Unix for Mac OS X Users

As the lessons are being completed make noted in DETAILS section. Then summarise in the SUMMARY section. Just commands should contain no, or *very* little, explanations.

JUST COMMANDS

`echo $SHELL`

`Command k`

`cat fruit.txt`

`grep apple fruit.txt`

Switches: -i -w -v -n -c

`grep -R apple .`

Read short_file.txt and concatenate new_file.txt to it
`cat short_file.txt new_file.txt`

Read lorem_ipsum.txt using less. It outputs one screenful of text at a time.
`less lorem_ipsum.txt`
While in less, `f` or `spacebar` to to forward one page, `b` to go back one page, `g` to start of document, `G` to end of document, `q` to exit.

"Follow" a file:
`tail -f short_file.txt`
Then if any process changes the bottom of the file, e.g. a log file, it shows up immediately.

Move or rename a file or directories. Syntax:
`mv` *source destination*
`mv newfile.txt testdir/newfile.txt`
Options: `-n` no overwriting, `-f` default force overwriting, `-i` interactive overwriting "ask me", `-v` verbose

Copy
`cp` *source destination*

Delete
`rm` *source destination*

Make a directory
`mkdir` *name*

Delete an empty directory
`rmdir` *name*

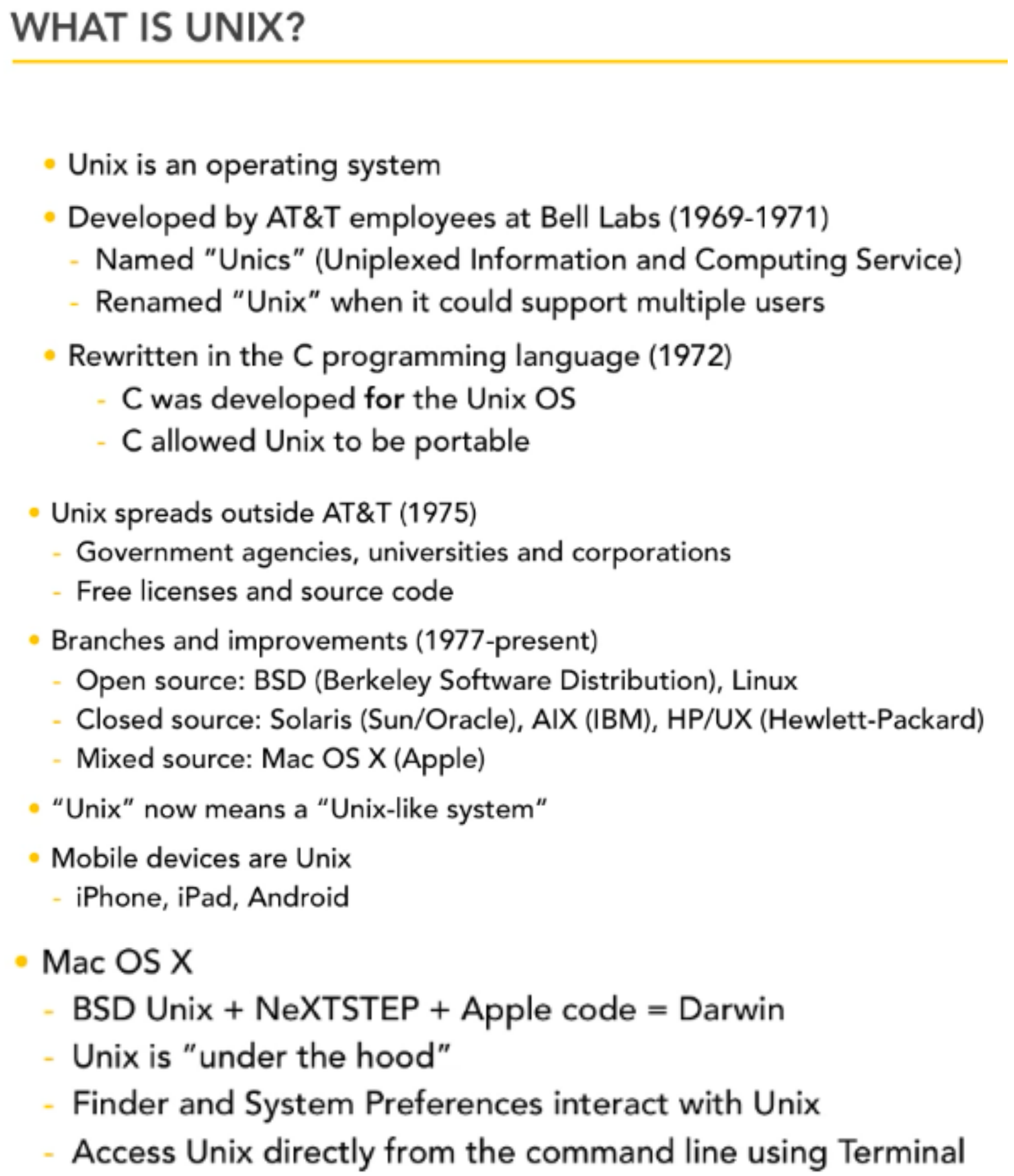
Delete a non-empty directory. You remove files and directories recursively.
`rm -R` *name*

SUMMARY

DETAILS

Chapter 1 Introduction to Unix

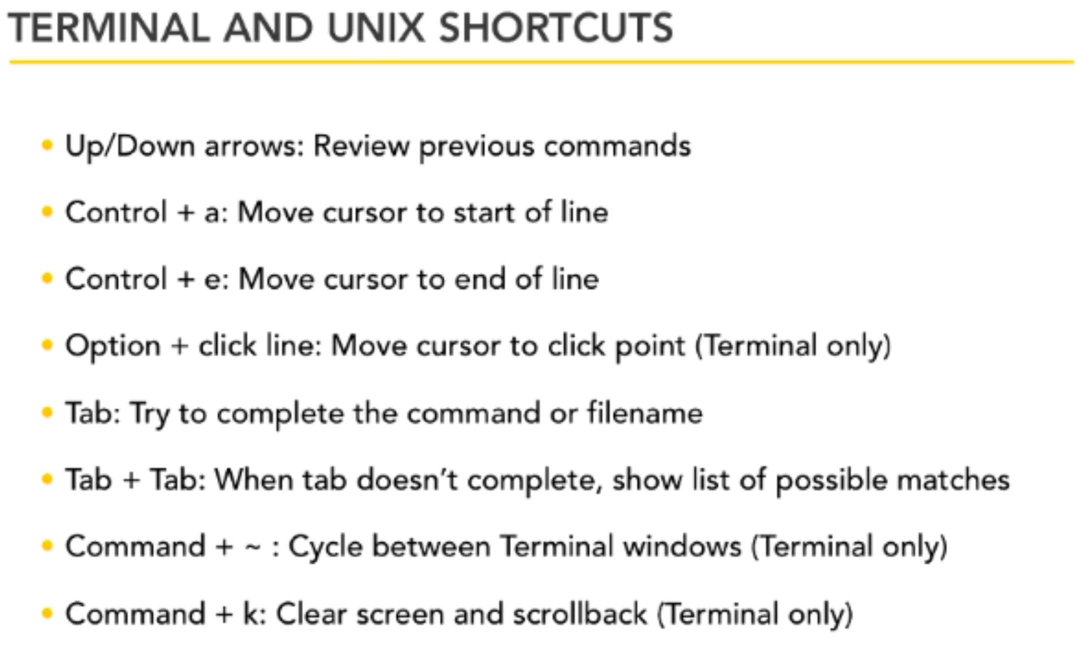
What is Unix?



WHAT IS UNIX?

- Unix is an operating system
- Developed by AT&T employees at Bell Labs (1969-1971)
  - Named "Unics" (Uniplexed Information and Computing Service)
  - Renamed "Unix" when it could support multiple users
- Rewritten in the C programming language (1972)
  - C was developed **for** the Unix OS
  - C allowed Unix to be portable
- Unix spreads outside AT&T (1975)
  - Government agencies, universities and corporations
  - Free licenses and source code
- Branches and improvements (1977-present)
  - Open source: BSD (Berkeley Software Distribution), Linux
  - Closed source: Solaris (Sun/Oracle), AIX (IBM), HP/UX (Hewlett-Packard)
  - Mixed source: Mac OS X (Apple)
- "Unix" now means a "Unix-like system"
- Mobile devices are Unix
  - iPhone, iPad, Android
- Mac OS X
  - BSD Unix + NeXTSTEP + Apple code = Darwin
  - Unix is "under the hood"
  - Finder and System Preferences interact with Unix
  - Access Unix directly from the command line using Terminal

Logging in and using the command prompt



TERMINAL AND UNIX SHORTCUTS

- Up/Down arrows: Review previous commands
- Control + a: Move cursor to start of line
- Control + e: Move cursor to end of line
- Option + click line: Move cursor to click point (Terminal only)
- Tab: Try to complete the command or filename
- Tab + Tab: When tab doesn't complete, show list of possible matches
- Command + ~ : Cycle between Terminal windows (Terminal only)
- Command + k: Clear screen and scrollback (Terminal only)

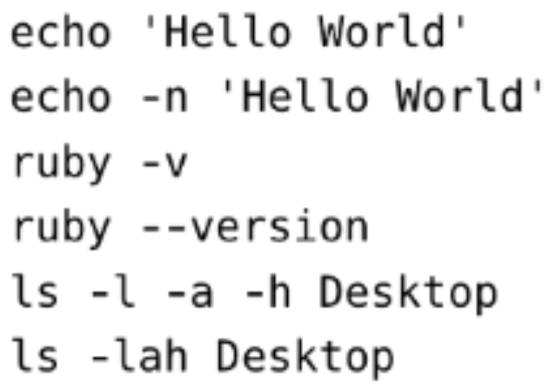
Command structure

Unix commands have the following structure

command options arguments

where *command* is always a **single** word.

