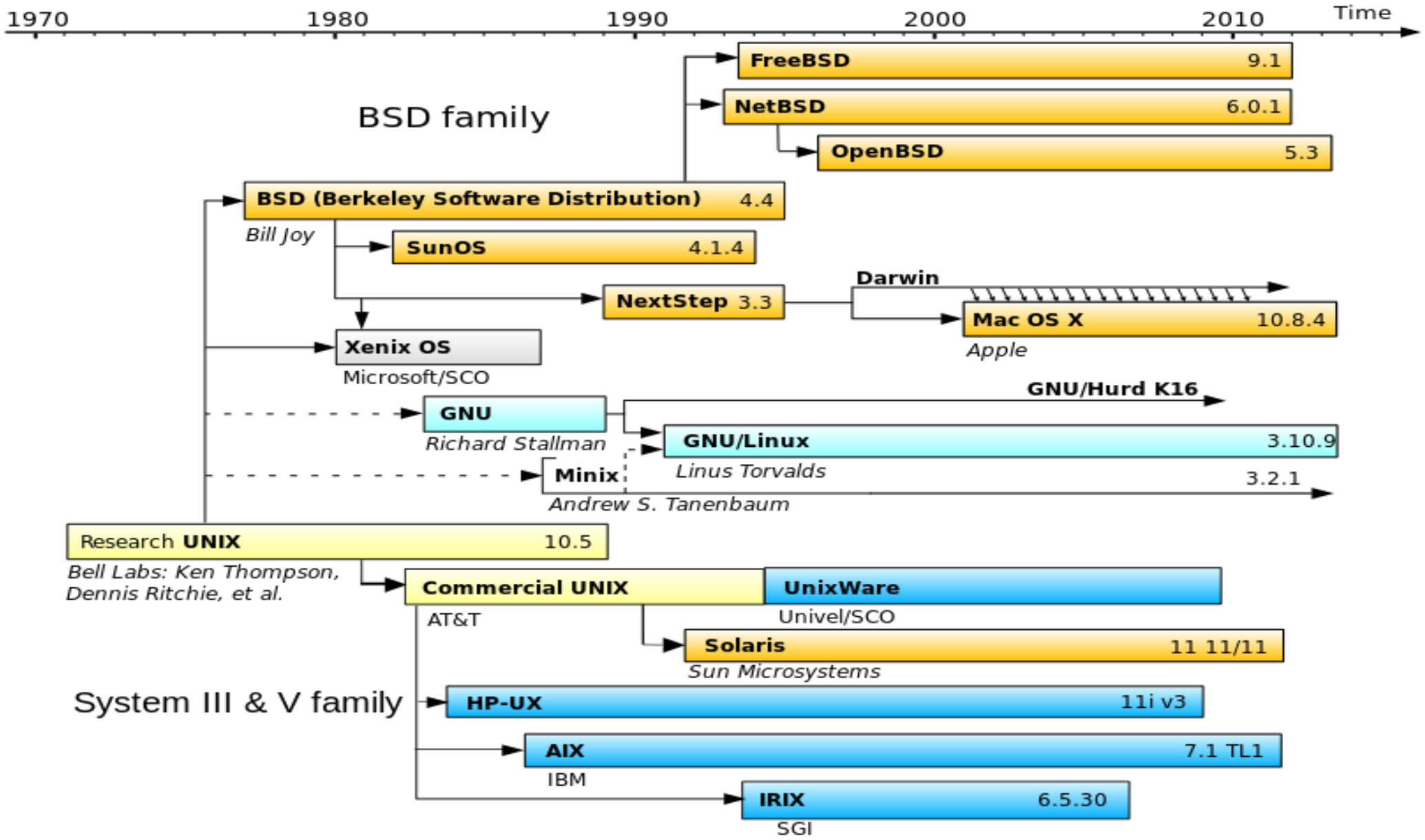
CS 288 Intensive Programming in Linux

Professor Ding, Xiaoning

Linux was a Unix-based OS



Brief Intro of Linux

- Created in 1991 by Linus Torvalds
- PC-based operating system
- Based on the existing UNIX operating system
- Released in 1994 as Version 1
- Initially was developed for 80x86 processors (IA32 or i386 architecture processors)
- Today it support various processor
  - AMD, ARM, Power PC, etc.

Linux is popular but not well taught

"You use Linux every day but you don't know it. It's such a fundamental part of our lives. It runs air traffic control, it runs your bank, and it runs nuclear submarines. Your life, money, and death is in Linux's hands …”

--- Jim Zemlin, executive director of the Linux Foundation

- Which courses have taught you a lot of Linux knowledge?

