172.168.7.152

2021-22

Exit from vi editor

ESC+shift+ZZ

1) $cat

To create a new file run the **cat command followed by the redirection operator > and the name of the file you want to create**. Press Enter type the text and once you are done press the CRTL+D to save the files.

Creating a File with cat Command

The cat command is mainly used to read and concatenate files, but it can also be used for creating new files.

To create a new file run the cat command followed by the redirection operator › and the name of the file you want to create. Press `Enter` type the text and once you are done press the `CRTL+D` to save the files.

```
cat > file1.txt
```

Creating a File with echo Command

The `echo` command prints the strings that are passed as arguments to the standard output, which can be redirected to a file.

To create a new file run the `echo` command followed by the text you want to print and use the redirection operator › to write the output to the file you want to create.

```
echo "Some line" > file1.txtCopy
```

If you want to create an empty simply use:

```
echo > file1.txtCopy
```

Creating a File using Heredoc

Here document or Heredoc is a type of redirection that allows you to pass multiple lines of input to a command.

This method is mostly used when you want to create a file containing multiple lines of text from a shell script.

For example, to create a new file `file1.txt` you would use the following code:

```
cat << EOF > file1.txt
Some line
```

```
Some other line
EOF
```

The body of the heredoc can contain variables, special characters, and commands.

Creating a Large File

Sometimes, for testing purposes, you might want to create a large data file. This is useful when you want to test the write speed of your drive or to test the download speed of your connection.

Using `dd` command

The `dd` command is primarily used to convert and copy files.

To create a file named `1G.test` with a size of 1GB you would run:

```
dd if=/dev/zero of=1G.test bs=1 count=0 seek=1GCopy
```

Using `fallocate` command

`fallocate` a command-line utility for allocating real disk space for files

The following command will create a new file named `1G.test` with a size of 1GB:

```
fallocate -l 1G 1G.testCopy
```

Conclusion

In this tutorial, you learned how to create a new file in Linux from the command line using various commands and redirection.

If the command line is not your thing you can easily create a blank text file using the right-click menu in the File Manager.

If you have questions, feel free to leave a comment below.

Knowing how to create a new file is an important skill for anyone using Linux on a regular basis. You can create a new file either from the command line or from the desktop file manager.

In this tutorial, we'll show you various ways to quickly create a new file in Linux using the command line.

Before you Begin

To create a new file you need to have write permissions on the parent directory. Otherwise, you will receive a permission denied error.

If you want to display the contents of a directory use the `ls` command .

Creating a File with touch Command

The `touch` command allows us to update the timestamps on existing files and directories as well as creating new, empty files.

The easiest and most memorable way to create new, empty files is by using the `touch` command.

To create a new file simply run the `touch` command followed by the name of file you want to create:

```
$ touch file1.txtCopy
```

If the file `file1.txt` doesn't exist the command above will create it, otherwise, it will change its timestamps.

To create multiple files at once, specify the file names separated by space:

```
$ touch file1.txt file2.txt file3.txtCopy
```

Creating a File with the Redirection Operator

Redirection allows you to capture the output from a command and send it as input to another command or file. There are two ways to redirect output to a

file. The > operator will overwrite an existing file, while the >> operator will append the output to the file.

To create an empty zero-length file simply specify the name of the file you want to create after the redirection operator:

```
$ > file1.txt
```

This is the shortest command to create a new file in Linux.

When creating a file using a redirection, be careful not to overwrite an important existing file.

Vi Commands

Note: vi is often a symbolic link to vim (vi Improved) or an alias to vim.

It's easy to invoke vi. At the command line, you type `vi <filename>` to either create a new file, or to edit an existing one.

```
$ vi filename.txt
```

The vi editor has two modes: Command and Insert. When you first open a file with vi, you are in Command mode. Command mode means that you can use keyboard keys to navigate, delete, copy, paste, and do a number of other tasks—except entering text. To enter Insert mode, press `i`. In Insert mode, you can enter text, use the **Enter** key to go to a new line, use the arrow keys to navigate text, and use vi as a free-form text editor. To return to Command mode, press the **Esc** key once.

Note: In vi's Command mode, almost every letter on the keyboard has a function.

To save a file, you must first be in Command mode. Press **Esc** to enter Command mode, and then type `:wq` to write and quit the file. The other, quicker option is to use the keyboard shortcut `ZZ` to write and quit. To the non-vi initiated, *write* means save, and *quit* means exit vi. If you've made mistakes along the way in your editing and want to back out (abandon) all non-saved changes, enter Command mode by pressing **Esc** and type `:q!` This command quits without saving any changes and exits vi.