Examples



```
echo 'Hello World'
echo -n 'Hello World'
ruby -v
ruby --version
ls -l -a -h Desktop
ls -lah Desktop
```

Sometimes an option needs an argument of its own. For example

```
banner -w 50 'Hello world!'
```

the 50 is an argument to the -w (width) option. So, it makes it clearer if we write

```
banner -w50 'Hello world!'
```

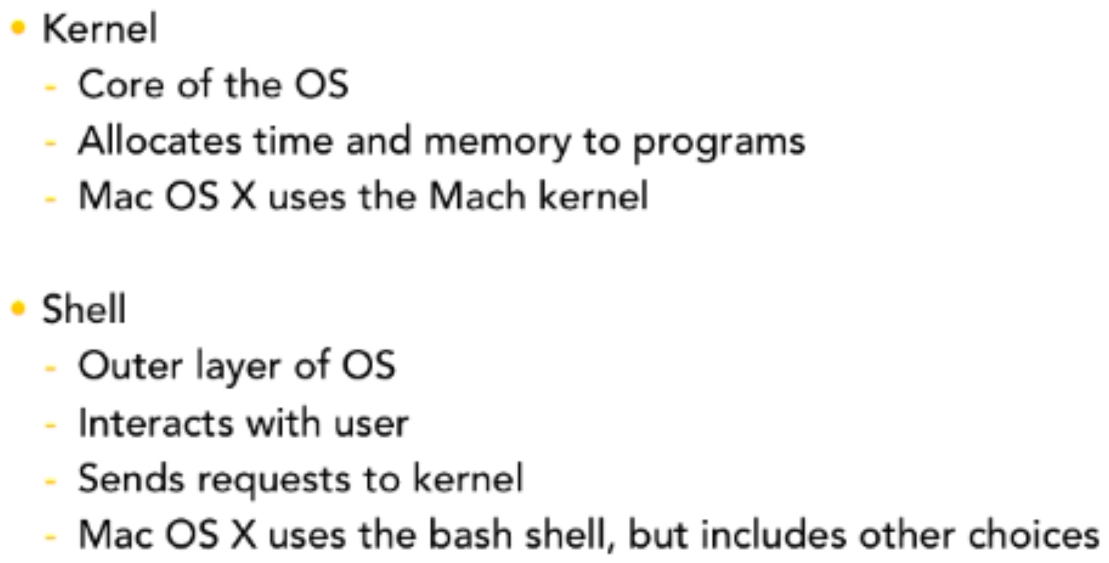
Note the space between -w and 50 has been eliminated.

We can also have multiple arguments. E.g.

```
cat -n file1.txt file2.txt
```

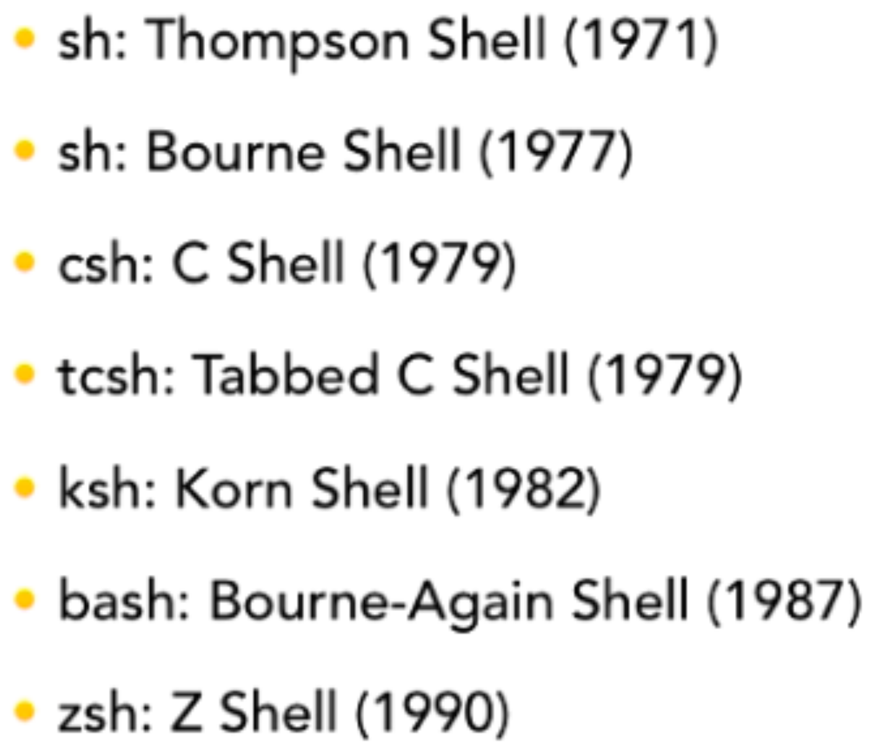
file1.txt and file2.txt are two arguments to the cat command.

Kernel and shells



- Kernel
  - Core of the OS
  - Allocates time and memory to programs
  - Mac OS X uses the Mach kernel

- Shell
  - Outer layer of OS
  - Interacts with user
  - Sends requests to kernel
  - Mac OS X uses the bash shell, but includes other choices



- sh: Thompson Shell (1971)
- sh: Bourne Shell (1977)
- csh: C Shell (1979)
- tcsh: Tabbed C Shell (1979)
- ksh: Korn Shell (1982)
- bash: Bourne-Again Shell (1987)
- zsh: Z Shell (1990)

Which shell am I working with?

```
echo $0
-bash
```

To change to another shell, type its name. For example, to change to the Tabbed c-shell, tcsh type

```
tcsh
```

Unix manual pages

Manual page command syntax:

man command

E.g.

```
man echo
```

Then

`Spacebar` or `f` forward

`b` back

`q` quit

E.g.

```
man man
```

```
man -h
```

If you know only part of the name of a command, use the -k option. E.g.

```
apropos ban
```

Will show three commands all of which contain 'ban'

2. Filesystem Basics

The working directory

```
pwd
```

Listing files and directories

```
ls -lah
```

l: different format
a: all, including hidden 'dotfiles'
h: human readable size (appends B for bytes; K for kilobytes; etc)

Moving around the filesystem

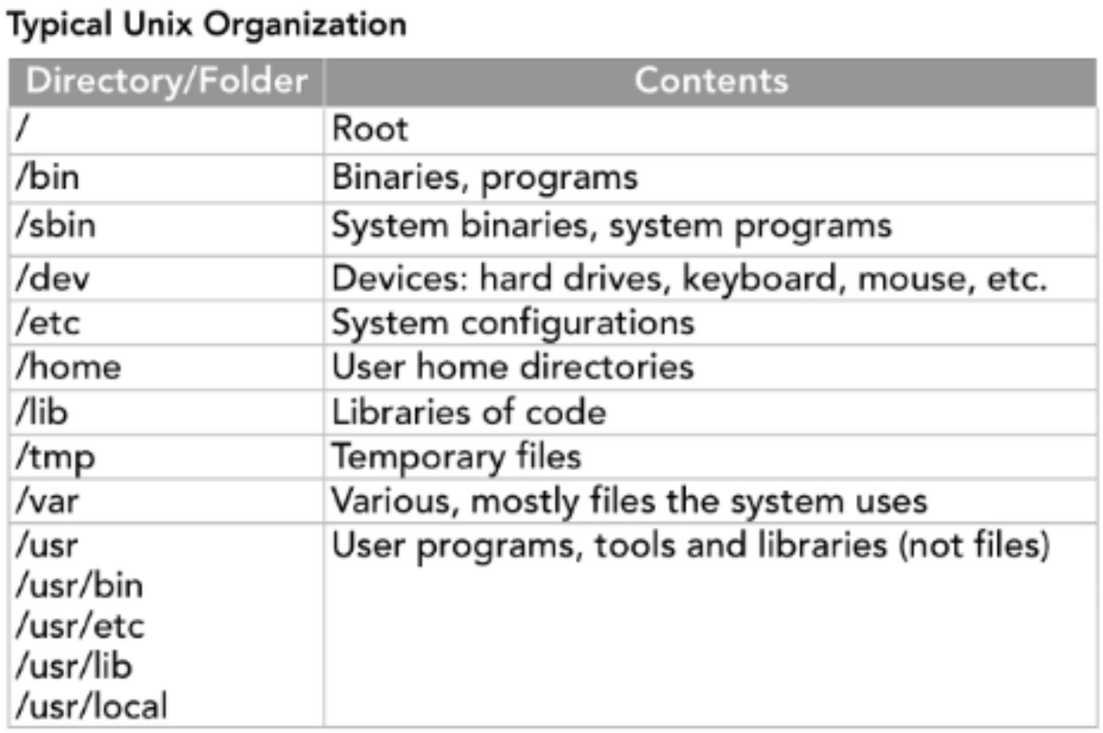
`cd L`<Tab> for auto-complete. Will complete if, for example `L`ibrary is a subdirectory.

