There is a directory available to you where you can practice filename expansions.

```
cp -r ~eric.brauer/uli101/wordlist ~
```

```
cd ~/wordlist
```

```
less 0README
```

Introduction to Expansions

The shell not only accepts specific filenames. You can also use 'expansions' to capture more than one file.

```
touch bat bet bit bot boat bath && ls
```

```
bat bet bit bot boat bath
```

Aside: We can combine two commands on one line using `&&` like you see above. The second command will only run if the first completed with no errors.

If we type in `ls -l bat`, we will get back long list information about only that specific file, identified by three *explicit* characters: b, a, and t.

```
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bat
```

Matching Any Single Character

Ok, so here's the first expansion symbol:

- ? : matches a single character

```
ls -l b?t
```

```
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bat
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bet
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bit
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bot
```

Here, all the matches are lower case letters. But keep in mind that *character* refers to any *letter, number, or symbol that can be used to name a file*.

Matching a Single Specified Character

That last example was very *greedy*, because any single character between the 'b' and the 't' would be accepted. Let's say that instead, we want to specify which single character we want to match. Let's only match files with 'a' or 'o'. To do this, we will use square brackets.

```
ls -l b[ao]t
```

```
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bat
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bot
```

So we have matched two filenames. Both filenames are three letters long, and both start with ‘b’ and end with ‘t’. The single character in the middle is either ‘a’ or ‘o’.

Note #1: Notice that we *didn’t* match the filename `boat`! Remember that square brackets only match a *single* character!

Inverting A Single Specified Character

We can invert our previous match by putting an exclamation mark *inside* the square brackets:

```
ls -l b[!ao]t
```

```
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bet
-rw-rw-r-- 1 eric eric 0 Sep  8 15:22 bit
```

This is matching any filename that is three characters long, starting with ‘b’ and ending with ‘t’ as long as the middle character *isn’t* an ‘a’ or an ‘o’.

Note #2: A common mistake by students is to put single characters between square brackets, ie. `b[o]t`. *There is no difference between `b[o]t` and `bot`*, except that using a single character will lose you marks on a test. However, `b[!o]t` will match any character that isn’t ‘o’ and is therefore acceptable.

Selecting A Single Character With A Range

As well specifying characters inside square brackets, we can use ranges. All of the below are acceptable:

```
ls -l [A-B]at ls -l [A-Za-z]at ls -l lecture[0-9] ls -l abc[a-z0-9-_]
```

Selecting Zero-To-Many Characters

Using either the `?` or `[ ]` will match only a single character (and never zero characters). Our next symbol will do this:

- * : matches zero to many of any character

Let’s demonstrate this by removing our old files and creating new files:

```
rm * touch .f f foo foot footprint && ls
```

```
f foo foot footprint
```

Notice that `.f` seems to be missing. Any file or directory that begins with ‘.’ will be considered a *hidden* file. To view them you will have to use:

```
ls -a
```

```
. .. .f foo foot footprint
```

Notice we are also seeing '.' and '..', which represent the current directory and the parent directory.

```
ls f*
```

```
f foo foot footprint
```

We are matching any filename that starts with 'f', but then after that we match with zero characters, we match with two characters (oo), we match with three characters (oot), and with even more characters (ootprint). Basically, we are saying we don't care about how many characters come after 'f', just as long as that 'f' is there at the beginning.

Note #3: Notice that we *didn't* match with the hidden file `.f`!

```
ls f*t
```

```
foot footprint
```

This time, we don't care how many characters are in between 'f' and 't', just as long as our filename begins with 'f' and ends with 't'.

Practical Applications

We've been using expansions with `ls`, but that doesn't have to be the case.

```
cp *.jpg images/
```

This is going to match any filename that ends with '.jpg', and copy those files into a directory called images. This is useful if you are reorganizing a web project.

```
rm 2018-0[1-5]-*.c
```

Let's say that you're sick of looking at your old source code from the Winter semester of 2018. (And you named all of your source code with dates!) Any date that matches between the month of 01 and 05 will get deleted.

```
cat lecture3?.md > lecture3.md
```

Maybe at some point I will decide that instead of having two lectures a week, I want to create one big markdown file for each week. `cat` will combine two textfiles into one.

---

find Part II

File expansions get *very* useful with the `find` command, since usually we lost something because we typed it in wrong. The first thing we can do to expand our search is to *ignore case*. This is easy to do.

Instead of this command:

```
find ~ -name report.pdf
```

Use this command:

```
find ~ -iname report.pdf
```

For every alphabetic character, the find command will include all upper and lower case possibilities.

But you will also benefit sometimes from using expansions such as 'report*' in your search.

We aren't limited to just searching by name, either. We can combine a 'name' or 'iname' search with other conditions. For example, we might want to limit our search to only *files* or only *directories*.

```
find ~ -iname report* -type f
```

```
find ~ -iname oxford -type d
```

You can combine this with filtering your search by *time*. You can specify that you want to see when files were *changed*, and when they were *accessed*.

Try this. We are going to use the file called 'friendly' from last week. If you don't have this file, no problem. The command below will create it:

```
echo "This is a test" >> friendly
```

```
find . -mmin -20
```

You will see testfile show up in results. `-mmin` stands for *modification* + *minutes*. `-20` sets the search for less than 20. So this search should return all files that have had their contents changed less than 20 minutes ago. **Note: Timestamps in Linux are weird. More info.**

Finally, one more useful search. This is for size.

```
find ~ -size +1G
```

This will return all files *larger* than 1 Gigabyte. G = Gigabyte, M = Megabyte, k = kilobyte.

`find` is an complicated tool. By default, it will use `ls` to list the positive results, but you can even specify other commands to run on the results. I encourage you to search through examples and options online and through the man pages.

Head and Tail

If you want to only see the first ten lines of a file, use `head`. If you want to only see the last ten lines of a file, use `tail`. You can also specify the number of lines with an argument:

```
head -3 Acceptable-Use-Policy
```

```
Please refer to http://www.senecac.on.ca/policies/itau.html for the latest and
up to date copy of the Information Technology Acceptable Use.
```

Learning About a File

The most common way to learn about files is to use `ls -l`, which you've seen before. But there are more options.

Let's use an image from last week:

```
ls -l lecture2a-nautilus.png
```

```
-rw-r--r-- 1 eric eric 70K Sep  9 21:21 lecture2a-nautilus.png
```

Let's learn more about this with the `file` command.

```
file lecture2a-nautilus.png
```

```
lecture2a-nautilus.png: PNG image data, 1242 x 696, 8-bit/color RGBA, non-interlaced
```

In Windows, traditionally the *extension* of the file was very important. If you took an image called 'file.jpg' and renamed it 'file.txt', it would probably cause notepad.exe to choke on that file. In Linux, extensions aren't required, and we mostly keep them around by convention.

```
cp lecture2a-nautilus.png test.txt
```

```
file test.txt
```

```
test.txt: PNG image data, 1242 x 696, 8-bit/color RGBA, non-interlaced
```

Summary

Commands

- find, continued: type, size, amin, cmin
- file: Information about files
- head: Show first X lines
- tail: Show last X lines
- history: Show history

Expansions

- *: Zero or more characters
- ?: One character
- [ ]: Match *one* character of whatever's inside the brackets
- [!]: Exclude *one* character of whatever's inside the brackets