CSCI 330

The UNIX File System

Jon Lehuta

Northern Illinois University

August 17, 2020

The UNIX File System - Outline

Northern Illinois University

File System

hierarchical organization of files

- contains directories and files and devices
- *EVERYTHING* is a file
- always single tree

Basic commands to list and manipulate files

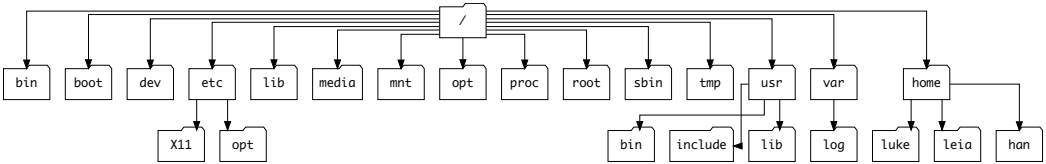
- independent of physical file system organization

Typical Unix file system types

- ext4 (formerly ext2, ext3)
- reiserfs
- also: vfat, ntfs

Northern Illinois University

UNIX file system Layout



Typical UNIX directory structure.

Northern Illinois University

Common UNIX directories

Location	Purpose
`/bin`	Essential command binaries
`/boot`	Static files of the boot loader
`/dev`	Device files
`/etc`	Host-specific system configuration
`/lib`	Essential shared libraries and kernel modules
`/media`	Mount point for removable media
`/mnt`	Mount point for temporary file systems
`/opt`	Add-on application software packages
`/proc`	data on running system
`/root`	home directory for system administrator
`/sbin`	Essential system binaries
`/home`	Contains home directories for non-administrators
`/tmp`	Temporary files
`/usr`	Secondary hierarchy
`/var`	Variable data (logs, mostly)

Northern Illinois University

Directory content

- Regular files
  - *file data* is always stored as a collection of bytes
  - Text mode – the bytes in *text files* represent human readable characters only (ASCII/Unicode, etc)
  - Binary mode – can contain values outside of the human readable range
  - Executable – can be run as a program
- System files
  - device files: character or block special – can be used to talk to hardware
  - networking endpoints: socket, pipe
- Directories
  - contain other files
- Links to other files or directories

Northern Illinois University

Directory terminology

Root Directory: `/`

- top-most directory in any UNIX file structure
- every other file must be a descendant of the root directory

Home Directory: `~`

- `~` is shorthand for current user's home directory
- `~x` is shorthand for user `x`'s home directory
- actual directory usually in `/home/username` or `/Users/username`
- directory belongs to the user whose home it is
- default location when user logs in

Current Directory: `.`

- *current working directory* – default location for working with files

Parent Directory: `..`

- directory immediately above the current directory (the one that contains it)

File and directory names

The following characters can be used in filenames:

- Uppercase letters (`A-Z`)
- Lowercase letters (`a-z`)
- Numbers (`0-9`)
- Underscore (_)
- Period/dot (.)
- Letters from alternate languages (Unicode)

Avoid the following characters:

```
& * \ | [ ] { }
$ < > ( ) # ? /
" ' ; ^ ! ~
Space Tab
```

These characters can be used, but special care must be taken when using them.

Northern Illinois University

Wildcards in filenames

The shell allows special characters in filename to specify a pattern to help with file selection.

`*`	allows a match for *any* zero or more characters at this position
`?`	matches any single character at this position
`[]`	allow any of a list (inside the `[]`) of possible characters at this position
`{}`	allow any of a list (inside the `{}`, separated by `,`) of possible words at this position

These special characters are called *wildcards*. You may also hear them referred to as *shell globbing*.

Their special meaning here is what makes them inconvenient as characters in a file's name.

Filenames

UNIX file name does not require file extension, in general

- no consideration for extension when treating files

However, some extensions can be convenient

- Program source code: `.c`, `.C`, `.h`, `.cc`, `.cpp`, `.f`, `.py`, `.js`
- Compiled object code: `.o`, `.a`, `.so`, `.sa`
- Compressed files: `.gz`, `.zip`, `.bz2`
- Archive files: `.tar`, `.tgz`
- Web site source code: `.html`, `.css`, `.php`
- Text files that will be moved to Windows: `.txt`
- Executable files typically have no extension, unlike Windows (so no `.exe` needed)

Path

Path: directions from one part of the file system to another – list of names of intermediate steps, separated by `/`

Absolute Path

- Traces a path from root directory (`/`) to a file or a directory
- Always begins with the root (`/`) directory, and will have a `/` at the beginning.

