerror &
nohup ./myprogram > myoutput 2>&1 &
```

Process control

- ps - reports a snapshot of the current processes.
- There are hundreds of options but usually I use only two:

  ```
  ps
  ps -ef | grep <...>
  ```

- top - provides a dynamic real-time view of a running system, sorts the processes by load, memory usage, etc
- If you need to kill a running process, find its pid with either ps or top and then

  ```
  kill <pid>
  ```

  or, if it does not work, as the last resort

  ```
  kill -9 <pid>
  ```

Process control

- `&` - if you put it at the end of the command, it is executed in the background and you can continue using the shell; however, would not persist after you disconnect from the terminal

  ```
  <program> &
  ps
  ```

- `nohup` - execute a program in a background, it would still run after you disconnect

  ```
  nohup <yourprogram> > <output> 2>&1 &
  ```

- You can put the existing foreground program into background with `Ctrl+Z`
- To bring it back into foreground, `fg`
- It is sometimes useful to insert into your scripts

  ```
  sleep <N>[s|m|h]
  ```

  to make the program sleep for N seconds/minutes/hours

Miscellaneous useful commands

- `grep` - search for strings in a file
- `sort` - sort text
- `uniq` - eliminate duplicate lines in text
- `head`, `tail` - show first/last *n* lines in a text file
- `history` - list previous commands
- `cut` - cut columns from a tabular file
- `paste` - merge columns from different files
- `alias` - define an alias for a command or combination of commands
- `cat` - print a file to screen
- `more`, `less` - show a text file, page by page
- `gzip` - compress file
- `tar` - archive directory tree into a file

Wildcards

- `ls z*.out` - show a list of files in the current directory that start with `z` and end with `.out`. There can be any number of characters in between or nothing
- `ls t?at` - show a list of files that start with `t`, end with `at` and have one character in between

Shell programming

- Shell can be used as a programming language
- You can set environmental variables, define aliases, functions, list commands to execute
- There are flow control structures like if conditional statements, loops, functions, etc.
- I usually just use shell to set up environment, aliases, to list commands; for more complicated scripting I would rather use perl or python which are much more convenient to program
- To get command line arguments into shell script:
  - $0 - name of the file with script
  - $1 - name of the first argument, etc.
- If you want to know more about bash shell programming, take a look at:

  `http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html`
  `http://tldp.org/LDP/abs/html/`