Workshop 1: **Introduction to UNIX command-line**

Peter Scott, PhD | pscott17@ucla.edu
QCBio Fellow

Swiss Army knife" set of tools

Day 1

pwd - report your current directory

cd *<to where>* - change your current directory

ls *<directory>* -list contents of directory

cp *<old file> <new file>* - copy file

cp –r *<old dir> <new dir>* - copy a directory and its contents

mv *<old file/dir> <new file/dir>* - move (or rename)

rm *<file>* -delete a file

rm –r <dir> - remove a directory and its contents

mkdir *<new directory name*> -make a directory

Using hoffman2

- Log on to hoffman2:
  - `ssh myname@hoffman2.idre.ucla.edu`

- Request an interactive shell:
  - `qrsh -l i,time=3:00:00,h_data=2g`

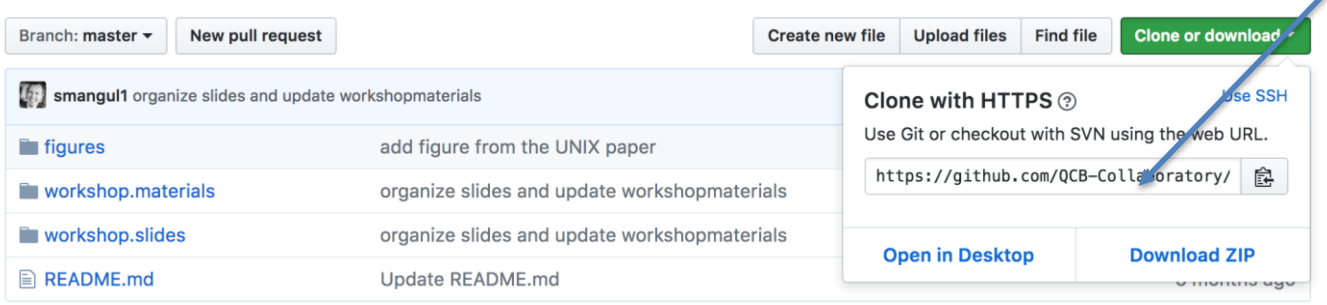
```
You can make a “program” with the
interactive shell script
```

Copy the working materials

```
[pscott17@login2 ~]$ git clone
https://github.com/p-scott17/Intro2Unix.git

Initialized empty Git repository in
/u/home/b/brigitta/code/W1.UNIX.command.line/.git/
remote: Counting objects: 88, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 88 (delta 0), reused 7 (delta 0), pack-reused 79
Unpacking objects: 100% (88/88), done.
```

https://github.com/p-scott17/Intro2Unix.git



smangul1 organize slides and update workshopmaterials	
figures	add figure from the UNIX paper
workshop.materials	organize slides and update workshopmaterials
workshop.slides	organize slides and update workshopmaterials
README.md	Update README.md

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

```
https://github.com/QCB-Collaboratory/
```

https://qcb.ucla.edu/collaboratory/workshops/introtounix/

Working materials

- `cd Intro2Unix`
- `ls`

```
bwa_loop_pipe.sh  bwa_loop.sh  bwa.sh
day1_Unix_PAS_winter2020.pdf day2_Unix_PAS_winter2020.pdf
day3_Unix_PAS_winter2020.pdfemp.txt       hg19.gtf
file_sed.txt   f.txt numbers.txt   regex2.txt regex_sort.txt
regex.txt  sales.txt  tobe.txt   toy3.reads.fastq
toy2.reads.fastq toy.reads.fastq   toy.ref.fasta
    toy.ref.fasta.amb  toy.ref.fasta.ann   toy.ref.fasta.bwt
    toy.ref.fasta.pac  toy.ref.fasta.sa
```

Relative vs. absolute path

- A file or a directory can be referred to by
  - Relative path
    - **if you are at** `/u/home/p/pscott17/test/new/`
    - and you want `text.txt`
    - `../test.txt`
  - Absolute path
    - `/u/home/p/pscott17/test/test.txt`

Relative

Absolute

245 Highland Ave, Manhattan Beach, California 90266

File permissions

- Each file in Unix has an associated permission level
- This allows the user to prevent others from reading/writing/executing their files or directories
- Use "ls -l *filename*" to find the permission level of that file
- There are 3 kinds of people in the world: you (user), your friends (group) and

  the world (others).