Example: `/home/student/Desktop/assign1.txt`

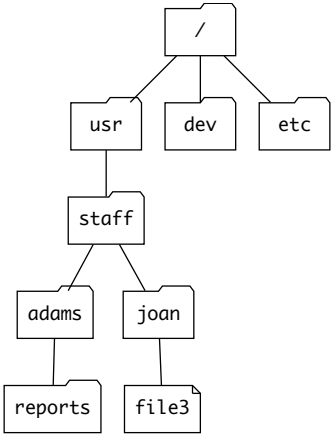
Relative Path

- Traces a path from the *current working directory*
- *No* initial forward slash (`/`)
- Dot (`.`) refers to current directory
- Two dots (`..`) refers to one level up in directory hierarchy

Example: `Desktop/assign1.txt`

Northern Illinois University

Absolute Path Example

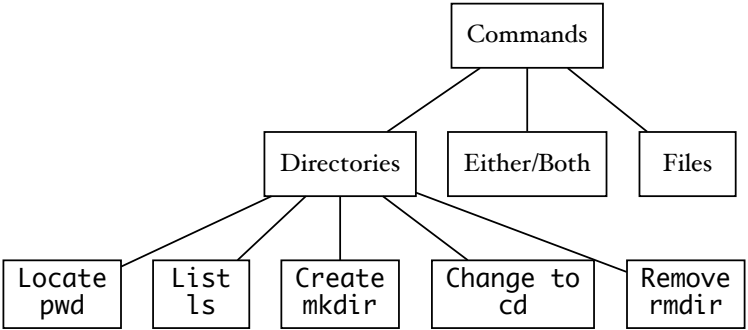


Example file system.

The absolute path to `file3` is `/usr/staff/joan/file3`

Northern Illinois University

File system commands



Tree classifying various file system commands.

Current Directory's Path

To find out the absolute path of the current working directory, use the `pwd` command.

`pwd` stands for "print working directory", and prints (to *standard output* the absolute path of the current working directory)

Example: User `student` is in their home directory, and wants to see the absolute path to it:

```
% pwd

/home/student/
```

List directory content

The most frequently used file system command: `ls`

Displays names of files present on the file system

Syntax: `ls [options] [path]`

- displays the files/directories specified as the `path` parameter
- if `path` is not supplied, lists files in current directory
- displays an error if what `path` points to is not present

Northern Illinois University

`ls` options

Common options:

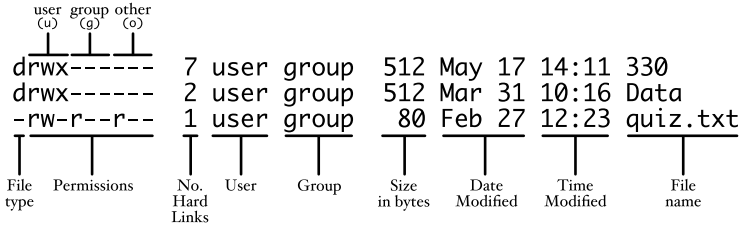
`-a`	show all files
`-l`	show long version of listing
`-t`	show files sorted by time stamp
`-S`	show files sorted by file size
`-r`	show files in reverse sorted order

Northern Illinois University

Long List Option

List contents of the current directory in long format

```
% ls -l
```



Long listing from `ls -l`.

- `.` is current dir
- `..` is parent dir
- directories have a `d` in first column
- plain files have a `-` in the first column

Northern Illinois University

Long Listing of Everything

List contents of the current directory in long format, showing hidden files.

```
% ls -al

drwxr-xr-x 13 user group 1024 Apr 26 15:49 .
drwxr-xr-x 15 root root   512 Apr 24 15:18 ..
-rwx------  1 user group 1120 Apr 12 13:11 .config
-rw-r--r--  1 user group  141 Mar 14 13:42 .logout
-rwx------  1 user group  436 Apr 12 11:59 .profile
drwx------  7 user group  512 May 17 14:11 330
drwx------  2 user group  512 Mar 31 10:16 Data
-rw-r--r--  1 user group   80 Feb 27 12:23 quiz.txt
```

- `.` is current dir
- `..` is parent dir
- directories have a `d` in first column
- names beginning with a dot (`.`) are hidden
- plain files have a `-` in the first column