`cd L`<Tab><Tab> Will give a list of possible options.

`~` Current user directory

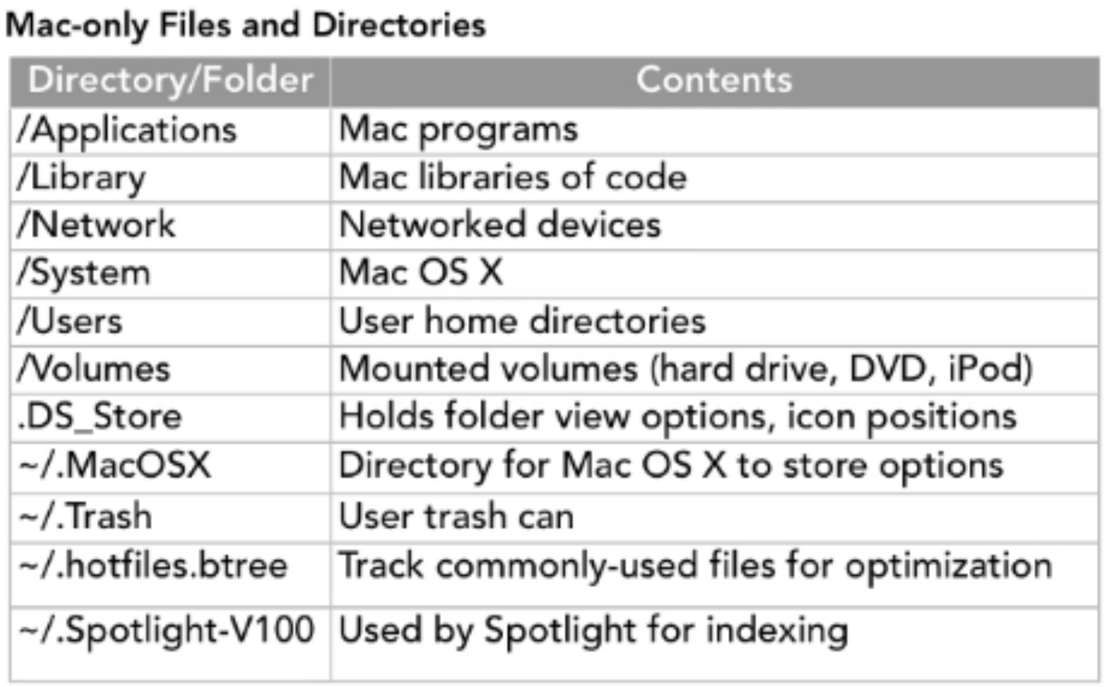
`-` Previous directory

Filesystem organization



Typical Unix Organization

Directory/Folder	Contents
/	Root
/bin	Binaries, programs
/sbin	System binaries, system programs
/dev	Devices: hard drives, keyboard, mouse, etc.
/etc	System configurations
/home	User home directories
/lib	Libraries of code
/tmp	Temporary files
/var	Various, mostly files the system uses
/usr /usr/bin /usr/etc /usr/lib /usr/local	User programs, tools and libraries (not files)



Mac-only Files and Directories

Directory/Folder	Contents
/Applications	Mac programs
/Library	Mac libraries of code
/Network	Networked devices
/System	Mac OS X
/Users	User home directories
/Volumes	Mounted volumes (hard drive, DVD, iPod)
.DS_Store	Holds folder view options, icon positions
~/.MacOSX	Directory for Mac OS X to store options
~/.Trash	User trash can
~/.hotfiles.btree	Track commonly-used files for optimization
~/.Spotlight-V100	Used by Spotlight for indexing

Chapter 3 Working with Files and Directories

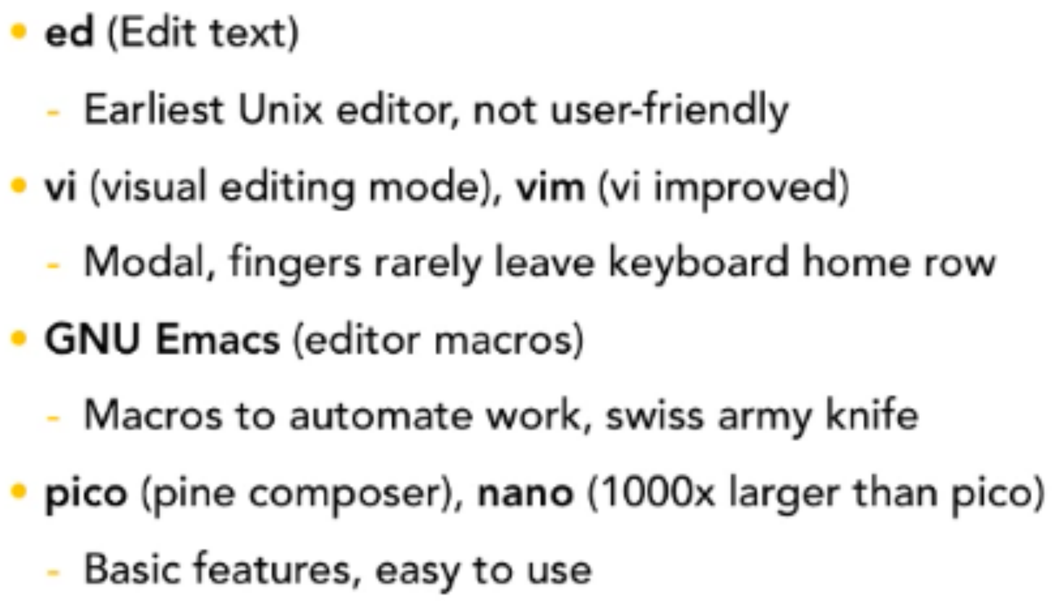
Naming files

- Maximum of 255 characters
- Avoid / \ * & % ? $ | ^ ~ < > and most other symbols
- Use A-Z, a-z, 0-9, period, underscore, hyphen
- Typically lowercase
  - "MyFile" and "myfile" would be different
- Underscores are better than spaces
  - Escape spaces with \
  - Use quotes around names with spaces
- File endings (.txt, .png, .html, etc.) not required but helpful
  - Differentiates files from commands and directories

Creating files

- Primary techniques to create files
  - Unix text editors
  - Direct output to file
  - `touch`

Unix text editors



- **ed** (Edit text)
  - Earliest Unix editor, not user-friendly
- **vi** (visual editing mode), **vim** (vi improved)
  - Modal, fingers rarely leave keyboard home row
- **GNU Emacs** (editor macros)
  - Macros to automate work, swiss army knife
- **pico** (pine composer), **nano** (1000x larger than pico)
  - Basic features, easy to use

Use nano as it is user friendly. Three ways:

`nano` Just start nano

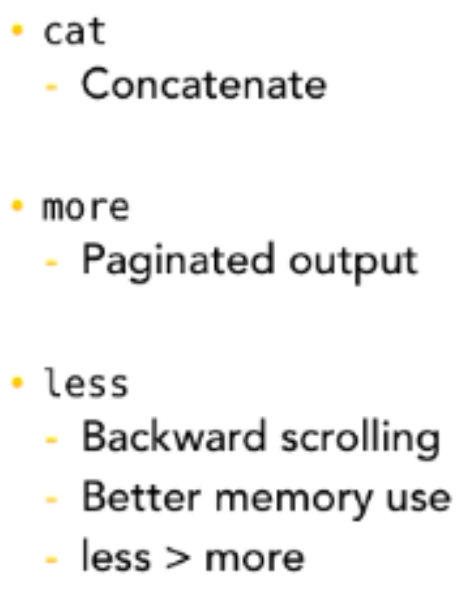
`nano fruit.txt` Start editing fruit.txt (assuming it extists)

`nano fruit1.txt` Start editing a blank file, and at the end prompt me for the name fruit.txt. Hit ***Return*** when prompted.

Some features of nano

^w Where Is: Same as find
^Y Previous Page
^V Next Page
^K
^U
Fro more: nano-editor.org

Reading files



- `cat`
  - Concatenate
- `more`
  - Paginated output
- `less`
  - Backward scrolling
  - Better memory use
  - less > more

In mac less has *replaced* more.
man pages use less. So the same commands (f or spacebar; b; q) we use for man pages work in less.
Also

`g` go to start of the document

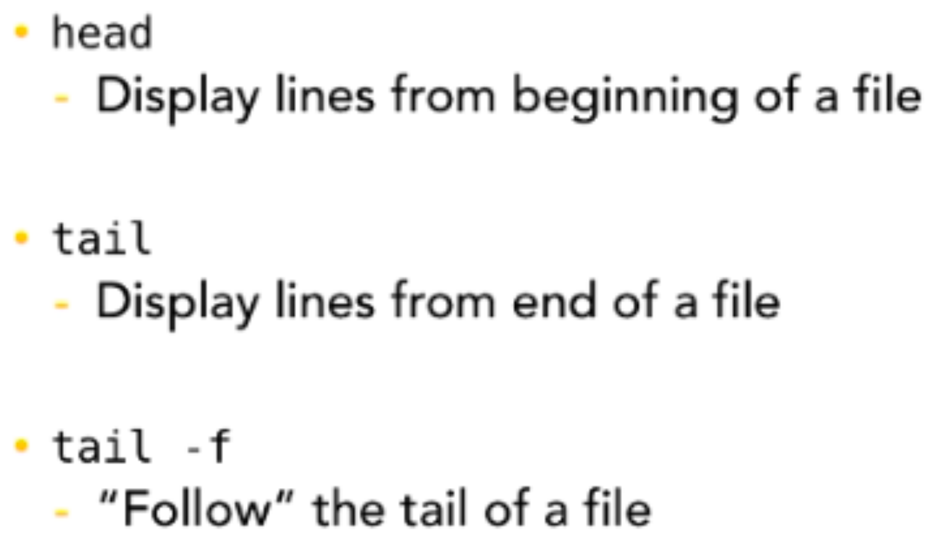
`G` go to the end of the document

`-M` option provides more info (lines 1–23/523, for example)

```
less -M lorem_ipsum.txt
```
…
```
Curabitur vestibulum aliquam purus eget venenatis.
Phasellus scelerisque, telluslorem_ipsum.txt lines
1-23/523 4%
```