Permission levels

- **“r”** means “read only” permission
- **“w”** means “write” permission
- **“x”** means “execute” permission
  - In case of directory, **“x”** grants permission to list directory contents

File Permissions

```
-rw-r--r-- 1 pscott17 hbshaffe  72 Mar 11 14:22 large.txt
-rw-r--r-- 1 pscott17 hbshaffe 263 Mar 11 15:18 new.tar
-rw-r--r-- 1 pscott17 hbshaffe  13 Mar 11 15:27 test.txt
drwxr-xr-x 2 pscott17 hbshaffe4096 Mar 11 15:36 dfgdf
```

Type

User (you)

File Permissions

```
-rw-r--r-- 1 pscott17 hbshaffe  72 Mar 11 14:22 large.txt
-rw-r--r-- 1 pscott17 hbshaffe 263 Mar 11 15:18 new.tar
-rw-r--r-- 1 pscott17 hbshaffe  13 Mar 11 15:27 test.txt
drwxr-xr-x 2 pscott17 hbshaffe4096 Mar 11 15:36 dfgdf
```

Type

Group

File Permissions

```
-rw-r--r-- 1 pscott17 hbshaffe   72 Mar 11 14:22 large.txt
-rw-r--r-- 1 pscott17 hbshaffe  263 Mar 11 15:18 new.tar
-rw-r--r-- 1 pscott17 hbshaffe   13 Mar 11 15:27 test.txt
drwxr-xr-x 2 pscott17 hbshaffe4096 Mar 11 15:36 dfgdf
```

Type

“The World”

Command: chmod

- If you own the file, you can change it’s permissions with “chmod”
  - Syntax:

```
chmod [user/group/others/all]+-[permission] [file(s)]
```

```
[pscott17@login2 test]$ ls -l
drwxr-xr-x 3 pscott17 hbshaffe4096 Mar 11 15:23 archive
-rw-r--r-- 1 pscott17 hbshaffe  72 Mar 11 14:22 large.txt
-rw-r--r-- 1 pscott17 hbshaffe 263 Mar 11 15:18 new.tar
-rw-r--r-- 1 pscott17 hbshaffe  13 Mar 11 15:27 test.txt
[pscott17@login2 test]$ chmod g+w large.txt
[pscott17@login2 test]$ ls -l
drwxr-xr-x 3 pscott17 hbshaffe4096 Mar 11 15:23 archive
-rw-rw-r-- 1 pscott17 hbshaffe  72 Mar 11 14:22 large.txt
-rw-r--r-- 1 pscott17 hbshaffe 263 Mar 11 15:18 new.tar
-rw-r--r-- 1 pscott17 hbshaffe  13 Mar 11 15:27 test.txt
```

Redirection

- `program_a`
  - display program_a' s output at the terminal
- `program_a > file.txt`
  - program_a' s output is written to file.txt
  - ">" will **overwrite** any existing data in file.txt
- `program_a < input.txt`
  - program_a gets its input from a file called "input.txt"
- `program_a >> file.txt`
  - program_a's output is **appended** to the end of file.txt

Let’s practice!

```
[pscott17@login4 test]$ wc -l large.txt
300 large.txt
[pscott17@login4 test]$ wc -l large.txt > f_ls.txt
[pscott17@login4 test]$ head f_ls.txt
[pscott17@login4 test]$ ls > f_ls.txt
[pscott17@login4 test]$ head f_ls.txt
[pscott17@login4 test]$ head large.txt  >> f_ls.txt
[pscott17@login4 test]$ head f_ls.txt
```

Pipeline

pipe character

- `program_a | program_b`
  - program_a's output becomes program_b's input
  - Analogous to

```
program_a > temp.txt
program_b < temp.txt
```

Command: **wc**

- To count the characters, words, and lines in a file use **wc**

```
wc <filename>
```

  - The first column in the output is lines, the second is words, and the last is characters
  - **-l** to count the lines

```
#lines          #words       #characters
                   ↓
         300  300  1092  large.txt
```

Let's practice!

```
[pscott17@login2 test]$ wc test.txt
300 300 1092 large.txt
[pscott17@login2 test]$ wc -l test.txt
300 large.txt
[pscott17@login2 test]$ ls | wc -l
5
```

Command : **cat**

- Concatenate files together and displayed in the terminal.

```
cat <file1> <file2> …
```

```
[pscott17@login2 test]$ cat large.txt f_ls.txt | wc -l
301
[pscott17@login2 test]$ cat large.txt test.txt > all.txt
[pscott17@login2 test]$ tail -n 3 all.txt
299
300
300 large.txt
```