Northern Illinois University

List all in a specific directory

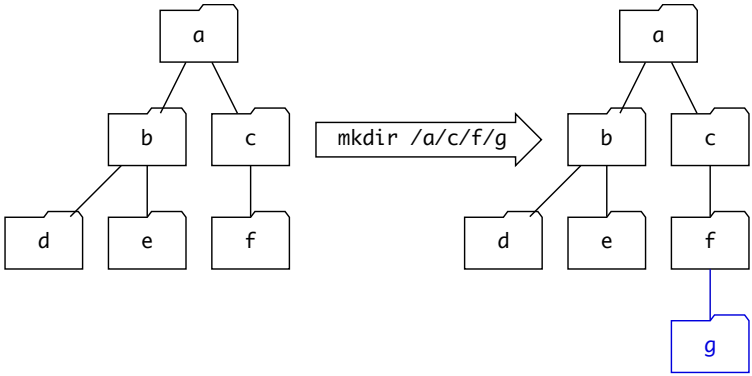
```
% ls -l 330/grades

 -rwxr-xr-x 3 user group  72 Jan 19 19:12 330assign-graderun
-rwxr-xr-x 1 user group  70 Jan 19 19:13 330exam-graderun
-rwxr-xr-x 2 user group  70 Jan 19 19:12 330quiz-graderun
-r-x------ 1 user group 468 Feb  1 11:55 test-330grade
-r-x------ 1 user group 664 Feb  1 11:55 test-330scores
```

Northern Illinois University

Creating a New Directory

Syntax: `mkdir [ -p ] directory-list`



File system before (left) and after (right) creation of the directory `g`.

Northern Illinois University

`mkdir` examples

```
% mkdir csci330

% mkdir test-data

% mkdir dirOne dirTwo

% mkdir /home/student/unix/demo

% mkdir –p /home/student/unix/demo
```

Directories in path must already exist.

Northern Illinois University

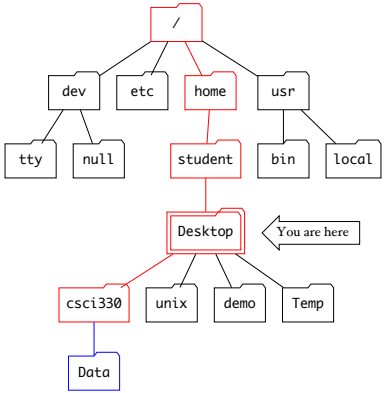
What is `mkdir -p`?

- Sometimes when using `mkdir`, you get a weird error message.

```
% mkdir a/b
mkdir: cannot create directory ‘a/b’: No such file or directory
```

- You're trying to create a directory, *of course* it doesn't exist yet. Why is that a problem?
- The answer is that, though you're trying to create `b`, the directory `a` has to be there for it to be created.
- When it is not, the command fails. This is what the `-p` option is there to solve.
- With the `-p` option supplied, all of the intermediate directories are created as well.

Example: Create a Directory



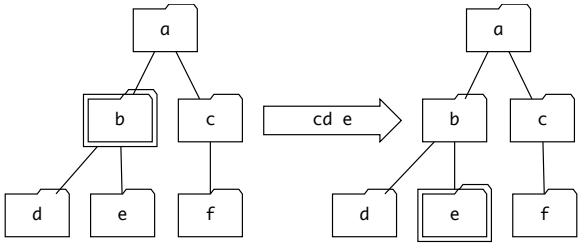
Example file system used in this slide.

To create a directory called `Data` under `csci330`:

- Using Absolute Path: `mkdir /home/student/Desktop/csci330/Data`
- Using Relative Path: `mkdir csci330/Data`

Northern Illinois University

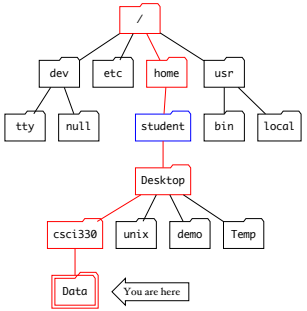
Changing Directory



Using `cd` to changing current working directory from `b` to `e`.

Northern Illinois University

Changing Directory



File system for example: change from `Data` to your home directory.