Linux distributions (distro) and Linux OS kernel

- Each distro is a package including the OS and different applications
  - Linux OS kernel.
  - X Window System and GUI interfaces.
  - Different applications
    - vi, Libre Office, Tex, photo editor, etc.
  - Documentations
- Different distros provide different applications
  - Major distros: Ubuntu, Linux Mint, Fedora, MX Linux, …
  - Many many different distros: http://en.wikipedia.org/wiki/List_of_Linux_distributions
  - Distribution suggested in the class: Ubuntu 18.04.3 LTS or above
    - **Ubuntu terminal and Wubi are NOT suggested to use in the class.**

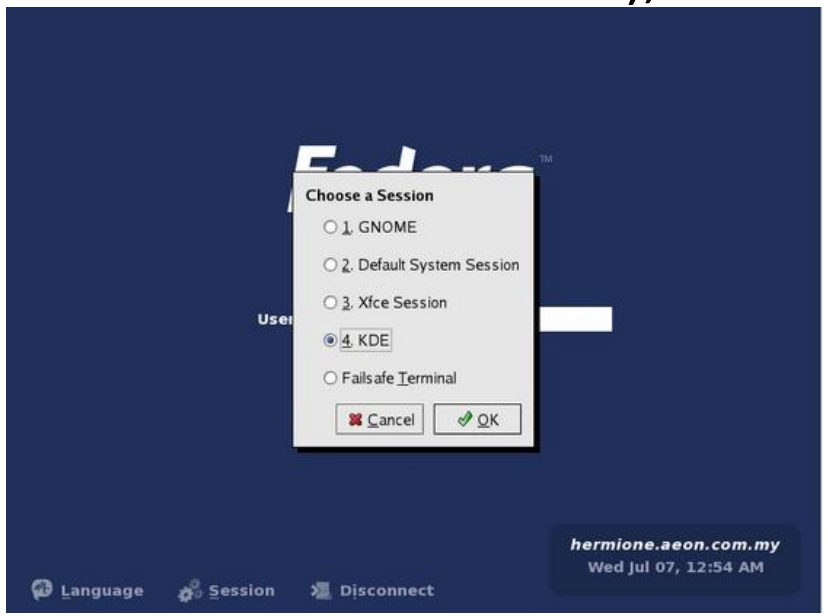
Linux command line interface (CLI)

- CLI is provided by a Linux **shell**
  - The first thing you see after you log into the server.
  - Interact with you and run your commands.

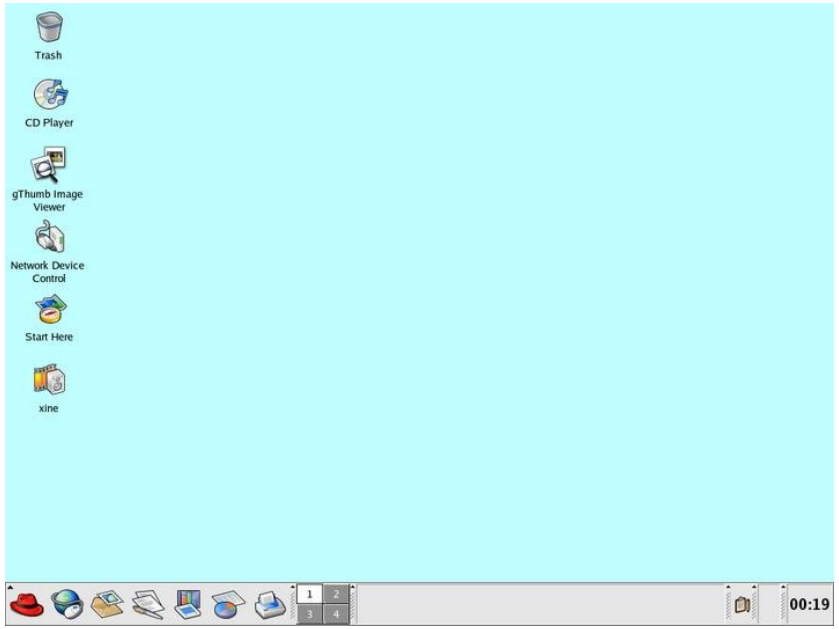
- A variety of Linux shells are available
  - bash, Bourne Shell (sh), C Shell, tcsh, scsh
  - We use **bash** in the course