The best way to learn vi is to create a new file and try it out for yourself. Feel free to use the common keyboard shortcut table below to help you learn vi's

extensive vocabulary. This list of shortcuts is by no means exhaustive, but they will enable you to edit files and learn vi in a short amount of time.

Always make a copy of an existing file prior to editing with vi or any editor. This is especially critical when editing system and configuration files.

Command	Purpose
`$ vi <filename>`	Open or edit a file.
`i`	Switch to Insert mode.
Esc	Switch to Command mode.
`:w`	Save and continue editing.
`:wq` or `ZZ`	Save and quit/exit vi.
`:q!`	Quit vi and do not save changes.
`yy`	Yank (copy a line of text).
`p`	Paste a line of yanked text below the current line.
`o`	Open a new line under the current line.
`O`	Open a new line above the current

Command	Purpose
	line.
A	Append to the end of the line.
a	Append after the cursor's current position.
I	Insert text at the beginning of the current line.
b	Go to the beginning of the word.
e	Go to the end of the word.
x	Delete a single character.
dd	Delete an entire line.
Xdd	Delete X number of lines.
Xyy	Yank X number of lines.
G	Go to the last line in a file.
XG	Go to line X in a file.

Command	Purpose
gg	Go to the first line in a file.
:num	Display the current line's line number.
h	Move left one character.
j	Move down one line.
k	Move up one line.
l	Move right one character.

Basic UNIX commands

Note: not all of these are actually part of UNIX itself, and you may not find them on all UNIX machines. But they can all be used on **turing** in essentially the same way, by

typing the command and hitting return. Note that some of these commands are different on non-Solaris machines - see SunOS differences. If you've made a typo, the easiest thing to do is hit **CTRL-u** to cancel the whole line. But you can also edit the command line (see the guide to More UNIX). UNIX is case-sensitive.

Files

- **ls** --- lists your files
  ls -l --- lists your files in 'long format', which contains lots of useful information, e.g. the exact size of the file, who owns the file and who has the right to look at it, and when it was last modified.
  ls -a --- lists all files, including the ones whose filenames begin in a dot, which you do not always want to see.
  There are many more options, for example to list files by size, by date, recursively etc.
- **more** ***filename*** --- shows the first part of a file, just as much as will fit on one screen. Just hit the space bar to see more or **q** to quit. You can use ***/pattern*** to search for a pattern.
- **emacs** ***filename*** --- is an editor that lets you create and edit a file. See the emacs page.
- **mv** ***filename1 filename2*** --- moves a file (i.e. gives it a different name, or moves it into a different directory (see below)
- **cp** ***filename1 filename2*** --- copies a file
- **rm** ***filename*** --- removes a file. It is wise to use the option rm -i, which will ask you for confirmation before actually deleting anything. You can make this your default by making an alias in your .cshrc file.
- **diff** ***filename1 filename2*** --- compares files, and shows where they differ
- **wc** ***filename*** --- tells you how many lines, words, and characters there are in a file
- **chmod** ***options filename*** --- lets you change the read, write, and execute permissions on your files. The default is that only you can look at them and change them, but you may sometimes want to change these permissions. For example, **chmod o+r** ***filename*** will make the file readable for everyone, and **chmod o-r** ***filename*** will make it unreadable for others again. Note that for someone to be able to actually look at the file the directories it is in need to be at least executable. See help protection for more details.
- File Compression

OpSy. Lab 05.C

- **gzip *filename*** --- compresses files, so that they take up much less space. Usually text files compress to about half their original size, but it depends very much on the size of the file and the nature of the contents. There are other tools for this purpose, too (e.g. **compress**), but gzip usually gives the highest compression rate. Gzip produces files with the ending '.gz' appended to the original filename.
- **gunzip *filename*** --- uncompresses files compressed by gzip.
- **gzcat *filename*** --- lets you look at a gzipped file without actually having to gunzip it (same as **gunzip -c**). You can even print it directly, using **gzcat *filename* | lpr**

- printing
  - **lpr *filename*** --- print. Use the -P option to specify the printer name if you want to use a printer other than your default printer. For example, if you want to print double-sided, use 'lpr -Pvalkyr-d', or if you're at CSLI, you may want to use 'lpr -Pcord115-d'. See 'help printers' for more information about printers and their locations.
  - **lpq** --- check out the printer queue, e.g. to get the number needed for removal, or to see how many other files will be printed before yours will come out
  - **lprm *jobnumber*** --- remove something from the printer queue. You can find the job number by using lpq. Theoretically you also have to specify a printer name, but this isn't necessary as long as you use your default printer in the department.
  - **genscript** --- converts plain text files into postscript for printing, and gives you some options for formatting. Consider making an alias like **alias ecop 'genscript -2 -r \!* | lpr -h -Pvalkyr'** to print two pages on one piece of paper.
  - **dvips *filename*** --- print **.dvi** files (i.e. files produced by LaTeX). You can use **dviselect** to print only selected pages. See the LaTeX page for more information about how to save paper when printing drafts.