Command	How?
`cd /home/student`	absolute path
`cd ../../..`	relative path
`cd`	`cd` with no parameters goes to home directory
`cd ~`	~ expands to the home directory path

Remove Directories

If empty, use `rmdir`

Example: To remove an empty directory called `test`

```
% rmdir test
```

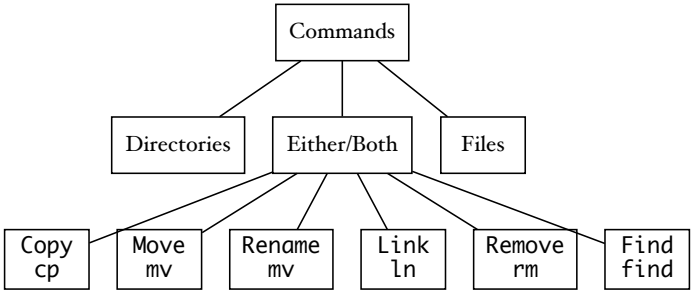
If non-empty, use `rm -r`

Example: To remove non-empty directory `old-data`,

```
% rm -r old-data
```

Northern Illinois University

File System Commands



Commands that can work with files *or* directories.

File System Commands, `cp`

Syntax: `cp source target`

`source` is one or more paths for items to copy

`target` is where to put the copy/copies:

- if only copying a single file and `target` does not exist, it is created and becomes a copy of the original
- if only copying a single file and `target` exists,
  - if `target` is a normal file, it is overwritten with the data in `source`
  - if `target` is a directory, the `source` file will be copied into that directory with the same name
- if copying multiple files, `target` *must* be a directory

Commonly used options:

`-i`	if `target` exists, the command cp prompts for confirmation before overwriting
`-R`	recursively copy entire directories
`-p`	preserve access times and permission modes

Examples: Copying a file

Make a copy of a file

```
% cp assign1.txt assign1.save
```

Copy `assign1.txt` to a different directory with name `assign1.save`

```
% cp assign1.txt  ~/archive/assign1.save
```

Copy `assign1.txt` to a different directory

```
% cp assign1.txt  ~/archive/
```

Northern Illinois University

Copying multiple files

Syntax: `cp source-files destination-directory`

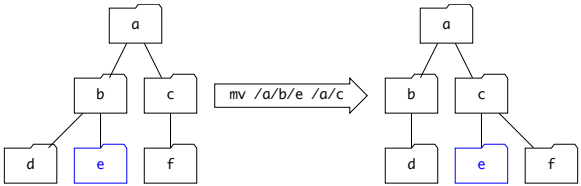
```
% cp assign1.txt assign2.txt ~/archive
% cp assign?.txt  ~/archive
```

Files will have same name in destination directory

Northern Illinois University

Moving/Renaming files/directories, `mv`

There is no special command for renaming, but you can accomplish the task with the move command, `mv`.



Moving the `e` folder from `b` to `c`.

Example: Rename file `unix` to `csci330`.

```
% mv unix csci330
```

Caveat:

What happens if `csci330` exists and is a directory?

Northern Illinois University

Moving a file

Move `assign1.txt` a different directory:

- If the destination file exists, `mv` will not overwrite existing file.

```
% mv assign1.txt  ~/archive
```

Move `assign1.txt` a different directory and rename it to `assign1.save`

```
% mv assign1.txt  ~/archive/assign1.save
```

Northern Illinois University

Moving multiple Files

Syntax: `mv source-files destination-directory`

```
% mv assign1.txt  assign2.txt  ~/archive

% mv assign?.txt  ~/archive
```

- Files will have same name in destination directory

Deleting files, `rm`

Syntax: `rm [options] path-list`

Where `path-list` is a list of paths to files that are to be removed.

Commonly used options:

`-f`	force remove regardless of permissions
`-i`	prompt for confirmation before removing
`-r`	"recursive" removes everything under the indicated directory as well

Without the `-r` flag, `rm` will emit an error when used on directories.

Example: Remove the file, `old-assign`

```
% rm unix/assign/old-assign
```

Linking Files

A *link* is a tool that allows a file (or directory) to be referenced by another name.

It is a special type of file that, when read or changed, affects another file on the system instead.

A link is:

- A reference to a file stored elsewhere on the system
- A way to establish a connection to a file to be shared

Two types:

- *Hard link* - made by `ln` without `-s` option present
- *Symbolic link* (sometimes incorrectly called a "soft link") - supply the `-s` option

Syntax: `ln [-sf] target linkname`

Northern Illinois University

The `ln` command

Hard link:

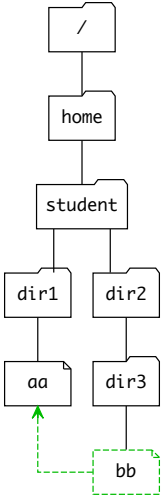
```
ln shared-file link-name
```

Symbolic link:

```
ln -s shared-file link-name
```

Northern Illinois University

Link illustration



In the above, `bb` in `dir3` is a link to file `aa` in `dir1`

File System Layout

Files on UNIX file system consist of:

- data blocks
- identified by block id
- file meta information: *inode*, containing:
  - which blocks make up file
  - permissions, etc.
- stored in *inode table*
  - index into table is *inode number*

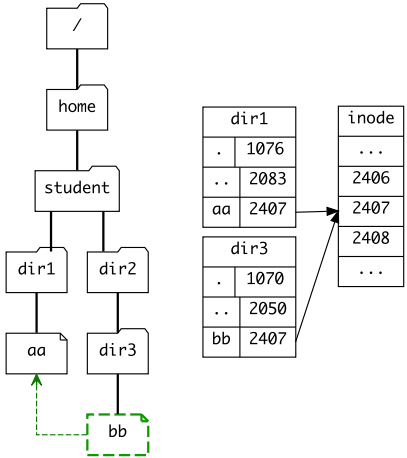
A directory is table of:

- *file name* → *inode number*

Northern Illinois University

Hard Link example: `ln`

```
% ln /home/student/dir1/aa /home/student/dir2/dir3/bb
```



Notice that a hard link to a file will share the inode from the original file.

Symbolic Link example: `ln -s`

Symbolic links do not share an inode with their target. Instead, the path to the target is stored. This allows us to make links across physical devices with different inode tables, but changes the behavior of the links as a consequence.

To see where a symbolic link points, you can use `ls -l`.

```
% ls -l bb

lrwxrwxrwx 1 user group 22 Nov 17  2018 bb ->
/home/student/dir1/aa
```

Notice the `l` in the first column, and the `->` at the end, indicating the target.

Link type comparison

Hard Link	Symbolic Link
▶ Target file must exist upon link creation, in order to know which inode. ▶ Original file will continue to exist as long as any hard link to it exists. ▶ Cannot link to a file located on a different physical device. ▶ Cannot circularly link to another hard link.	▶ Can be created even before the target file exists. ▶ Cannot access the target if it is missing or if the user doesn't have permission for the file. ▶ Can link across physical file systems. ▶ Can be circularly linked to another symbolically-linked file.

Finding Files

Syntax: `find path-list expression(s)`

`find` recursively descends through directories in `path-list` and applies the supplied expression for every file

Get details on available expressions by typing `man find`.

Northern Illinois University

The `find` command

- `path-list` specifies where to look for files

Example: `find /tmp ~`

- `expressions`
  - specify what to do with files that were found
  - can be a *test* or an *action*

Northern Illinois University

`find` expressions

- Simple tests:
  - `-name`
  - `-type`
  - `-size`
  - `-empty`
  - `-executable`

Examples:

```
% find . -name "*.txt"
% find ~ -type d
% find /tmp -empty
```

Northern Illinois University

`find` expressions

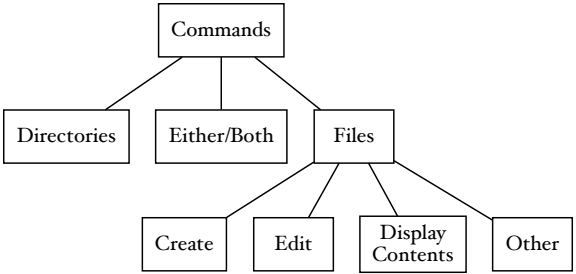
- Simple actions:
  - `-print` (default)
  - `-delete`
  - `-exec`

Examples:

```
% find . -name "*.txt"
% find ~ -name "*.txt" -exec lpr
% find /tmp -empty -delete
```

Northern Illinois University

Operations Unique to Regular Files



Things that can be done with regular files.

Creating New Files

Simplest way to create a new empty file:

`touch` command

- originally meant to update access time stamp
- side effect: if file does not exist, it will be created as an empty file

Example :

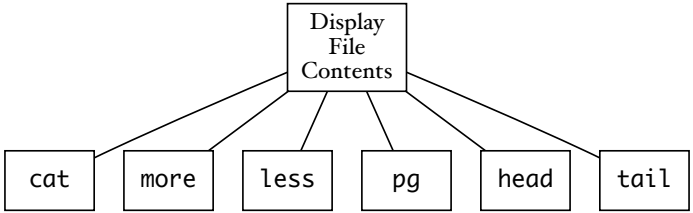
```
% touch newfile
```

Other ways to create new files:

- text editor
- redirect output from command

Northern Illinois University

Display Contents of Text Files



Simple commands that show the *contents* of files.