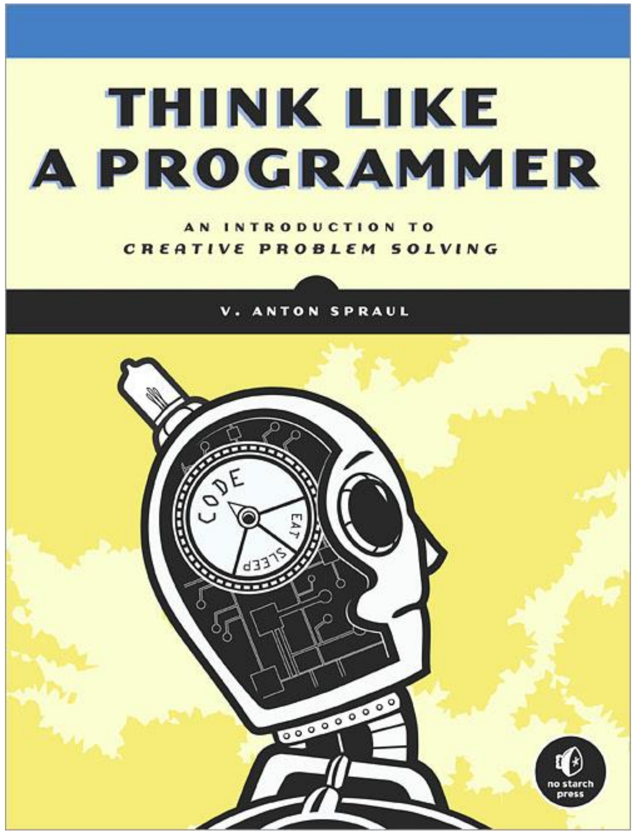
Directory

Find

- find new -name test.txt -type f

File

Tools for processing text files



Command : **grep**

- allows to search one file or multiple files for lines that contain a certain string
- g/re/p (globally search a regular expression and print)
- **grep** options
  - lines not containing the selected string (-v)
  - line numbers where the string occurs (-n)
  - number of lines containing the string (-c)
  - filenames where the string occurs (-l)
  - makes the match case-insensitive (-i)

Grep syntax treats the first argument as the pattern and the rest as filenames

Let’s practice!

```
[pscott17@login4 test]$ grep "1" large.txt
1
10
…
19
[pscott17@login4 test]$ grep -n "1" large.txt
1:1
10:10
…
19:19
[pscott17@login4 test]$ grep -c "1" large.txt
138
[pscott17@login4 test]$ grep -l "1" large.txt f_ls.txt
large.txt
[pscott17@login4 test]$ grep "1" large.txt f_ls.txt
large.txt:1
large.txt:10
…
```

Alternative?

Grep syntax treats the first argument as the pattern and the rest as filenames

Lines corresponding to chr2

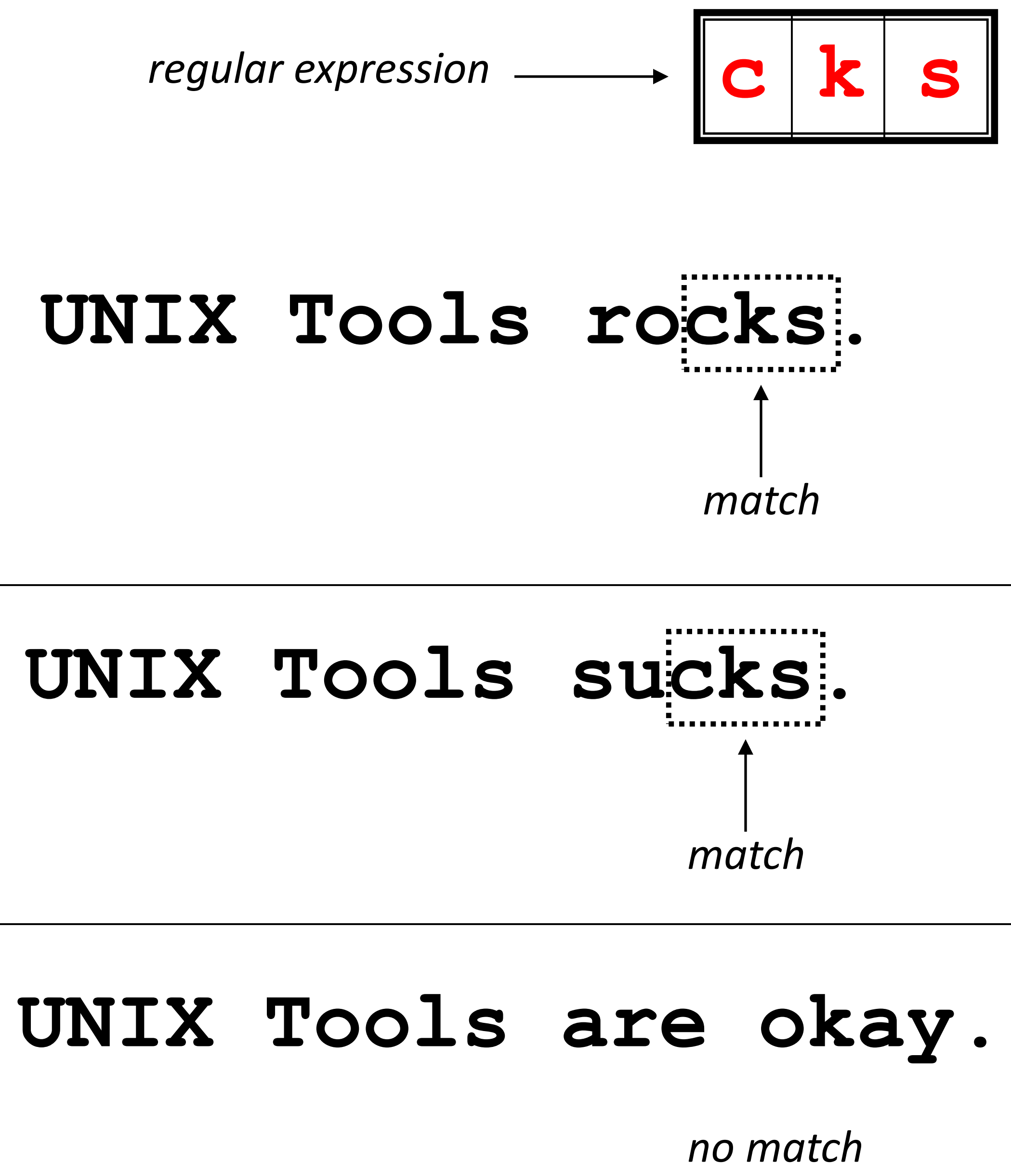
```
[pscott17@login4 test]$ cd ~/Intro2Unix
[pscott17@login4 test]$ grep "chr2" hg19.gtf > chr2.txt
[pscott17@login4 test]$ tail -n 1 chr2.txt
chr21  hg19_knownGene   CDS 33066517  33066602  0.000000
   gene_id "uc002ypd.2"; transcript_id "uc002ypd.2";
```