Linux graphic user interface

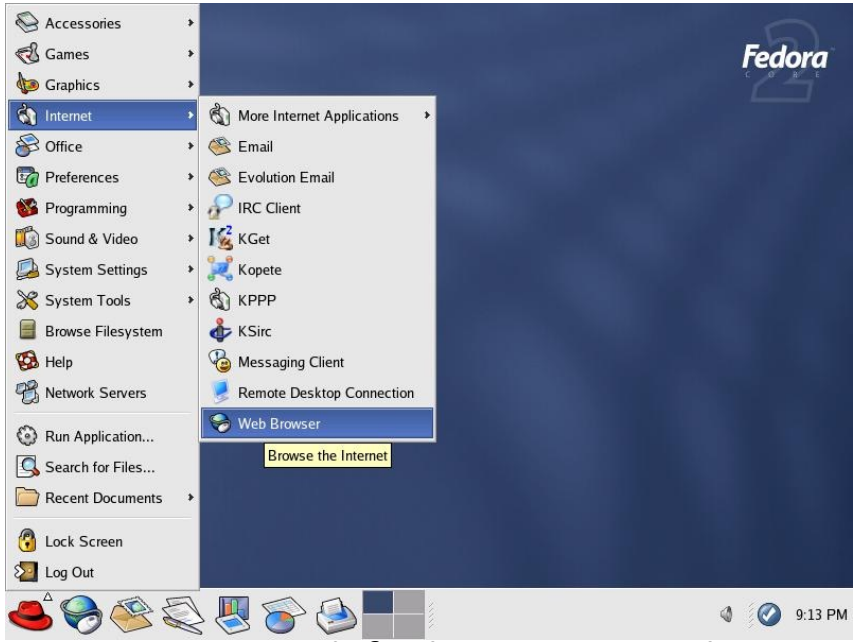
- X Window System or X provides standard mechanisms for displaying device-independent, bit-mapped graphics
- How the actual interface looks or feels depends on the GUI interface
  - KDE (K-desktop Environment), GNOME (GNU Network Object Model Environment), etc.



Choosing KDE at the Login Screen



The KDE Desktop



default GNOME Desktop

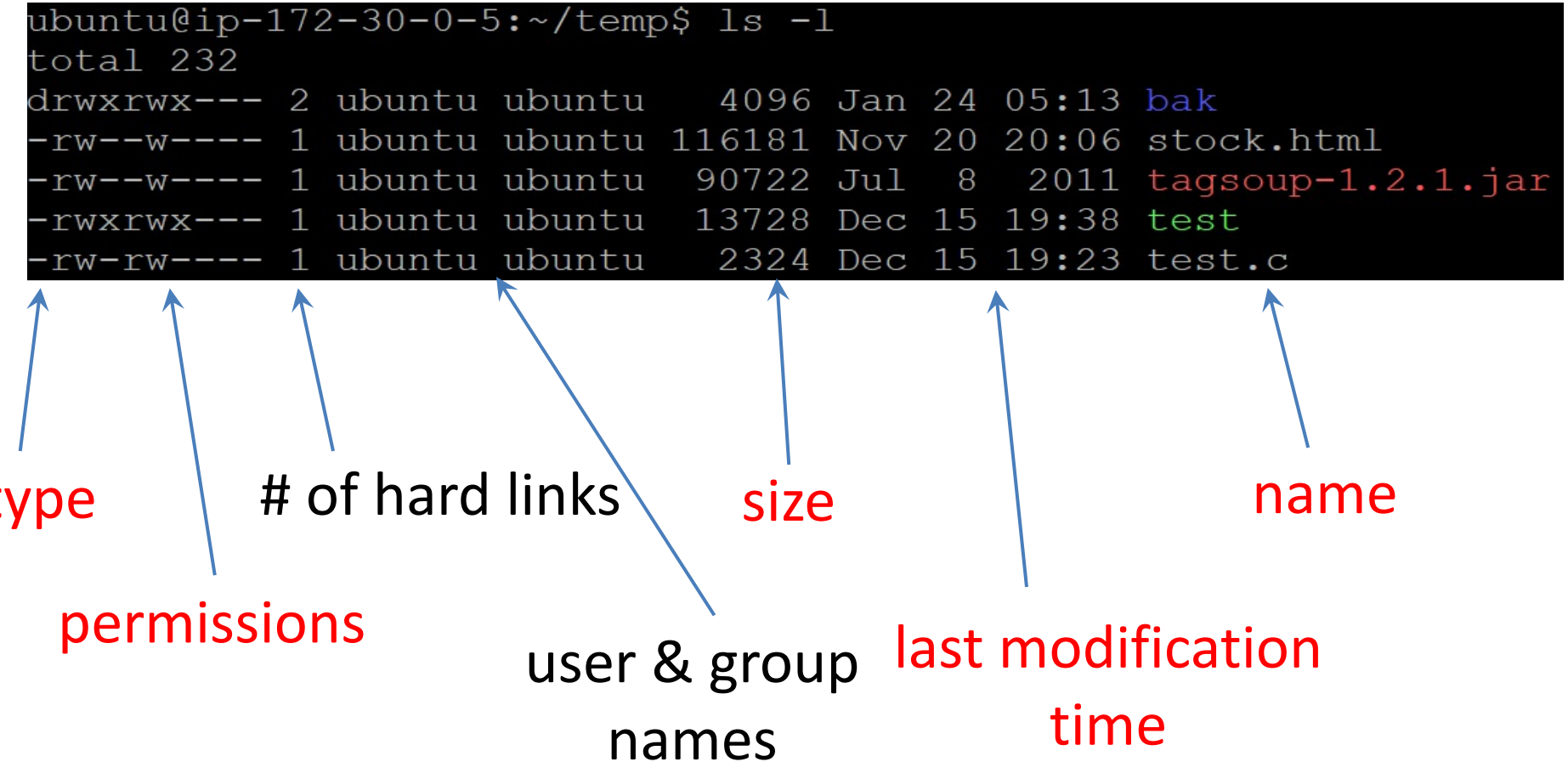
You must know how to use ***help*** command