`-N` option adds line numbers

Reading portions of files



- head
  - Display lines from beginning of a file
- tail
  - Display lines from end of a file
- tail -f
  - "Follow" the tail of a file

`head lorem_ipsum.txt`

`tail lorem_ipsum.txt`

`tail -f lorem_ipsum.txt` follow the file i.e. update the file in real time

`^c` to get out of tail -f

Sumamry: Use
cat for small files
less for large files
head or tail to look at beginning or end of a file

Creating directories

`mkdir newdir` will create a directory called newdir

`mkdir -p testdir/test1/test2` the `-p` option says create any parent directories as needed

`-v` the verbose option will give feedback to say what directories were created

Moving and renaming files and directories

mv *fileToBeMoved pathToMoveTo*

We move directories in the same way that we move files.
We also use mv to rename files and directories

mv options

`-n` no overwriting

`-f` force overwriting defaule

`-i` interative overwriting, ask me

`-v` verbose

Copying files and directories

cp *source destination*

`cp a.txt b.txt` will make a copy of a.txt and call the copy b.txt

cp options

Are the same as options for mv

cp will not copy directories unless the -R (recursive) option is passed

`cp -R testdir myfiles/testdirCopy` copy testdir directory and all its subdirectories recursively to myfiles and call the copy testdirCopy

Deleting files and directories

rm *filename*

`rm somrfile.txt` remove a file

To delete a directory we have two choices: use `rmdir` and `rm -R`

`rmdir dir1` Works for only empty directories `rm -R dirBig` Will delete dirBig and recursively everything in it

Finder aliases in Unix

Are useful to Finder, but useless to UNIX itself. (Aliases that Finder creates for bot files and directories are files containing information in a format understandable to only Finder.)

Hard Links

This created a hard link called myHardLink to a file called myFile.txt

```
ln myFile.txt myHardLink
```

When you create a hard link, you create another file ***name*** that points to the same **file** (set of bytes on the HDD).

Consequences:

- Open myHardLink and change it. Then myFile.txt will be changed. And vice versa.
- Delete one or the other of myFile.txt or myHardLink, and the file will not be deleted, although the name you deleted will be. So delete myFile.txt. The name will disappear, but the file is still on disk and can be accessed via myHardLink.
- If you move the name myHardLink to another directory, it will still work fine. Also, if you move the name myFile.txt to another directory, it will still work fine.

Symbolic Links

Same syntax as hard links, but use the `-s` option

```
ln -s myFile.txt mySymLink
```

When you create a symbolic link called mySymLink to a file called myFile.txt, you create not just a file name, but also another ***file***. The new file contains the path of myFile.txt. So, for example, in the above example the file called mySymlink will contain the literal string "myFile.txt" as the symlink and target file are in the same directory.

Consequences:

- If the original file is moved or deleted, the link breaks.

Finder aliases

Are a third breed of the animal. They are a sophisticated symlink invented by apple that tries to keep track when the target filename is moved.

Searching for files and directories

We use `find` to search for files.

Syntax: `find` *path expression*

path: *where* to look
expression: *what* to look for

Find in `~/Documents` any file that has name `someimage.jpg`

```
Find ~/Documents -name "someimage.jpg"
```

Note: `-name "someimage.jpg"` is the expression—the hyphen in front of 'name' gives the impression it is an option, but it is not.

Wildcard characters:

`*`
zero or more characters (called the 'glob')
`?`
any one character
`[a-c]`
a or b or c

find in current directory all files whose name is *.plist and whose full path does not contain *QuickTime* (*anythig***QuickTime***anything*)

```
find . -name *.plist -and -not -path *QuickTime*
```

We can use `-and`, `-or`, `-not`, it the usual sense. Can be used repeatedly. For example

```
find . -name *.plist -and -not -path *QuickTime* -and
-not -path *Preferences*
```

will also drop any files whose full paths contains Preferences.

Find all files of type file with size less than 41k case-insensitive name *.pdf

```
find . -type f -size -41k -iname '*.pdf'
```

Find all files of type file with size less than 41k case-insensitive name *.pdf and move them to the directory ./small

Notes:

{} is substituted with file names to read, for example mv file1 ./small

\; is necessary to terminate the many lines that this command gets translate to

```
find . -type f -size -41k -iname '*.pdf' -exec mv {} ./
small \;
```

4. Ownership and Permissions

Who am I?

```
whoami
kamranlayegh
```

```
echo $HOME
/Users/kamranlayegh
```

Unix groups

A group is a set of users. Every user must belong to at least one group.

To find out which groups you belong to, usr the groups command

```
groups
staff com.apple.sharepoint.group.1
```

...

The above is the groups that Apple have created for you.

File and directory ownership

You can see the ownership of files and directories you do ls -la. It's the second and third columns that you see.

```
    ls -la
    drwxr-xr-x  10 kamranlayegh  staff     340 10 May
19:45 .
```

OWNER (USER) GROUP
kamranlayeg staff

To change ownership use the chown command

For example, suppose the file test.txt belongs to kamranlayegh:

```
    ls -la
    -rw-r--r--   1 kamranlayegh  staff     0 10 May
19:58 test.txt
```

To change ownership to user `Kam2` in group `staff`, we use the `chown` command:

```
    chown Kam2:staff test.txt
    chown: test.txt: Operation not permitted

    sudo chown Kam2:staff test.txt
    ...
```

To prove that ownership has changed:

```
    ls -la
    ...
    -rw-r--r--   1 Kam2          staff     0 10 May
19:58 test.txt
```

To change just the owner to Kam2

```
    chown Kam2 test.txt
```

To change just the group to staff

```
    chown :staff test.txt
```

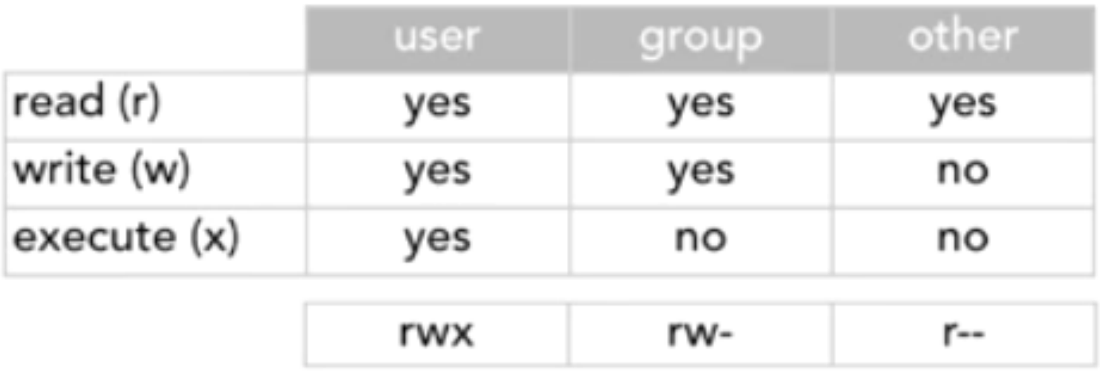
To change the ownership of a directory without changing the ownership of the contents of it

```
chown kamranlayegh:staff test1
```

To change the ownership of a directory and the contents of the directory use the recursive option, -R

```
chown -R kamranlayegh:staff test1
```

File and directory permissions



	user	group	other
read (r)	yes	yes	yes
write (w)	yes	yes	no
execute (x)	yes	no	no
	rwx	rw-	r--

execute permission on a **directory** means being able to **search** in it.