`Gtf file:` The Gene transfer **format** (**GTF**) is a **file format** used to hold information about gene structure. It is a tab-delimited text **format** based on the general feature **format** (GFF), but contains some additional conventions specific to gene information. (https://en.wikipedia.org/wiki/Gene_transfer_format)

Regular Expression

RegEx
Regular Expression
/h[a4@](([c<]((k)|(\|<)))|((k)|(\|<))|(x))\s+\
((d)|([t\+]h))[3ea4@]\s+p[l1][a4@]n[3e][t\+]/i
(C)2006 FTS Conventures - www.ftsconventures.com

- A **regular expression** is a string that can be used to describe several sequences of characters.



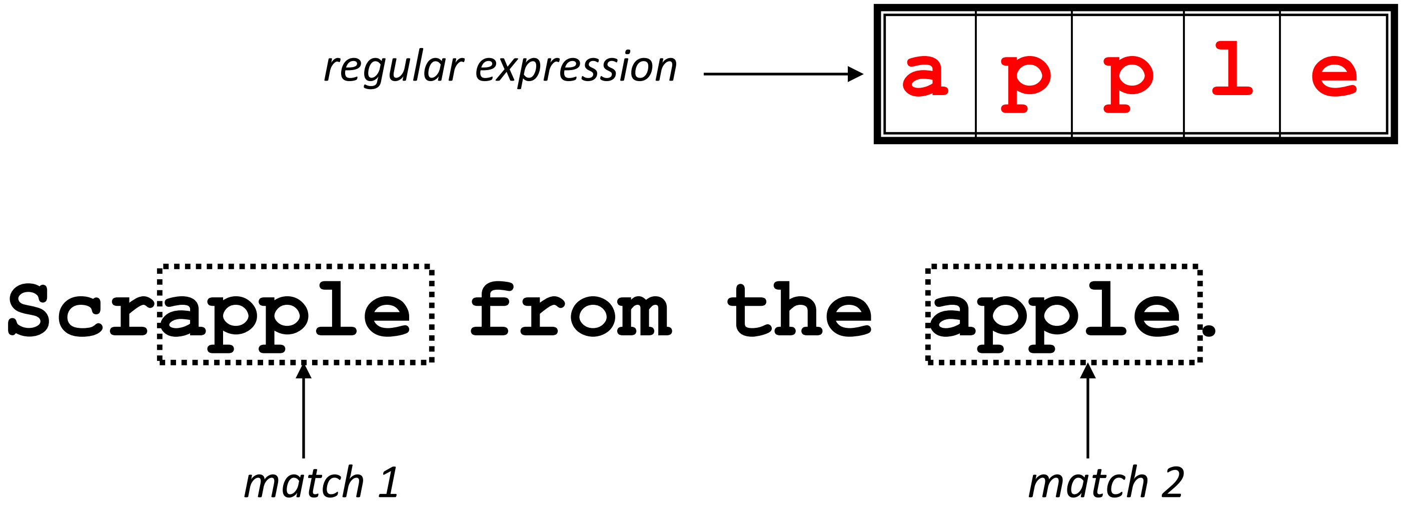
```
UNIX Tools rocks.

UNIX Tools sucks.

UNIX Tools are okay.
```