- help command displays information about shell built-in commands.
- It is also useful when writing a shell script.
- "`help`" or "`help topic`"
  - "`help`" prints the list
  - "`help topic`" prints the help page of the topic.
- what is "`help help`"?
  - Help page of the "help" command.

You must know how to use ***man*** command

- e.g., `man chmod`
- man [section #] topic
  - 1 Executable programs or shell commands (e.g., `man 1 read`)
  - 2 System calls (functions provided by the kernel) (e.g., `man 2 read`)
  - 3 Library calls (functions within program libraries) (e.g., `man 3 read`)
  - 4 Special files (usually found in /dev)
  - 5 File formats and conventions eg /etc/passwd
  - …
- Most manuals can be found online
  - Try to google “man 2 read”

File and directory information

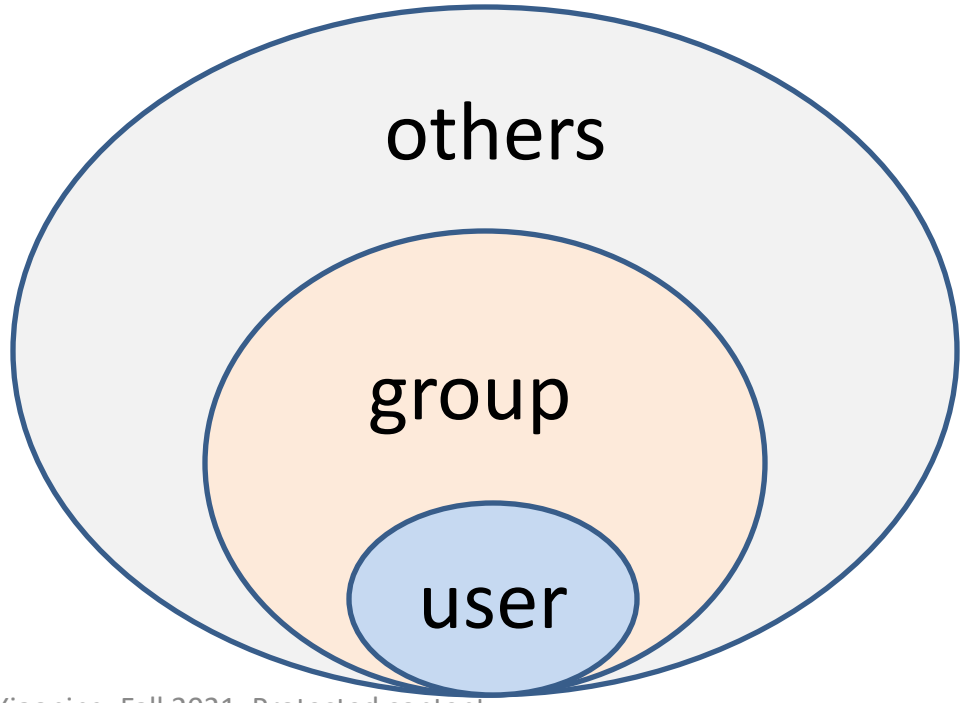


File system permissions in Linux

Permission type	**When used with files**	**When used with directories**
Read	Read a file or copy a file	List the contents of a directory
Write	Write to the file, including deleting the file	Create files
Execute	Execute programs and shell scripts, which are text files containing Linux commands	Modify the file permissions