Viewing Contents of Text Files

The command, `cat`, can be used to display/concatenate one or more files, displaying the output all at once

Example : Display the contents of file `assign1.txt`

```
% cat assign1.txt
```

Viewing Contents of Text Files

i `more`, `less` or `pg` display the contents of one or more files, one page at a time

- Space bar – to advance to next page
- b – to go back a page
- Enter Key – to advance to next line

Example: Display the contents of file `assign1.txt`, one page at a time

```
% more assign1.txt
```

Viewing Contents of Text Files

`head` displays the beginning portion of indicated file(s); the default `head` size is 10 lines.

Example: Display first 20 lines of file `assign1.txt`

```
% head -20 assign1.txt
```

Northern Illinois University

Viewing Contents of Text Files

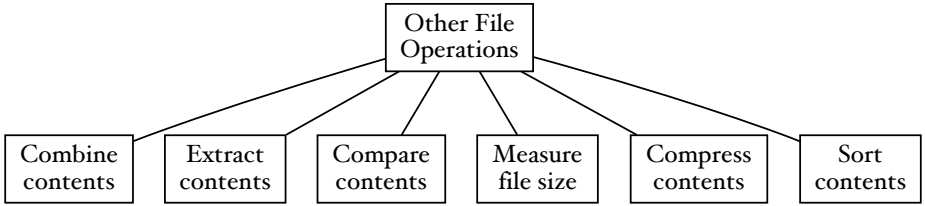
`tail` displays the ending portion of indicated file(s); the default `tail` size is 10 lines.

Example: Display last few lines of file `assign1.txt`

```
% tail assign1.txt
% tail -20 assign1.txt
```

Northern Illinois University

Operations Unique to Text Files



We will look at some of these operations on the slides that follow.

Combining Contents of Files 1

Method 1: To vertically concatenate the contents of two or more files, use the `cat` command.

Syntax: `cat file-1 file-2 file-3`

`cat` will display the combined contents of `file-1`, `file-2`, and `file-3` in top-down (vertical) fashion.

Combining Contents of Files 2

Method 2: To horizontally concatenate contents (columns/fields) of two or more files, use the `paste` command.

Syntax: `paste file-1 file-2`

`paste` will display the combined contents of `file-1` and `file-2` in side-by-side (horizontal) fashion

Extracting Part of Text

To extract one or more fields from a file, use the `cut` command.

Fields are *delimited* by special character.

- default: TAB, change via `-d` option
- common: `:`

Must specify list of fields to be extracted with the `-f` option

Example:

```
% cut -d: -f 5 /etc/passwd
```

Comparing Files: `diff`

Compare two files line by line, showing the differences.

Syntax: `diff [options] file-1 file-2`

- If `file-1` and `file-2` have the same contents, no output is produced
- If `file-1` and `file-2`'s contents are not the same, `diff` reports a series of commands that can be used to convert the first file to the second file (via the `patch` command)

Determining File Size

Recall: The `ls` command with the option `-l` gives the file size in bytes as part of its output.

Use the command named `wc` to display the size of files as number of lines, words, and characters

Syntax: `wc file-list`

common options:

`-l`	display the number of lines
`-w`	display the number of words
`-c`	display the number of characters

Compress File Contents

utilities to compress and uncompress files common on Linux:

- `gzip`, `gunzip`, `zcat`
- file extension: `.gz`

Example:

```
% gzip assign1.txt
% zcat assign1.txt.gz
% gunzip assign1.txt.gz
```

Also:

- Smaller compressed file than `gzip`: `bzip2` or `lzma`
- Windows compatible: `zip`/`unzip`, `rar`/`unrar`

Sorting Files

Syntax: `sort [options] file-name`

Commonly used options:

`-r`	sort in reverse order
`-n`	numeric sort
`-t`	field delimiter (default: blank)
`-k` x	sort based on value in field/column x
`-f`	ignore case

Sorting Files

Examples:

```
% sort fileOne
% sort -r fileOne
% sort -k 2 fileOne fileTwo
```

Northern Illinois University

User's Disk Quota

quota is upper limit of

- amount disk space
- number of files

for each user account

The command: `quota -v`

- displays the user's disk usage and limits

2 kinds of limits:

- Soft limit: ex. 100MB
  - May be exceeded temporarily
  - System will nag
- Hard limit: ex. 120MB – Cannot be exceeded