Setting permissions using alpha notation

u means user (owner)
g means group
o means others

Set permissions for owner and group and others

```
chmod ugo=rwx test.txt
```

Set permissions selectively

```
chmod u=rwx,g=rw,o=r test.txt
```

Add write permission to owner and group

`chmod ug+w test.txt`

Take away write permission from others

`chmod o-w test.txt`

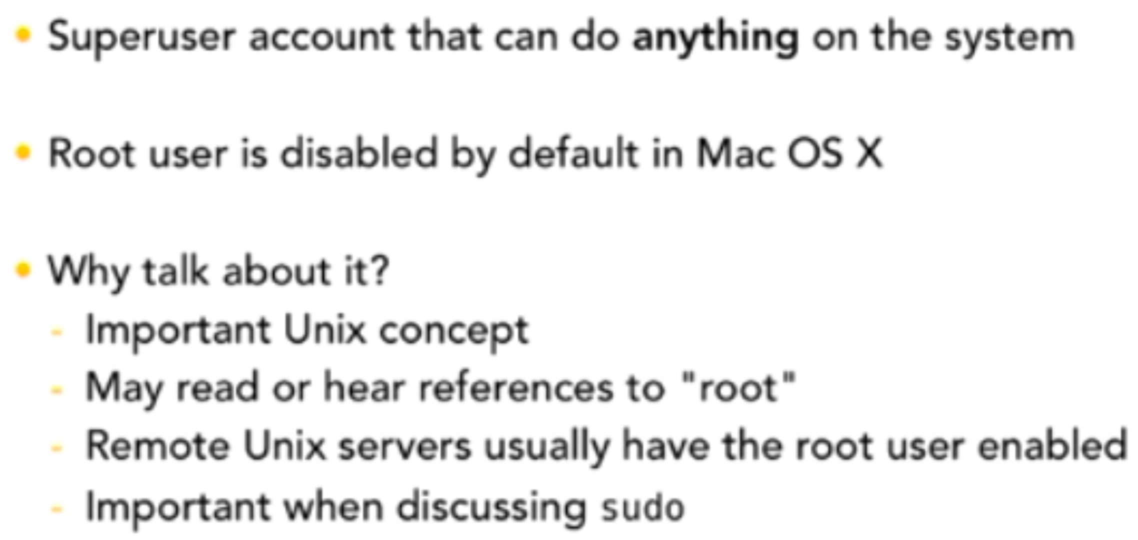
We can use the shorthand a for all (ugo)

`chmod a+rw test.txt` add r and w permissions to ugo

is the same as

`chmod ugo+rw test.txt` add r and w permissions to ugo

The root user



- Superuser account that can do **anything** on the system
- Root user is disabled by default in Mac OS X
- Why talk about it?
  - Important Unix concept
  - May read or hear references to "root"
  - Remote Unix servers usually have the root user enabled
  - Important when discussing sudo

From Apple

- Any user with an administrator account can become the root user or reset the root password.
- A root user has the ability to access other users' files on the computer.
- The root user doesn't appear in Users & Groups, Users, or Accounts preferences.

Is a root user enabled?

It looks like there is no way of checking either using Apple GUI, or the command line. But what is the point in checking?

https://support.apple.com/en-gb/HT204012

From OSXDaily

Enable a root User Account In Yosemite, OS X Mavericks and Mountain Lion

```
% dsenableroot
username = Paul
user password:
root password:
verify root password:
dsenableroot:: ***Successfully enabled root user.
```

Disable Root User Account

```
% dsenableroot -d
username = Paul
user password:
dsenableroot:: ***Successfully disabled root user.
```

http://osxdaily.com/2015/02/19/enable-disable-root-command-line-mac/

sudo and sudoers

As admin users on the Mac we can do everything that the root user can do. Use sudo, say 'sue-do'.

sudo stands for 'substitute user and do' not 'superuser and do'.

You prefix a command with sudo:

```
sudo ls -la
```

When you do a sudo and authenticate successfully, it remains valid for about 5 minutes. You can kill (expire) the authentication immediately by using

```
sudo -k
```

If you want to run a command not as root, but as another user, use the -u option

```
sudo -u Kam2 whoami
Password:
Kam2
whoami
kamranlayegh
```

But

```
sudo whoami
root
```

Unix keeps track of who is allowed to do a sudo in a special file

```
sudo cat /etc/sudoers
...
```

Do not edit this file directly. Use Apple GUI "Users & Groups" in System Preferences.

5. Commands and Programs

Command basics

The PATH variable

System information commands

Disk information commands

Viewing processes

Monitoring processes

Stopping processes

Text file helpers

Utility programs

Using the command history

Chapter 6 Redirecting Input and Output

Piping output to input

```
echo "Hello World" | wc
```

produces

```
 1       2      12
```

(1 line, 2 words, 12 chars)

Chapter 7 Configuring Your Work Environment

Profile, login, and resource files

When bash starts, it reads, in order:

```
/etc/profile
~/.bash_profile, ~/.bash_login, ~/.profile, ~/.login
```

`/etc/profile` is for all users and should not be edited. This file is always read.

Then, the file names on the second line above are checked. As soon as one of these is found to exist it will be read and the search terminated. That means the remaining files on the list will be ignored.

The file names on the second line above are for the current user and can be modified.

Kevin Scoglund recommends that we put all our configs in `~/.bash_profile`

In order to get our customisation take effect for both cases when we
- open a new terminal window and
- open a sub-shell by typing 'bash'

Kevin Scoglund recomends we put all our configs in `~/.bashrc` (for bash resource) and in `~/.bash_profile` tell bash to read and execute `~/.bashrc`

KL followed this, 24.04.2016, to add colour to grep matches. See "Steps" below.

How?

Add the following to `~/.bash_profile`:

```
if [-f ~/.bashrc]; then
    source ~/.bashrc
fi
```

Then put all shell config in `~/.bashrc`

Note: the conditional simply says if `-f` (file exists) `~/.bashrc` then read it as your source for config.

"Steps"

```
cd "/Users/kamranlayegh"
touch .bashrc
```

```
nano .bash_profile
```

Then typed in:

```
if [-f ~/.bashrc]; then
    source ~/.bashrc
fi
```

Then Control X, y, return.

Then in `~/.bashrc` added:

```
export GREP_OPTIONS="--color=auto"
```

Cleanup

~/.bash_logout is read and executed when you close a terminal window.

Chapter 8 Unix Power Tools

Find a text inside a given file

23.04.2016

Quickly clear screen and lose output history

```
Command k
```

Quick look inside a text file?

```
cat fruit.txt
```

"Grep for apple inside fruit.txt" meaning: Search for and print... (**G**lobal **r**egular **e**xpression **p**rint)

```
grep apple fruit.txt
```

Case insensitive -i:

```
grep -i apple fruit.txt
```

Match only whole words -w, i.e. do not match pineapple

```
grep -w apple fruit.txt
```

Inversion option -v, i.e. match the lines that do not have apple in them. Will match many lines.

```
grep -v apple fruit.txt
```

Line number option -n, gives line numbers in front of the matching lines

```
grep -n apple fruit.txt
```

The count option -c, returns just a count integer

```
grep -c apple fruit.txt
```

Find 'apple' in any file whose filename ends with fruit.txt

```
grep apple *fruit.txt
```

Find 'apple' in any file whose filename ends with fruit.txt, but show me only the filenems

```
grep -l apple *fruit.txt
```

Find a text inside any of the files in a given directory
Pipe other output into grep

The -R (Recursive) option is necessary.

```
grep -R apple .
```

The `-h` option hides the path and the file name

The `-l` option shows only the file paths (not the lines of text containing the matches).

The `-L` option shows the file paths for which there is no match.

grep for apple inside any files in the current directory that ends in fruit.txt

```
grep apple *fruit.txt
```

We can pipe the result of a cat into grep

```
cat fruit.txt | grep apple
```

The above is a little pointless. However, we can pipe anything, for example a list of all running processes into grep:

```
ps aux | grep Terminal
```

We can pipe the history (of the commands in this session) to grep:

```
history | grep ls
```

8.2 grep- Multiple files, other input

Add colour to output for easy recognition of matches

If we just do a `grep lorem lorem_ipsum.txt`, we get a lot of output with the word lorem buried in there which makes it hard to find the occurrences of the word. Adding the --color option makes the word lorem appear coloured

```
grep --color lorem lorem_ipsum.txt
```

To get matches in grep show up coloured automatically,
put these lines in `.bashrc`

```
export GREP_COLOR="34;47"
export GREP_OPTIONS="--color=auto"
```

Then make bash use `.bashrc` by putting these lines in `.bash_profile`

```
if [ -f ~/.bashrc ]; then
  source ~/.bashrc
fi
```

OTHER COLOURS

foreground colors		background colors		attributes	
30	black	40	black	0	normal display
31	red	41	red	1	bold
32	green	42	green	4	underline (mono only)
33	yellow	43	yellow	5	blink on
34	blue	44	blue	7	reverse video on
35	magenta	45	magenta	8	nondisplayed (invisible)
36	cyan	46	cyan		
37	white	47	white		

8.3 grep- Coloring matched text

Using regular expressions

It is a good habit to put single quotes around regular expressions.

```
grep '.a.a.a' fruit.txt
banana
papaya
```