Directories

Directories, like folders on a Macintosh, are used to group files together in a hierarchical structure.

- **mkdir *dirname*** --- make a new directory
- **cd *dirname*** --- change directory. You basically 'go' to another directory, and you will see the files in that directory when you do 'ls'. You always start out in

your 'home directory', and you can get back there by typing 'cd' without arguments. 'cd ..' will get you one level up from your current position. You don't have to walk along step by step - you can make big leaps or avoid walking around by specifying pathnames.

- **pwd** --- tells you where you currently are.

Finding things

- **ff** --- find files anywhere on the system. This can be extremely useful if you've forgotten in which directory you put a file, but do remember the name. In fact, if you use **ff -p** you don't even need the full name, just the beginning. This can also be useful for finding other things on the system, e.g. documentation.
- **grep *string filename(s)*** --- looks for the string in the files. This can be useful a lot of purposes, e.g. finding the right file among many, figuring out which is the right version of something, and even doing serious corpus work. grep comes in several varieties (**grep**, **egrep**, and **fgrep**) and has a lot of very flexible options. Check out the man pages if this sounds good to you.

About other people

- **w** --- tells you who's logged in, and what they're doing. Especially useful: the 'idle' part. This allows you to see whether they're actually sitting there typing away at their keyboards right at the moment.
- **who** --- tells you who's logged on, and where they're coming from. Useful if you're looking for someone who's actually physically in the same building as you, or in some other particular location.
- **finger *username*** --- gives you lots of information about that user, e.g. when they last read their mail and whether they're logged in. Often people put other practical information, such as phone numbers and addresses, in a file called **.plan**. This information is also displayed by 'finger'.
- **last -1 *username*** --- tells you when the user last logged on and off and from where. Without any options, **last** will give you a list of everyone's logins.
- **talk *username*** --- lets you have a (typed) conversation with another user
- **write *username*** --- lets you exchange one-line messages with another user
- **elm** --- lets you send e-mail messages to people around the world (and, of course, read them). It's not the only mailer you can use, but the one we recommend. See the elm page, and find out about the departmental mailing lists (which you can also find in /user/linguistics/helpfile).

About your (electronic) self

- **whoami** --- returns your username. Sounds useless, but isn't. You may need to find out who it is who forgot to log out somewhere, and make sure *you* have logged out.
- **finger** & .plan files
  of course you can finger yourself, too. That can be useful e.g. as a quick check whether you got new mail. Try to create a useful .plan file soon. Look at other people's .plan files for ideas. The file needs to be readable for everyone in order to be visible through 'finger'. Do 'chmod a+r .plan' if necessary. You should realize that this information is accessible from anywhere in the world, not just to other people on turing.
- **passwd** --- lets you change your password, which you should do regularly (at least once a year). See the LRB guide and/or look at help password.
- **ps -u *yourusername*** --- lists your processes. Contains lots of information about them, including the process ID, which you need if you have to kill a process. Normally, when you have been kicked out of a dialin session or have otherwise managed to get yourself disconnected abruptly, this list will contain the processes you need to kill. Those may include the shell (tcsh or whatever you're using), and anything you were running, for example emacs or elm. Be careful not to kill your current shell - the one with the number closer to the one of the ps command you're currently running. But if it happens, don't panic. Just try again :) If you're using an X-display you may have to kill some X processes before you can start them again. These will show only when you use **ps -efl**, because they're root processes.
- **kill *PID*** --- kills (ends) the processes with the ID you gave. This works only for your own processes, of course. Get the ID by using **ps**. If the process doesn't 'die' properly, use the option -9. But attempt without that option first, because it doesn't give the process a chance to finish possibly important business before dying. You may need to kill processes for example if your modem connection was interrupted and you didn't get logged out properly, which sometimes happens.
- **quota -v** --- show what your disk quota is (i.e. how much space you have to store files), how much you're actually using, and in case you've exceeded your quota (which you'll be given an automatic warning about by the system) how much time you have left to sort them out (by deleting or gzipping some, or moving them to your own computer).