Linux Permissions

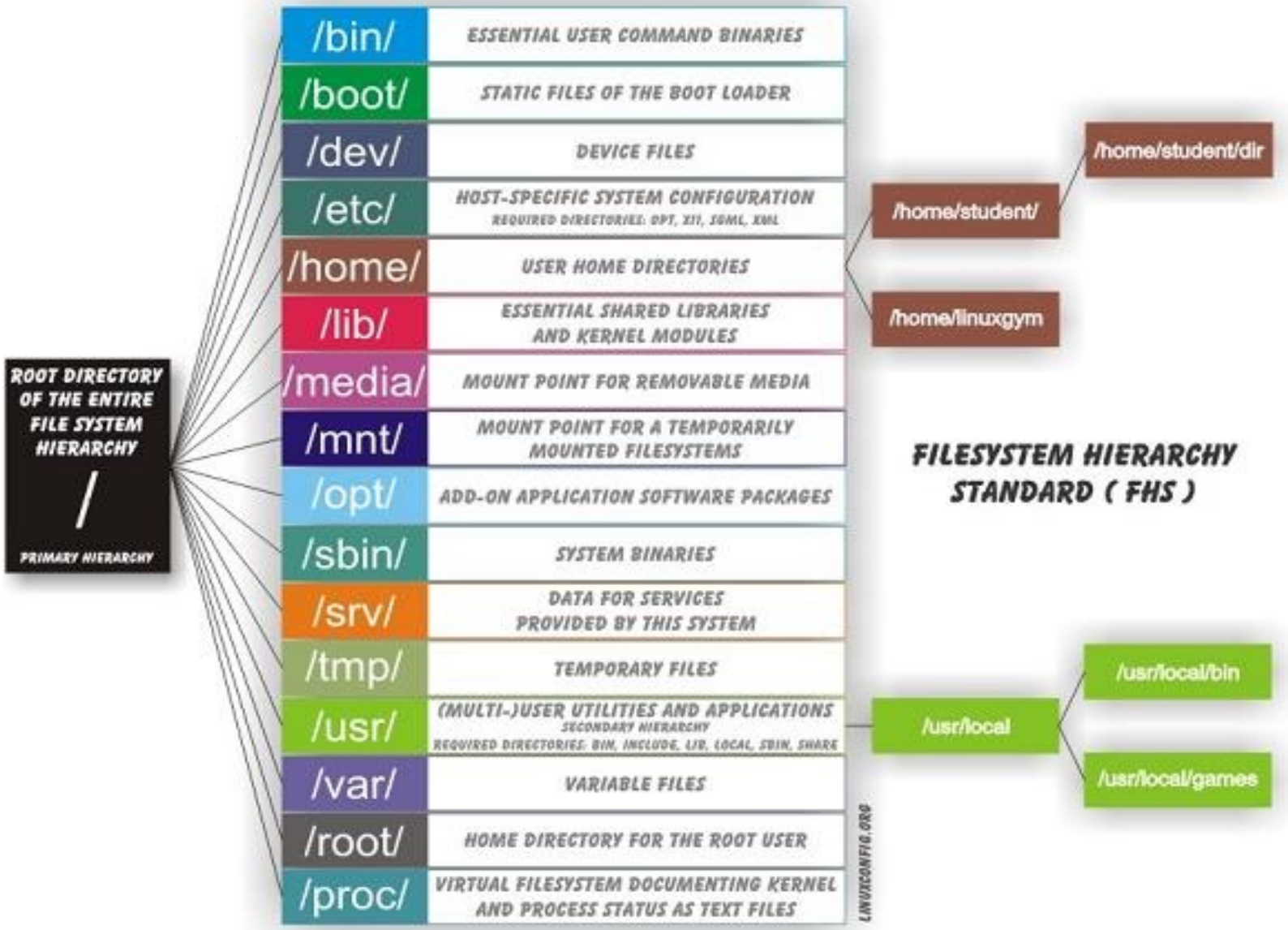
- Permissions are set for user, group, and others
- Each permission is set with a single digit from 0 to 7 based on the combination of permissions
  - read = 4
  - write = 2
  - execute = 1



Using chmod to Set Permissions

Command	**Permissions**		
	Owner	**Group**	**Other**
`chmod 755 myfile`	rwx 111	r-x 101	r-x 101
`chmod 540 myfile`	r-x 101	r— 100	--- 000
`chmod 744 myfile`	rwx	r--	r--

Linux directory hierarchy



File pathname

- Pathname: location and name of the file.
- Remember root is **/**, current directory (current working directory, pwd) is **.**, parent directory is **..**, home directory is **~**.
- Absolution pathnames start from root.

  $ cd /home/CS288/tom
- Relative pathnames are usually relative to the current directory.

  $ cd ./hw/hw1

  $ cd ../hw2

Some commands

- ls – list files

```
$ ls /home/john
```

- cd – change directory

```
$ cd /tmp
```

- pwd – current working directory
- mkdir – create a directory

```
$ mkdir /home/alice/proj1
```

- echo- display a line of text

```
$ echo "hello world"
```

- cat - concatenate files and print on the standard output

```
$ cat a.txt b.txt
```

  - With no file, or when file is -, read standard input.