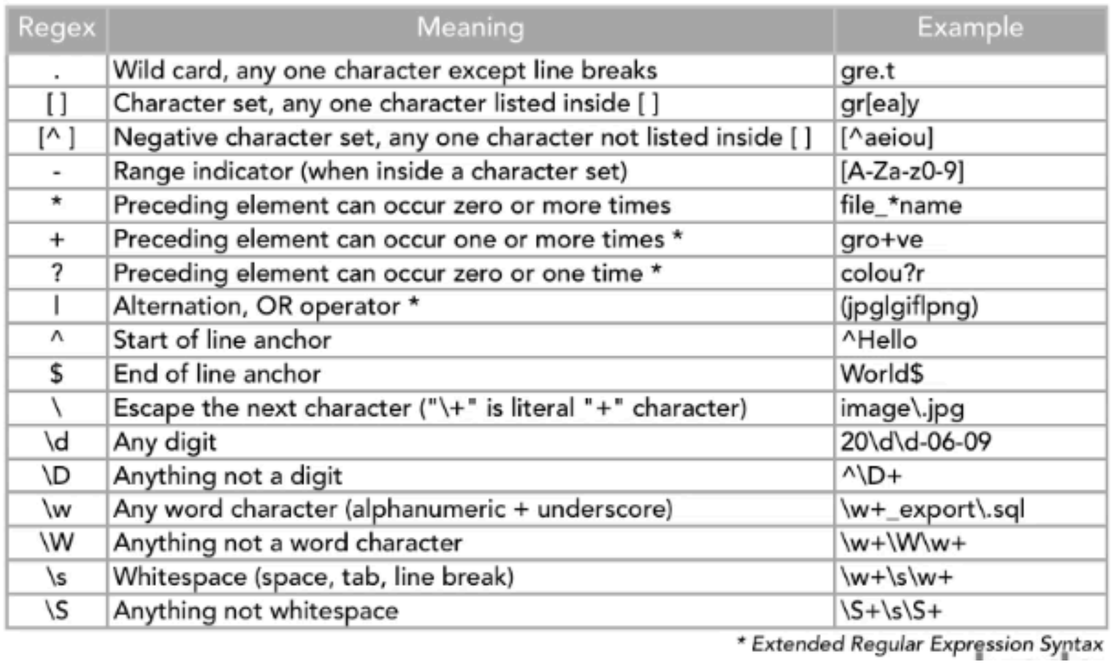
as . matches exactly one character.

```
grep 'a..le' fruit.txt
apple
pineapple
apple
```

```
grep 'ea[cp]' fruit.txt
peach
pineapple
```

8.4 Introduction to regular expressions

Basic Syntax



Regex	Meaning	Example
.	Wild card, any one character except line breaks	gre.t
[ ]	Character set, any one character listed inside [ ]	gr[ea]y
[^]	Negative character set, any one character not listed inside [ ]	[^aeiou]
-	Range indicator (when inside a character set)	[A-Za-z0-9]
*	Preceding element can occur zero or more times	file_*name
+	Preceding element can occur one or more times *	gro+ve
?	Preceding element can occur zero or one time *	colou?r
|	Alternation, OR operator *	(jpg|gif|png)
^	Start of line anchor	^Hello
$	End of line anchor	World$
\	Escape the next character ("\+" is literal "+" character)	image\.jpg
\d	Any digit	20\d\d-06-09
\D	Anything not a digit	^\D+
\w	Any word character (alphanumeric + underscore)	\w+_export\.sql
\W	Anything not a word character	\w+\W\w+
\s	Whitespace (space, tab, line break)	\w+\s\w+
\S	Anything not whitespace	\S+\s\S+

** Extended Regular Expression Syntax*

Regular Expression Character classes

Class	Represents
[:alpha:]	Alphabetic characters
[:digit:]	Numeric characters
[:alnum:]	Alphanumeric characters
[:lower:]	Lower-case alphabetic characters
[:upper:]	Upper-case alphabetic characters
[:punct:]	Punctuation characters
[:space:]	Space characters (space, tab, new line)
[:blank:]	Whitespace characters
[:print:]	Printable characters, including space
[:graph:]	Printable characters, not including space
[:cntrl:]	Control characters (non-printing)
[:xdigit:]	Hexadecimal characters (0-9, A-F, a-f)

8.5 Regular expressions- Basic syntax

Using Character Classes

Notice that the above character classes by themselves will be interpreted as character sets. The following will match : u p e r

```
    echo 'Aa:BbDcEeUu' | grep '[:upper:]'
Aa:BbDcEeUu
```

To get the desired effect, enclose in another set of square brackets

```
    echo 'AaBbDcEeUu' | grep '[[:upper:]]'
AaBbDcEeUu
```

```
    grep [[:alpha:]] fruit.txt
pear
raspberry
...
```

Note:

1.

```
    grep 'ap+le' fruit.txt
```

[No matches], but

2.

```
    grep -E 'ap+le' fruit.txt
apple
pineapple
apple
```

3.

```
grep 'ap\+le' fruit.txt
```

```
apple
pineapple
apple
```

Why?
In 1, the + is interpreted literally.
In 2, by adding -E option, we turn on the *extended* regex. See the Basic Syntax table above.
In 3, we escape the + so it is not interpreted as a literal + sign, but as meaning "1 or more repeats of the previous character".

8.6 Using regular expressions with grep

tr

Can search and replace for single characters. For example, search for ',' and replace them with '-'

```
echo 'a,b,c' | tr ',' '-'
```

```
a-b-c
```

Replace every 1 with A and every 2 with a B

```
echo '1121' | tr '12' 'AB'
```

```
AABA
```

We can also use character sets

```
echo '221213'|tr '1-3' 'a-c'
```

```
bbabac
```

For more, including -s (to suppress a sequence of repeated character to one occurrence), -d (to delete), -c (to get the complement of, i.e. not— e.g. tr -cd '[:alnum:]' to dlelete all non-alpha-numeric characters) see
https://en.wikipedia.org/wiki/Tr_(Unix)

We can use character classes

```
tr '[:upper:]' '[:lower:]' < people.txt
```

```
kevin
lynda
bob
...
```

Note that all upper case characters have been turned into lower case.

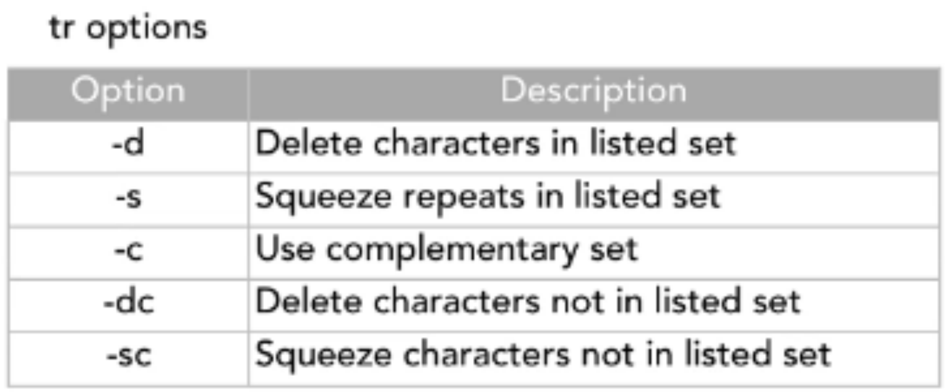
The following will read a set of comma-separated-values from us_presidents.csv, then replace all the commas with tabs, then output the result to a new file called us_presidents.tsv

```
tr ',' '\t' < us_presidents.csv > us_presidents.tsv
```

8.7 tr- Translating characters

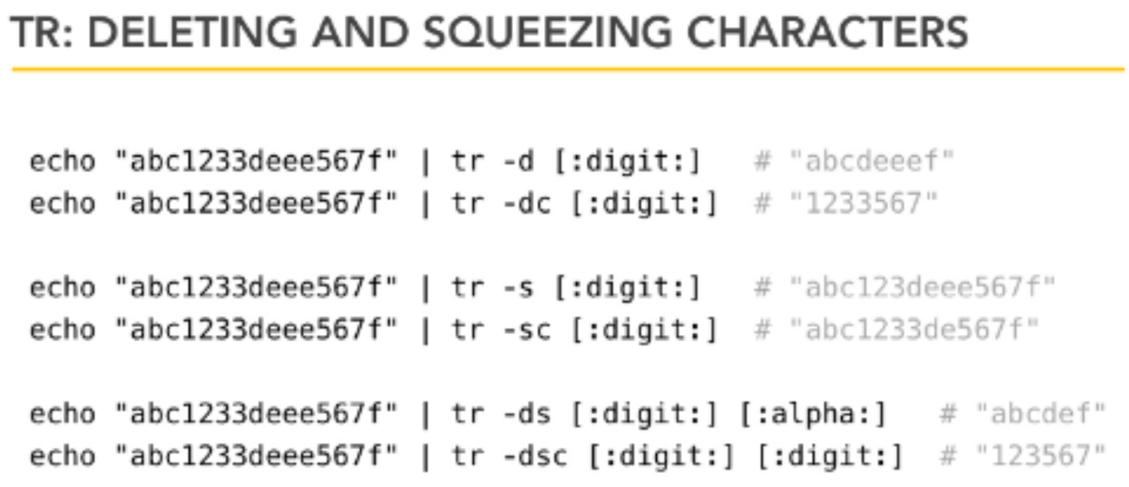
Deleting and 'squeezing with tr

-c (complement) is the negation operator and is used in combination with -d option. See examples in the next table.



tr options

Option	Description
-d	Delete characters in listed set
-s	Squeeze repeats in listed set
-c	Use complementary set
-dc	Delete characters not in listed set
-sc	Squeeze characters not in listed set

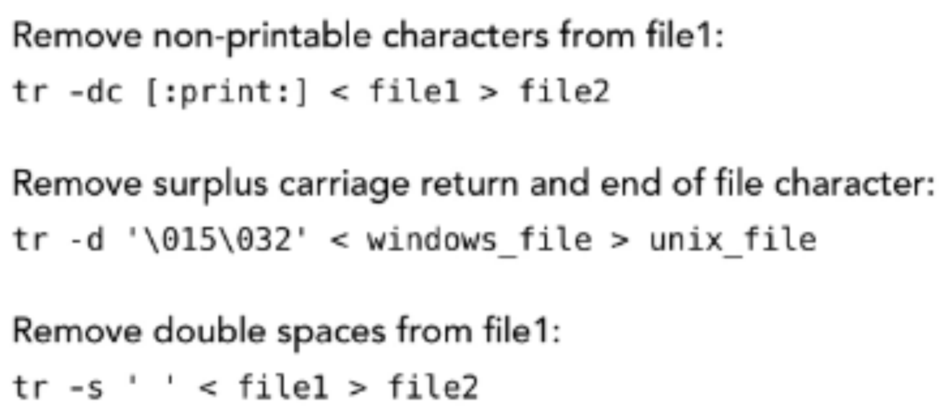


TR: DELETING AND SQUEEZING CHARACTERS

```
echo "abc1233deee567f" | tr -d [:digit:]    # "abcdeeef"
echo "abc1233deee567f" | tr -dc [:digit:]   # "1233567"

echo "abc1233deee567f" | tr -s [:digit:]    # "abc123deee567f"
echo "abc1233deee567f" | tr -sc [:digit:]   # "abc1233de567f"

echo "abc1233deee567f" | tr -ds [:digit:] [:alpha:]    # "abcdef"
echo "abc1233deee567f" | tr -dsc [:digit:] [:digit:]   # "123567"
```


Remove non-printable characters from file1:

```
tr -dc [:print:] < file1 > file2
```

Remove surplus carriage return and end of file character:

```
tr -d '\015\032' < windows_file > unix_file
```

Remove double spaces from file1:

```
tr -s ' ' < file1 > file2
```

8.8 tr- Deleting and squeezing characters

sed the Stream Editor

sed is more powerful than tr.