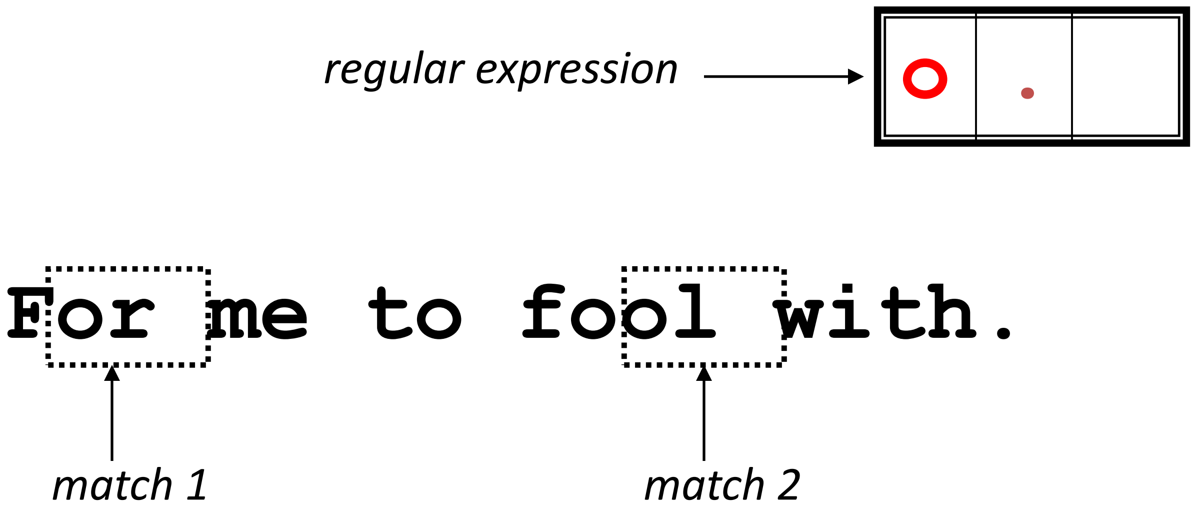
Regular Expressions

- A regular expression can match a string in more than one place.



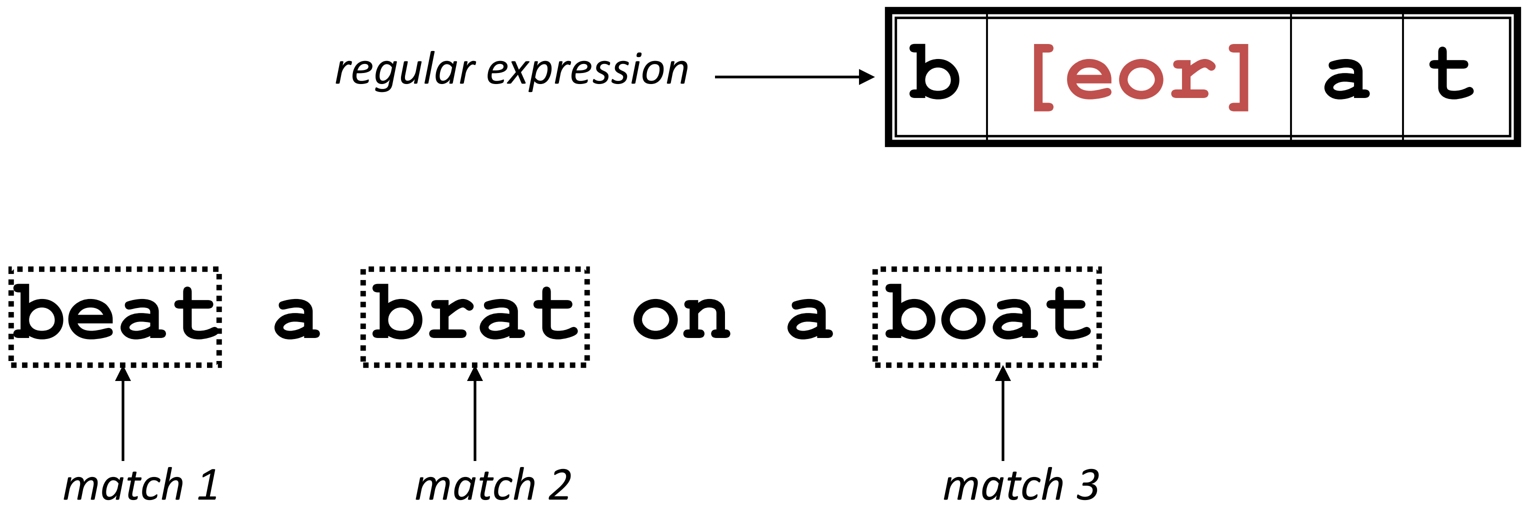
Regular Expressions

- The . regular expression can be used to match any character.