Some commands

- rmdir – remove a directory
  ```
  $ rmdir /home/tom/tmp
  ```
- rm – remove a file/files
  - remove directories and their contents recursively: `$ rm -r`
- mv – move or rename a file
  ```
  $ mv /etc/ftpaccess /var/ftp/ftpaccess
  ```
- cp – copy a file
  ```
  $ cp var/ftp/ftpaccess /home/amy/
  ```
- date -- current time
- time --- run program and report execution time
  ```
  $ time ls /usr/bin
  ```
- du – file size/space consumption
  ```
  $ du -b /home/amy/hw1
  ```
- locate, whereis, which – find a file
  ```
  $ locate ftpaccess
  $ whereis bash
  ```

Linux utility conventions

- POSIX recommends some conventions for command line arguments.
  - Programs implement them. getopt() makes it easy.
  - man and help describes command synopses following them (e.g., help cd).
  - https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap12.html

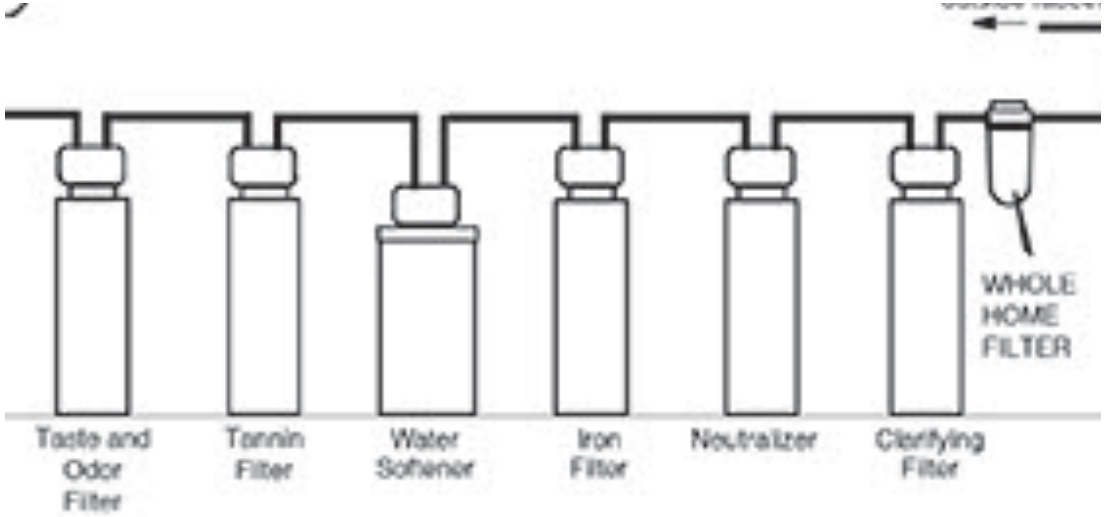
utility_name [-a] [-b] [-c option_arg] [-d|-e] [-f [option_arg]] [operand...]

- Arguments are **options** if they begin with a hyphen delimiter (‘-’).
  - Option names are single alphanumeric characters
  - Multiple options may follow a hyphen delimiter in a single token if the options do not take arguments. Thus, ‘-abc’ is equivalent to ‘-a -b -c’.
  - Certain options require an **option argument**.
- Options typically precede other non-option arguments, named **operands**.
- Arguments separated by '**|**' are **mutually-exclusive**.
- Arguments in '**[**' and '**]**' are **optional.**

A class of Unix tools called ***filters***

- Utilities that read from standard input, transform file contents, and write to standard output
  - Standard input: keyboard, i.e., where your program reads from using scanf()
  - standard output: screen, i.e., where your program prints to using printf().
- Usually work on text files in a line-by-line manner.

```
filter < abc > xyz
```



Redirecting (>, >>, <, |)

- Using > and >> to change standard output to a file
  - e.g., `cmd > filename`
  - *> overwrite, >> append*
- Using < to change standard input to a command.
  - `cmd < file.txt`
- Using | to change standard input and standard output to another program
  - `cmd1 | cmd 2`
- Difference from wring multiple commands on the same line.
  - `cmd1; cmd 2; cmd 3`

cat: the simplest filter

- The cat command copies its input to output unchanged (*identity filter*). When supplied a list of file names, it con**cat**enates them onto stdout.
- Some options:
  - **`-n`** **n**umber output lines (starting from 1)

```
cat file*
ls / cat -n
```

head

- Display the first few lines of a specified file
- Syntax: *head [-n] [filename...]*
  - *-n* - number of lines to display, default is 10
  - *filename...* - list of filenames to display
- When more than one filename is specified, the start of each files listing displays the filename as follows:

```
==>filename<==
```