The syntax for search and replacement is

```
sed 's/search_string/replacement_string/'
```

```
echo 'upstream' | sed 's/up/down/'
downstream
```

By default, sed replaces only the first match

```
echo 'upstream and upward' | sed 's/up/down/'
downstream and upward
```

To replace all the occurrences, add a g (global) to the end of the expression passed to sed

```
echo 'upstream and upward' | sed 's/up/down/g'
downstream and downward
```

The delimiters (forward slash in the above examples) can be replaced with other characters. sed looks at the first character that comes after the first s (for substitute) and uses it as the delimiter.

```
echo 'upstream' | sed 's:up:down:'
downstream
```

To read from a file as input stream, simply pass the file as a second parameter after the expression

```
sed 's/pear/golabi/' fruit.txt
```

works the same as

```
sed 's/pear/golabi/' < fruit.txt
```

To output to a file, you must pipe

```
sed 's/pear/golabi/' fruit.txt > golabi.txt
```

Note: Each line is treated as a stream. This means that the *first occurrence* of 'pear' in *each* line is replace. If there are other occurrences in a line, they will not be replaced unless if the g modifier is included at the end of the expression, as in

```
sed 's/pear/golabi/g' fruit.txt
```

To search for more than one string, prefix the expressions with -e

```
echo 'Duirng daytime we have sunlight' | sed -e 's/
day/night/' -e 's/sun/moon/'
Duirng nighttime we have moonlight
```

8.9 sed- Stream editor

sed and Regular Expressions

Use regular expressions with sed

```
echo 'Who needs vowels?' | sed 's/[aeiou]/./g'
Wh. n..ds v.w.ls?
```

Examples:

Indent every line two spaces

```
sed 's/^/  /g' fruit.txt
  pear
  raspberry
```

...
Note the indentation in above output.

sed does not understand \t as 'tab character' (grep does). So we have to use the following sequence:
ctrl v <tab>
Note that this shows only as a whitespace in the copy-paste

```
sed 's/^/	/' fruit.txt
pear
raspberry
...
```

Note the indentation by a tab in above output.

You can use the ctrl v sequence with some other characters as well such as

^ v ↺
and
^ v ↩

Remove html tags and their contents from homepage.htm

```
sed -E 's/<[^<>]+>//g' homepage.html
```

Note: As each line is treated as a new stream, if the opening < and closing > are on two different lines the above will not work. So, for example

```
 <div id="content">
```

was correctly removed, but

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

was not.

Back references

Put a regex in round brackets, then refer to it as \1. Up to 9 back references are allowed.

```
echo daytime | sed -E 's/(...)/\1 or night/'
day or nighttime
```

```
echo "Kamran Layegh" | sed -E 's/([[:alpha:]]+) ([[:alpha:]]+)/\2, \1/'
Layegh, Kamran
```

8.10 sed- Regular expressions and back-references

cut

Can cut characters, bytes, or fields. Here we will look at characters and fields.

Cutting characters

cut -c (characters) from column 1 to 3 (1-based count)
Note: you get the same result if you miss out the 1. Also 3- would return from character 3 to end.

```
cut -c 1-3 fruit.txt
pea
ras
ban
```

To cut more than one portion of the text, create a comma-separated list

```
cut -c 1-3,5-6 fruit.txt
```

Take command history, pipe it into grep 'fruit' which picks out the lines that contain 'fruit', then pipe it into cut to select characters 8 to end

```
history | grep 'fruit' | cut -c 8-
```

Cutting fields

By default cut uses the tab character as field separator. Hence

```
cut -f 2,6 us_presidents.tsv
```

Will show fields (-f) i.e. columns 2 and 6. Use `2-6` to select fields 2 to 6.

You can of course pipe the result to a file by adding ` > someFile.txt` to the end of the above command to write the result to disk.

If the delimiter is any character other than the tab character, you specify it by using the -d option after the list value. The following uses the comma as the delimiter of fields

```
cut -f 2,6 -d ',' us_presidents.csv
```

Some other options

The `-s` option does nothing, i.e. returns unchanged, the lines that have no delimiter in them.

8.11 cut- Cutting select text portions

diff

Compare the contents of two files and points out the differences on a line-by-line basis.

The left-right terminology is useful. Put the old (original) file on the left and the new (modified) file on the right.

```
diff original_file.txt revised_file.txt
```

```
2d1
< line 2:  delete delete delete delete delete delete
6c5
< line 6:  change change change change change change
---
> line 6:  change        change        change
11a11
> line 12: append append append append append append
```

diff uses these abbreviations: `d` for deleted; `c` for changed; `a` for appended.

deleted

So, in the above output, `2d1` means *"Look at line 2 of the left file and line 1 of the right file and see a deletion"*

Next line
```
< line 2:  delete delete delete delete delete delete
```
the `<` means *"in the left file"*
So altogether, it says *"in the left file"*
`line 2:  delete delete delete delete delete delete`
"was deleted"

changed

Next, we have
```
6c5
< line 6:  change change change change change change
---
> line 6:  change         change         change
```
The `6c5` says *"Look at line 6 of the left file and line 5 of the right file and see a change"*

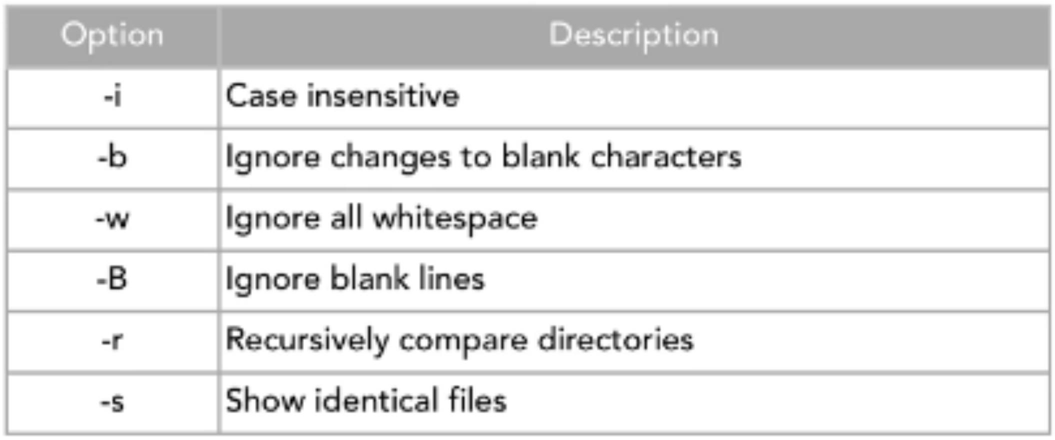
The three dashes separate the line 6 of the left file as it is now and line 6 of the right file.

appended

A similar pattern is used on the last two lines
```
11a11
> line 12: append append append append append append
```

diff options

diff comparison options

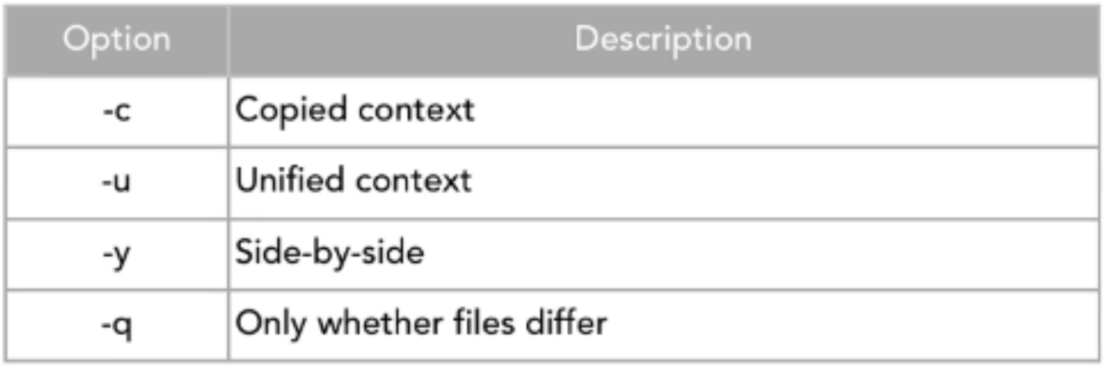


Option	Description
-i	Case insensitive
-b	Ignore changes to blank characters
-w	Ignore all whitespace
-B	Ignore blank lines
-r	Recursively compare directories
-s	Show identical files

8.12 diff- Comparing files

diff Controlling its output

diff output formats



Option	Description
-c	Copied context
-u	Unified context
-y	Side-by-side
-q	Only whether files differ

The `-c` option makes diff to output all the contents of both files with 3 coded markings: - for deleted; + for appended; ! for changed. The - can only appear in the left file. The + can only appear in the right file. The ! must appear in pairs in both files.