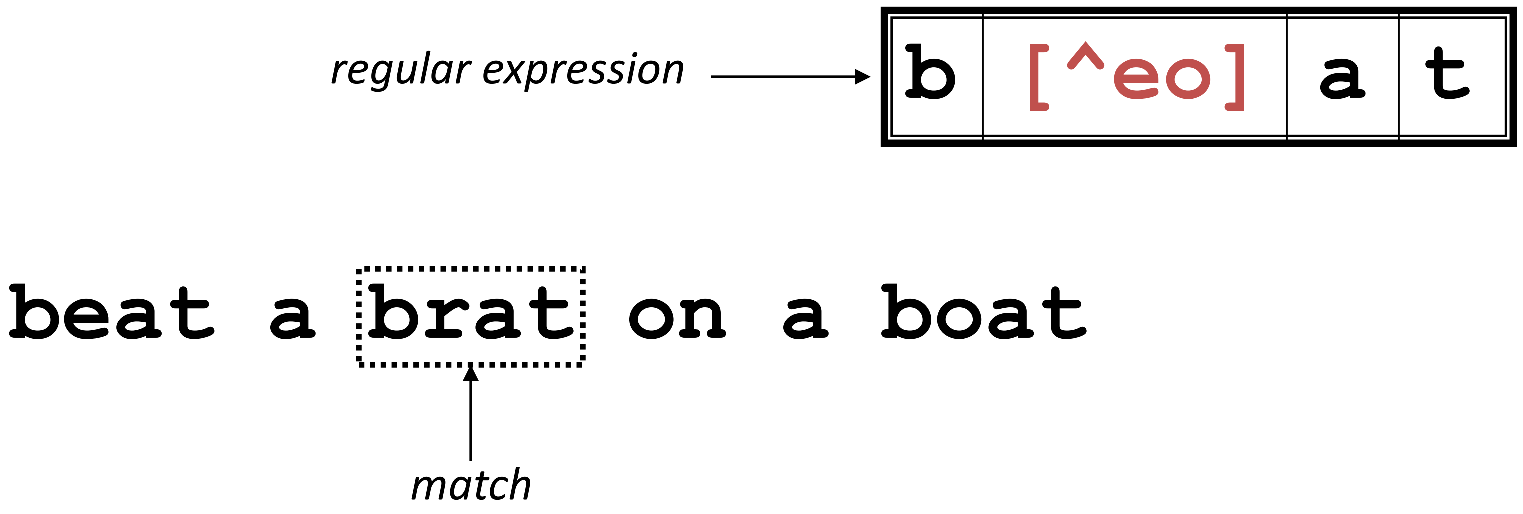
Character Classes

- Character classes [] can be used to match any specific set of characters.



Negated Character Classes

- Character classes can be negated with the **[^]** syntax.



Let’s practice!

```
[pscott17@login4 intro2unix]$ grep "boat" regex.txt
[pscott17@login4 intro2unix]$ grep "b[eor]at" regex.txt
[pscott17@login4 intro2unix]$ grep "b.at” regex.txt
[pscott17@login4 intro2unix]$ grep "b[^eor]at" regex.txt
[pscott17@login4 intro2unix]$ grep "b[^eor]" regex.txt
```

More About Character Classes

  - [aeiou] will match any of the characters a, e, i, o, or u
  - [kK]orn will match korn or Korn

- Ranges can also be specified in character classes
  - [1-9] is the same as [123456789]
  - [abcde] is equivalent to [a-e]
  - You can also combine multiple ranges
    - [abcde123456789] is equivalent to [a-e1-9]
  - Note that the - character has a special meaning in a character class ***but only*** if it is used within a range, [-123] would match the characters -, 1, 2, or 3

Alphanumeric characters

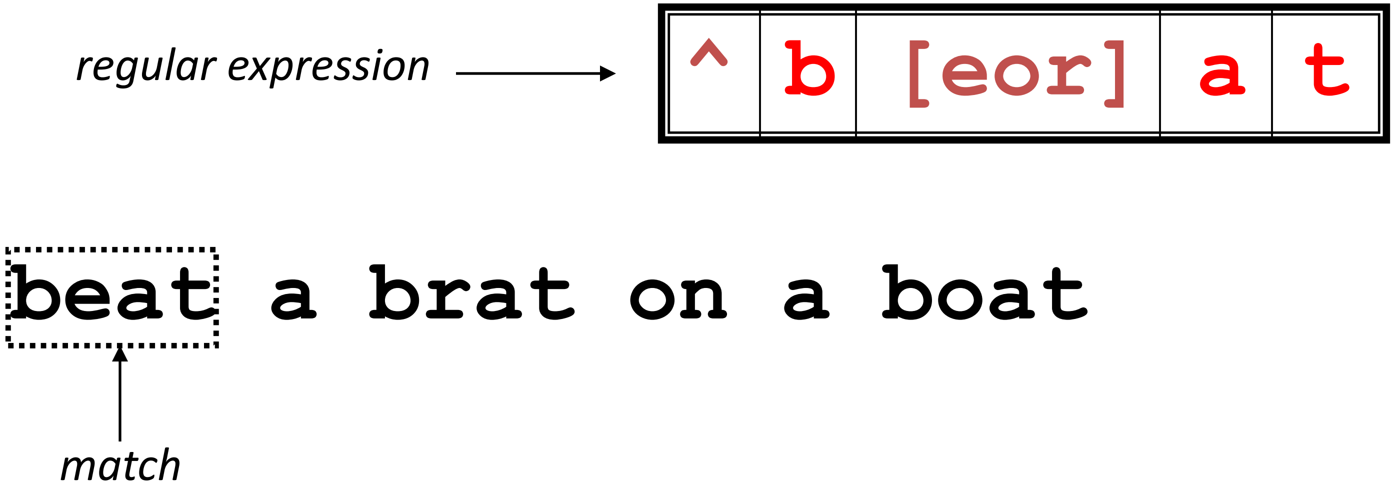
- `Alphabetic characters`
  - `[a-zA-Z]`
  - **`[[:alpha:]]`**
- `Digits`
  - **`[0-9]`**
  - `[[:digit:]]`
- `Alphanumeric characters`
  - `[a-zA-Z0-9]`
  - **`[[:alnum:]]`**