tail

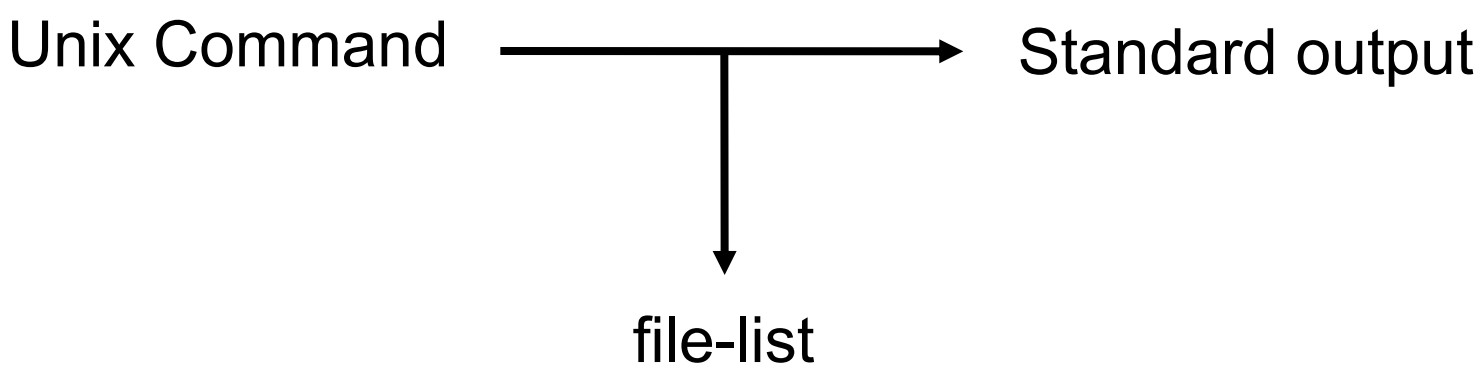
- Displays the last part of a file
- Syntax: *tail -n +|-number [filename]*

  or: *tail +|-number [filename]*
  - *+number* - begins copying at distance *number* from beginning of file
  - *-number* - begins from end of file
  - *if number isn't given, defaults to 10*

head and tail examples

```
head /etc/passwd
head /etc/*.conf
tail +20 /etc/passwd
ls /bin/ | tail -3
head -100 /etc/passwd | tail -5
```

tee



- Copy standard input to standard output and one or more files
  - Captures intermediate results from a filter in the pipeline

tee con’t

- Syntax: *tee [-a] file-list*
  - *-a* - append to output file rather than overwrite, default is to overwrite (replace) the output file
  - *file-list* - one or more file names for capturing output
- Examples

```
ls | head –10 | tee first_10 | tail –5

who | tee user_list | wc
```

Text files as delimited data

Tab separated

```
John      99
Anne      75
Andrew    50
Tim       95
Arun      33
Sowmya    76
```

pipe-separated

```
COMP1011|2252424|Abbot, Andrew John |3727|1|M
COMP2011|2211222|Abdurjh, Saeed |3640|2|M
COMP1011|2250631|Accent, Aac-Ek-Murhg |3640|1|M
COMP1021|2250127|Addison, Blair |3971|1|F
COMP4012|2190705|Allen, David Peter |3645|4|M
COMP4910|2190705|Allen, David Pater |3645|4|M
```

colon-separated

```
root:ZHolHAHZw8As2:0:0:root:/root:/bin/ksh
jas:nJz3ru5a/44Ko:100:100:John Shepherd:/home/jas:/bin/ksh
cs1021:iZ3sO90O5eZY6:101:101:COMP1021:/home/cs1021:/bin/bash
cs2041:rX9KwSSPqkLyA:102:102:COMP2041:/home/cs2041:/bin/csh
cs3311:mLRiCIvmtI9O2:103:103:COMP3311:/home/cs3311:/bin/sh
```

cut: select columns

- The cut command prints selected parts of input lines.
  - can select columns (assumes tab-separated input)
  - can select a range of character positions
- Some options:
  - **-f** *listOfCols*: print only the specified columns (tab-separated) on output
  - **-c** *listOfPos*: print only chars in the specified positions
  - **-d** *c*: use character *c* as the column separator
- Lists are specified as ranges (e.g. 1-5) or comma-separated (e.g. 2,4,5).

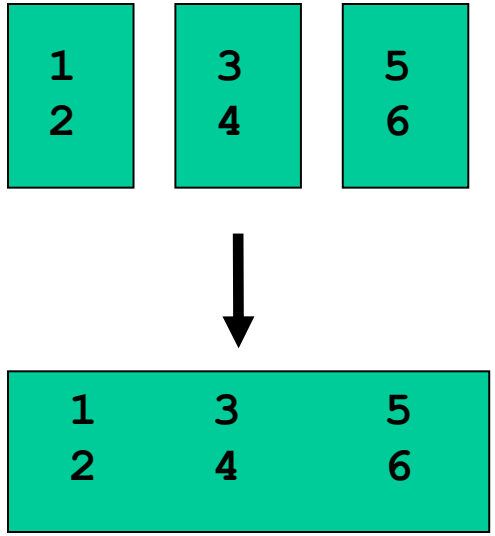
cut examples

```
cut -d':' -f 1 /etc/passwd
cut -d':' -f 1-3  /etc/passwd
cut -d':' -f 1,4  /etc/passwd
cut -d':' -f 4-  /etc/passwd
cut -d':' -f 1-3  data
cut -c 1-4 /etc/passwd
```

Unfortunately, there's no way to refer to "last column" without counting the columns.

paste: join columns

- The paste command displays several text files "in parallel" on output.
- If the inputs are files **a**, **b**, **c**
  - the first line of output is composed of the first lines of **a**, **b**, **c**
  - the second line of output is composed of the second lines of **a**, **b**, **c**