The `-y` option does the same thing, but puts the output side by side, rather than one above the other.

The `-u` option merges the two files and shows which lines will be deleted and which will be appended. A change is of course a deletion plus an insertion, so a change will appear as a - and a + on the same line number:

```
diff -u original_file.txt revised_file.txt
```

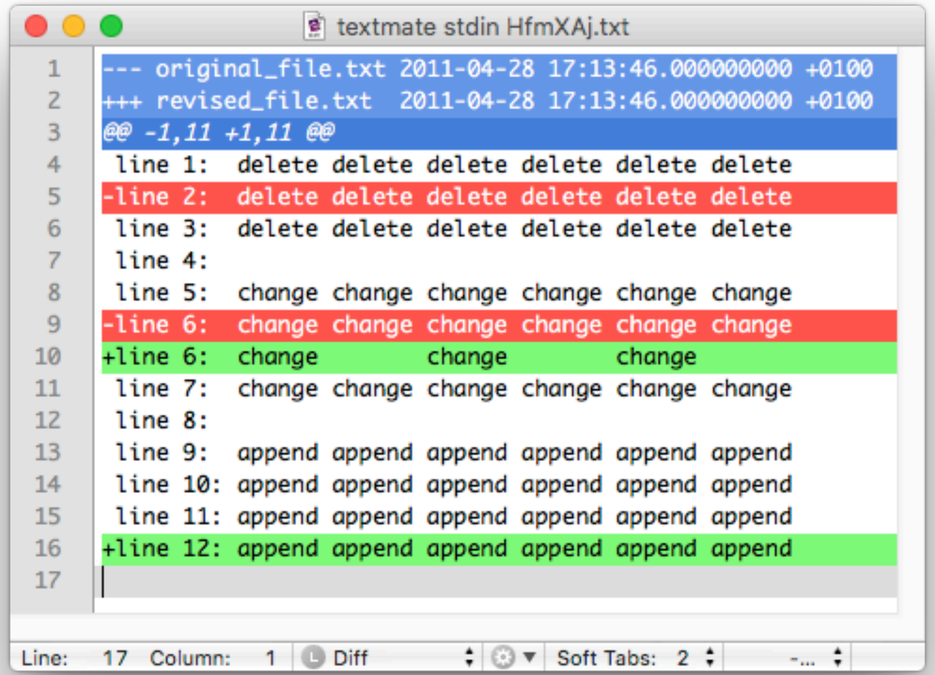
```
...
-line 6:  change change change change change change
+line 6:  change        change        change
...
```

The `-q` option simply tells us if the files differ or not.

If the files are identical, the `-q` option will fail silently. To force it to confirm that the files are identical use the `-qs` option.

We can pipe the output to TextMate

```
diff -u original_file.txt revised_file.txt | mate
```



```
--- original_file.txt 2011-04-28 17:13:46.000000000 +0100
+++ revised_file.txt  2011-04-28 17:13:46.000000000 +0100
@@ -1,11 +1,11 @@
 line 1:  delete delete delete delete delete delete
-line 2:  delete delete delete delete delete delete
 line 3:  delete delete delete delete delete delete
 line 4:
 line 5:  change change change change change change
-line 6:  change change change change change change
+line 6:  change        change        change
 line 7:  change change change change change change
 line 8:
 line 9:  append append append append append append
 line 10: append append append append append append
 line 11: append append append append append append
+line 12: append append append append append append
```

For a count of changes, pipe to `diffstat`, noting that `-u` option must be used for `diffstat` to work

```
diff -u original_file.txt revised_file.txt | diffstat
 revised_file.txt |    4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
```

Note that in `4 ++--` the 4 means four changes, the `++` means 2 insertions and the `--` means two deletions. This is because a change is a deletion plus an insertion

8.13 diff- Alternative formats

xargs

xargs executes arguments.

It parses an input stream into items. Then it loops through each item and passes them to a command. The command to which the items are passed is called the utility command. The exact manner in which the items are passed can be controlled by the `-n` option. `-n1` will cause the the items to be passed 1-at-a-time. `-n2` will cause the items to be passed 2-at-a-time, and so on. See examples below for illustration.

```
echo lorem_ipsum.txt | xargs -t  wc
wc lorem_ipsum.txt
     523    5289   36232 lorem_ipsum.txt
```

The -t option makes xargs to print out what it is doing — `wc lorem_ipsum.txt` — useful for debugging the command you are building.

As an example of looping (we pass the -t for illustration only)

```
echo lorem_ipsum.txt us_presidents.csv | xargs -t 
wc
wc lorem_ipsum.txt us_presidents.csv
     523    5289   36232 lorem_ipsum.txt
      45     121    4309 us_presidents.csv
     568    5410   40541 total
```

To see how the -n option works, see the following three examples. In the second one with -n2, for example, xargs is outputting 2 items at a time.

```
echo 1 2 3 4 | xargs -n1
1
2
3
4
```

```
echo 1 2 3 4 | xargs -n2
1 2
3 4
```

```
echo 1 2 3 4 | xargs -n3
1 2 3
4
```

The `-L` option

He is not very clear what this option does. See man pages for info. Roughly, it looks like:

the `-n2` option, for example, builds a command by outputting 2 items at a time, tokenised by *space*, whereas `-L2` option does the same but tokenised by *line break*, i.e. 2 lines at a time rather than 2 words at a time.

The -I option allows you to use a place holder, in the following example {}

```
cat fruit.txt | xargs -I {} echo "buy more {}"
buy more pear
buy more raspberry
...
```

Problem with filenames that contain spaces

```
ls ~/library | grep "^A.*"
Accounts
Acrobat User Data
Address Book Plug-Ins
...
```

Note the spaces. Now suppose we want to do something which each of these filenames using xargs.

```
ls ~/library | grep "^A.*" | xargs -n1
Accounts
Acrobat
User
Data
Address
```

Note that "Acrobat User Data" has broken up and become useless for processing.

Solution: Use the `-0` option

```
ls ~/library | grep "^A.*" | xargs -0 -n1
Accounts
Acrobat User Data
...
```

xargs some examples

1. Suppose we have a list of filenames in file_manifest.txt.

```
cat file_manifest.txt
people.txt
sorted_fruit.txt
...
```

We may want to do various things with their contents, but for the sake of this example let us say we want to concatenate them

```
cat file_manifest.txt | xargs cat
Kevin
Lynda
...
```

Pipe to less to be able to navigate the result more easily

```
cat file_manifest.txt | xargs cat | less
```

2. Suppose we have a list of people's names,

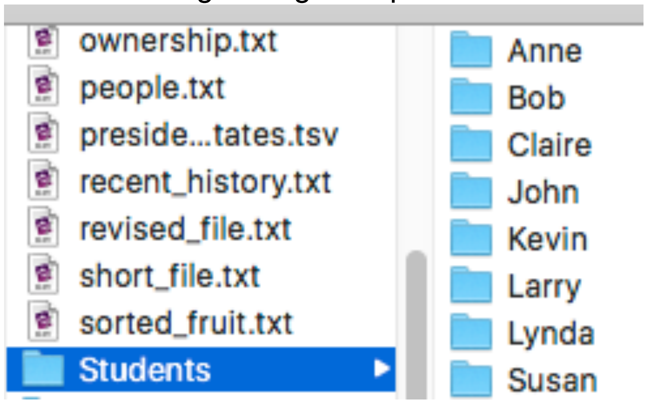
```
cat people.txt
Kevin
Lynda
```

...

and we would like to create a directory for each name

```
cat people.txt | uniq | xargs -I {} mkdir -p ./Students/{}
```

We are filtering through uniq in case some names are repeated.



3. Change permissions on all the files inside a given directory

```
find test1/ -type f -print0 | xargs -0 chmod 755
```

Says: find in test1 folder all files of type f (file) and print 0 (tag spaces with nul? This will be consumed by the -0 option to xargs to overcome the problem with spaces in filenames). Pipe all of this into xargs and the utility command is chmod 755.

4. Make a copy of al the files with name "new*.txt"

```
find . -name "new*.txt" -print0 | xargs -0 -I {} cp {} ./backup/{}.backup
```

Says: Find all files wit name "`new*.txt`" and `print0` them i.e. print appending null at the end of each filename, then pipe to xargs using the null terminator (-0) letting me define a placeholder (-I) which I do using {}. xargs should use cp (copy) as utility command. We pass to the utility command `{} ./backup/{}.backup` as argument.

8.15 xargs- Usage examples

T

c

m

T

c

m

T

c

m

T

c

m

T

c

m

• Bullet point

```
git config --global user.name "IMPORTANT CODE"
```

`code e.g xcode-select --install`

`file name`

Unobtrusively fade into the background

🟥 🟧 🟨 🟩 🟦 🟪 ⬜