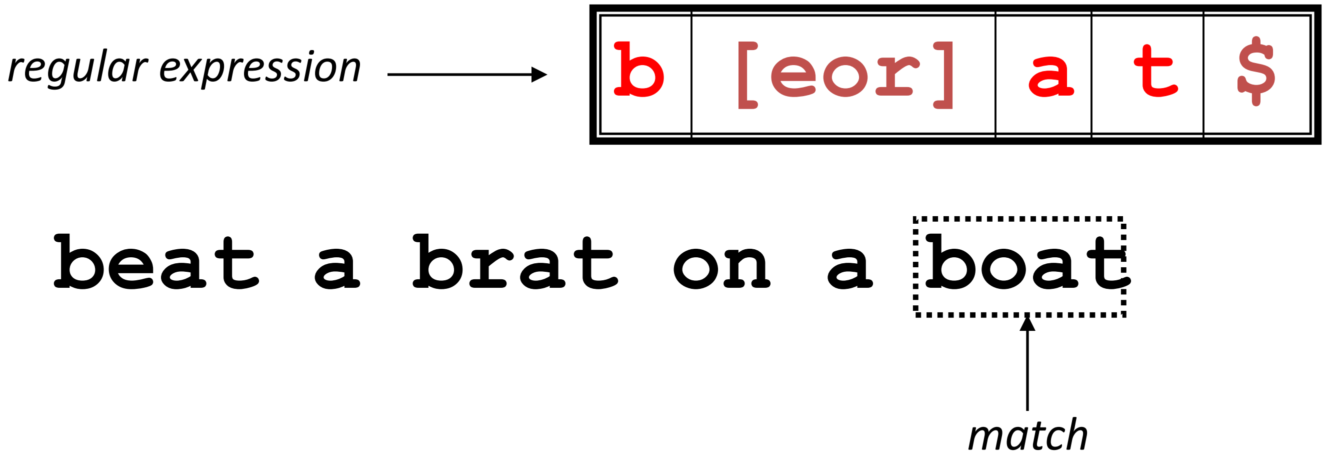
Anchors

- Anchors are used to match at the beginning or end of a line (or both).