- Lines from each file are separated by a tab character.
- If files are different lengths, output has all lines from longest file, with empty strings for missing lines.

paste example

```
cut -d':' -f 1 /etc/passwd > data1
cut -d':' -f 2 /etc/passwd > data2
cut -d':' -f 3 /etc/passwd > data3
paste data1 data3 data2 > newdata
```

sort: sort lines of a file

- The sort command copies input to output but ensures that the output is arranged in ascending order of lines.
- Very power:
  - can sort based on dictionary order (default) or numeric order.
  - Can specify the key used in the sort in a very flexible way.
    - By default, key is the whole line.

sort: options

- Syntax: *sort [options] [-o filename] [filename(s)]*

-n Numeric order, sort by arithmetic value

-d Dictionary order

-f Ignore case (fold into lower case)

-t Specify delimiter

-k Key

-r Sort in reverse order

-o filename - write output to filename

- Lots of more options…

sort: specifying fields

- Delimiter : **`-t`***d*
- Key: **`-k [start_position][,stop_position]`**
  - start_position and stop_position start from 1 and are inclusive
    - If stop_position is omitted, stop_position is the line's end by default.
  - Formation of each position: f[.c][options]
    - f is a field number, c is a character position in the field
    - options: one or more single-letter ordering options (dictionary? Numeric? Reverse?), which override global ordering options
  - There can be multiple keys.
  - **`-k 3.1,4 -k 1,3 -k4n`**

sort examples

```
sort -t':' -k3nr /etc/passwd
sort -o mydata mydata
sort -t':' -k7 -k3nr /etc/passwd
```

uniq: list UNIQue items

- Remove or report *adjacent* duplicated lines
  - How can you remove duplicated lines that are not adjacent?
- Syntax: *uniq [-cdu] [input-file] [output-file]*

`-d` Write only the duplicated lines

`-u` Write only those lines which are not duplicated

`-c` Supersede the -u and -d options and generate an output report with each line preceded by an occurrence count

- The default output is the union (combination) of *-d* and *-u*

wc: Counting results

- The word count utility, **wc**, counts the number of lines, characters or words
- Options:

  `-l` Count lines

  `-w` Count words

  `-c` Count characters
- Default: count lines, words and chars

wc and uniq Examples

- `who | sort | uniq -d`
- `wc my_essay`
- `who | wc`
- `sort file | uniq | wc -l`
- `sort file | uniq -d | wc -l`
- `sort file | uniq -u | wc -l`

tr: translate or delete characters

- tr [OPTION]... SET1 [SET2]

text transformations, e.g., uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace.

```
echo "url_that_I_have" | tr "_" "-"
url-that-I-have
```

-c first complement SET1

-d delete characters in SET1, do not translate

-s squeeze-repeats

tr: translate or delete characters

```
echo abcdefghijklmnop | tr -c 'a' 'z'
azzzzzzzzzzzzzzz

echo 'Phone: 01234 567890' | tr -cd '[:digit:]'
01234567890

echo 'clean this up' | tr -d 'up'
clean this

echo 'too many spaces here' | tr -s '[:space:]'
too many spaces here

echo Bad\ File\ nAme.txt | tr -d '\\' | tr ' ' '_' | tr
  '[:upper:]' '[:lower:]'
bad_file_name
```

Tools you need to write programs in Linux

- Editor: vi/vim
- Compiler: gcc
- Debugger: gdb

The following three slides give you brief intro.

Though we may not into details in the class, reading through the online materials will greatly help you on finishing the assignments later.

Vi editor (additional materials in canvas)

- Why vi, fast and easy
- Basic modes- edit and command,
  - ‘esc’ for command mode
  - ‘i, a’ for edit mode (insert or append mode)
- Other commands using colon- :q,:w,:q!,:e
  - :q for quit, :w for write, :q! quit without save
  - :e open another file for editing, :wq write and quit
- Searching using ‘/’
  - In command mode use ‘/’ then write the word you want to search
  - ‘n’ for forward search, ‘N’ for backward search
- Search and replace
  - :%s/this/that - will search string “this” and replace with “that”
  - :%s/this/that/gc --- search and replace interactively
- Advanced vi – **vim**(vi improve) and **gvim**(gnu vim)

Write and run a c program in Linux (additional materials in canvas)

- Write your c code:

```
vi  myprog.c
```

- Compile it using gcc

```
gcc myprog.c -o myprog
```

- run it

```
./myprog
```

Debug a program in Linux using gdb (Additional materials in canvas)

- Must compile the program in a special way (to include debugging information)

```
gcc  -g myprog.c -o myprog
```

or

```
gcc  -ggdb myprog.c -o myprog
```

- Run the program in gdb (the debugger)

```
gdb ./myprog
```