- **du *filename*** --- shows the disk usage of the files and directories in *filename* (without argument the current directory is used). **du -s** gives only a total.
- **last *yourusername*** --- lists your last logins. Can be a useful memory aid for when you were where, how long you've been working for, and keeping track of your phonebill if you're making a non-local phonecall for dialling in.

Connecting to the outside world

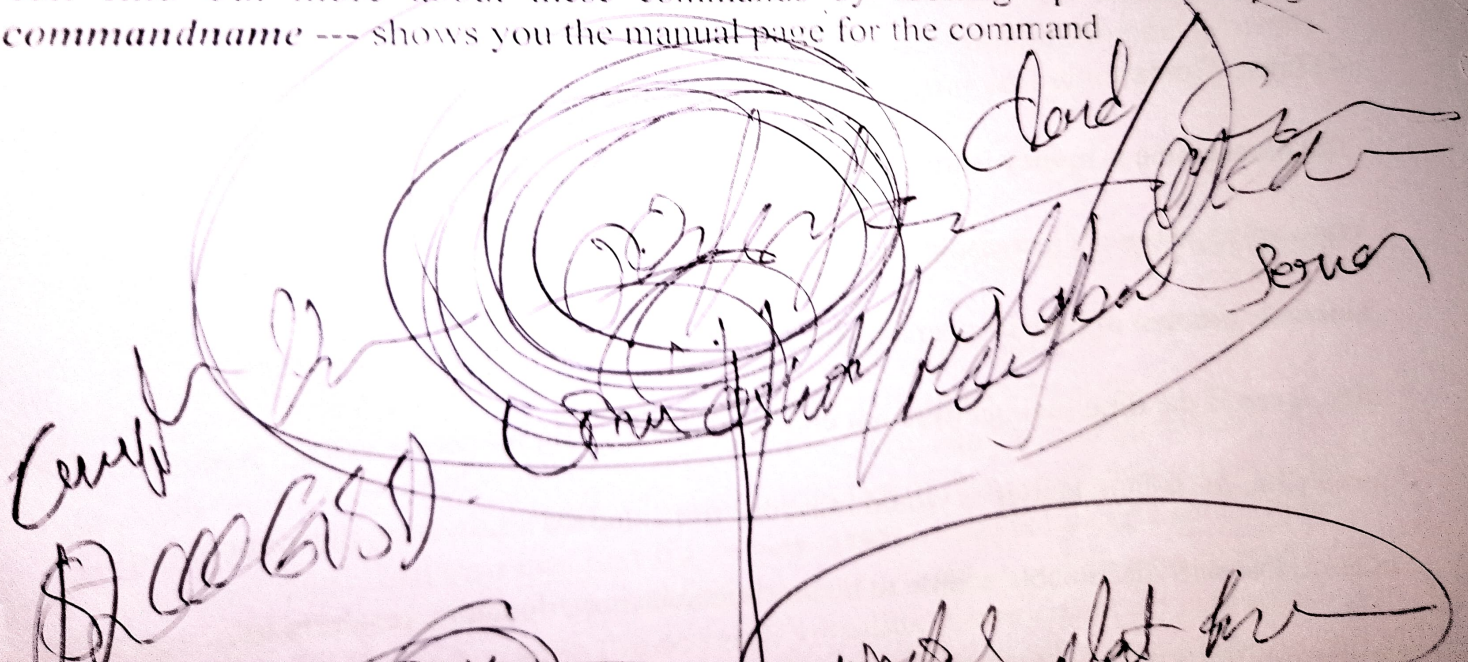
- **nn** --- allows you to read news. It will first let you read the news local to turing, and then the remote news. If you want to read only the local or remote news, you can use **nnl** or **nnr**, respectively. To learn more about **nn** type **nn**, then \tty{:man}, then \tty{=.*}, then \tty{Z}, then hit the space bar to step through the manual. Or look at the man page. Or check out the hypertext nn FAQ - probably the easiest and most fun way to go.
- **rlogin *hostname*** --- lets you connect to a remote host
- **telnet *hostname*** --- also lets you connect to a remote host. Use **rlogin** whenever possible.
- **ftp *hostname*** --- lets you download files from a remote host which is set up as an ftp-server. This is a common method for exchanging academic papers and drafts. If you need to make a paper of yours available in this way, you can (temporarily) put a copy in /user/ftp/pub/TMP. For more permanent solutions, ask Emma. The most important commands within ftp are **get** for getting files from the remote machine, and **put** for putting them there (**mget** and **mput** let you specify more than one file at once). Sounds straightforward, but be sure not to confuse the two, especially when your physical location doesn't correspond to the direction of the ftp connection you're making. ftp just overwrites files with the same filename. If you're transferring anything other than ASCII text, use binary mode.
- **lynx** --- lets you browse the web from an ordinary terminal. Of course you can see only the text, not the pictures. You can type any URL as an argument to the **G** command. When you're doing this from any Stanford host you can leave out the **.stanford.edu** part of the URL when connecting to Stanford URLs. Type **H** at any time to learn more about **lynx**, and **Q** to exit.

Miscellaneous tools

- **webster *word*** --- looks up the word in an electronic version of Webster's dictionary and returns the definition(s)

- **date** --- shows the current date and time.
- **cal** --- shows a calendar of the current month. Use e.g., 'cal 10 1995' to get that for October 95, or 'cal 1995' to get the whole year.

You can find out more about these commands by looking up their manpages:
man *commandname* --- shows you the manual page for the command



More UNIX Commands

*I have noticed that the overwhelming majority of visitors come to this page via a Lycos search. This page is probably *not* what you're looking for - see the links at the bottom of this page for more useful information!*

- `jobs` --- lists your currently active jobs (those that you put in the background) and their job numbers. Useful to determine which one you want to foreground if you have lots of them.
- `bg` --- background a job after suspending it.
- `fg %jobnumber` --- foreground a job
- `!!` --- repeat the previous command (but `CTRL-p`, is safer, because you have hit return in addition)
- `!pattern` --- repeat the last command that starts with *pattern*
- `echo $VARIABLE` --- shows the value of an environment variable
- `setenv` --- lets you set environment variables. For example, if you typed a wrong value for the `TERM` variable when logging in, you don't have to log out and start over, but you can just do `setenv TERM vt100` (or whatever). To see what all your environment variables are set to, type `env`. The one that you're most likely to have to set is the `DISPLAY` variable, when using an X-display.
- `unset VAR` --- lets you un-set environment variables. Useful, for example, if you've usually set `autologout` but want to stay logged on for a while without typing for some reason, or if you set the `DISPLAY` variable automatically but want to avoid opening windows for some reason.
- `source filename` --- you need to source your dotfiles after making changes for them to take effect (or log off and in again)
- `load` --- will show you the load average graphically
- `ispell filename` --- will check the spelling in your file. If you're running it on a LaTeX file use the `-T` option to tell it to ignore the LaTeX commands. You can create and use your own dictionary to avoid having it tell you that your own name, those of fellow linguists, and linguistics terminology are a typos in every paper you write.
- `weblint` --- checks the syntax of html files
- `latex2html` --- translates LaTeX files into HTML
- `wn word option` --- lets you access the WordNet database and display, for example, synonyms, hypernyms, or hyponyms, depending on the option you select

Command editing in the tcsh

These things are the same as in emacs:

`Backspace` --- delete previous character

`CTRL-d` --- delete next character

`CTRL-k` --- delete rest of line

`CTRL-a` --- go to start of line

`CTRL-e` --- go to end of line

`CTRL-b` --- go backwards without deleting

`CTRL-f` --- go forward without deleting

Other useful things

`TAB` --- complete filename or command up to the point of uniqueness

`CTRL-u` --- cancel whole line

`CTRL-p` --- show the last command typed, then the one before that, etc.

(you can also use the cursor up key for this)

`CTRL-n` --- go forwards in the history of commands

(you can also use the cursor down key for this)

`CTRL-c` --- cancel the processes after it has started

`CTRL-z` --- suspend a running process (e.g. in order to do something else in between)

you can then put the process in the background with `bg`

`CTRL-l` --- redraws the screen

`|` (**piping**) --- Lets you execute any number of commands in a sequence.

The second command will be executed once the first is done, and so forth, using the previous command's output as input. You can achieve the same effect by putting the output in a file and giving the filename as an argument to the second command, but that would be much more complicated, and you'd have to remember to remove all the junkfiles afterwards. Some examples that show the usefulness of this:

`ls | more` --- will show you one screenful at a time, which is useful with any command that will produce a lot of output, e.g. also `ps -aux`

`man ls | grep time` --- checks whether the man page for `ls` has something to say about listing files by time - very useful when you have a suspicion some command may be capable of doing what you want, but you aren't sure.

`ls -lR | grep dvi` --- will show you all your dvi files - useful to solve disk space problems, since they're large and usually can be deleted.