^ means beginning of the line

$ means end of the line


regular expression
^ b [eor] a t
beat a brat on a boat
match


regular expression
b [eor] a t $
beat a brat on a boat
match

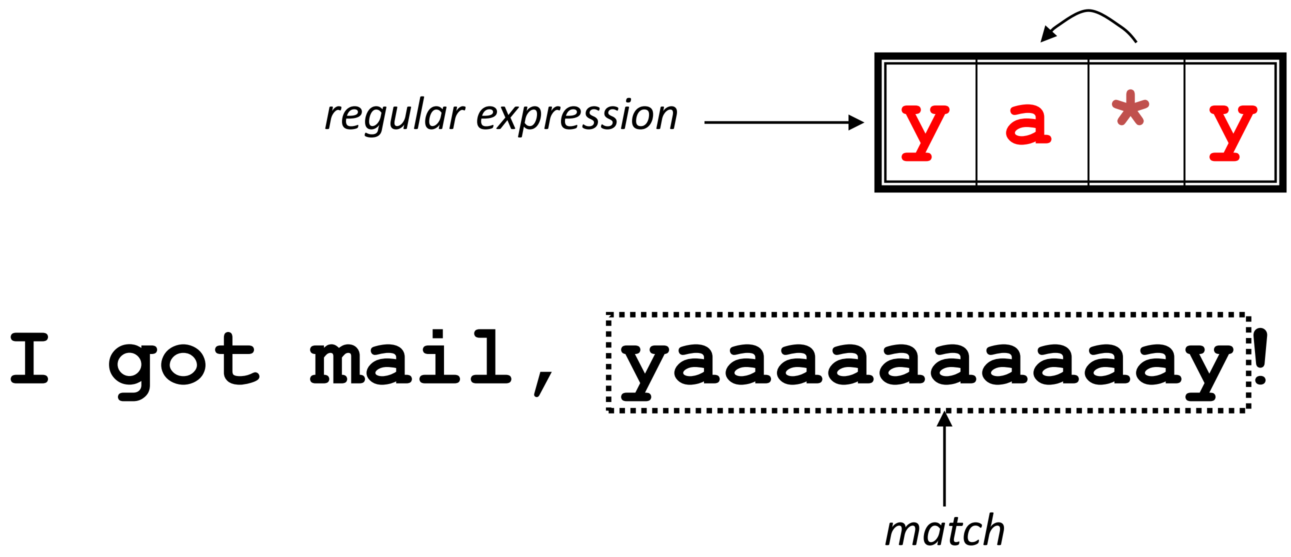
`^word$` `^$`

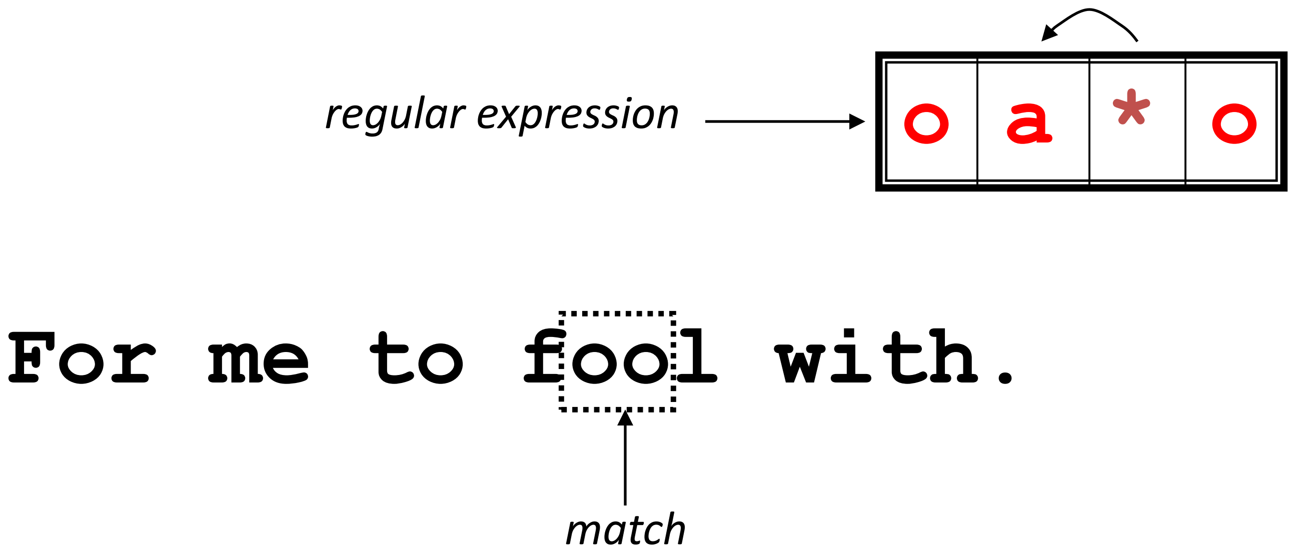
Let’s practice!

```
grep "[Aa]1" regex2.txt
grep "^[Aa]1" regex2.txt
grep "[Aa][0-9]$” regex2.txt
grep  "[0-9]" regex2.txt
grep "[[:alnum:]]" regex2.txt
grep  "[[:alpha:]]" regex2.txt
```

Repetition operators

- The * (asterisk) matches the zero or more occurrences of the **preceding** character


regular expression
y a * y
I got mail, yaaaaaaaaaay!
match


regular expression
o a * o
For me to fool with.
match

.*

Special characters

- **`\s`** `space`
- **`\t`** `tab`
- **`\s+`** `many spaces`
- **`\t\t`** `two adjacent tabs`

Lines corresponding to chr2

```
[pscott17@login4 test]$ grep "chr2" hg19.gtf > chr2.txt
[pscott17@login4 test]$ tail -n 1 chr2.txt
chr21  hg19_knownGene   CDS 33066517  33066602  0.000000
   gene_id "uc002ypd.2"; transcript_id "uc002ypd.2";
```

Lines corresponding to chr2

```
grep "chr2\s" hg19.gtf > chr2.gtf
```

Or more specific:

```
grep “^chr2\s" hg19.gtf > chr2.gtf
```

Repetition operators

*****	Zero or more...
?	Zero or one... (i.e. optional element)
+	One or more...
{x}	x instance of...
{x,y}	between x and y instances of...
{x,}	at least x instances of...
r1|r2	regular expressions r1 or r2

-E

```
grep -E <pattern> <filename>
```

Let’s practice!

```
grep -E "a1|b1" regex2.txt
```

Alternative

Let’s practice!

```
grep -E "a1|b1" regex2.txt
grep  "[ab]1" regex2.txt
```

Alternative

Repetition operators

- If you want to group part of an expression so that ***** or **{ }** applies to more than just the previous character, use **( )** notation
- Subexpresssions are treated like a single character
  - **a*** matches 0 or more occurrences of **a**
  - **abc*** matches **ab**, **abc**, **abcc**, **abccc**, …
  - **(abc)*** matches **abc**, **abcabc**, **abcabcabc**, …
  - **(abc){2,3}** matches **abcabc** or **abcabcabc**

Let’s practice!

```
grep -E "a+" regex2.txt
grep -E "a{3}" regex2.txt
grep -E "a{2,3}" regex2.txt
grep -E "a{2}" regex2.txt
grep -E "(abc)*" regex2.txt
grep -E "(abc)+" regex2.txt
grep -E "(abc){2}" regex2.txt
grep -E "[[:alpha:]]{3}" regex2.txt
grep -E "[[:alpha:]][0-9]{2}" regex2.txt
grep -E "([[:alpha:]][0-9]){2}" regex2.txt
grep -E "[[:alpha:]][0-9]\sa" regex2.txt
```

?

- `grep -E "[0-9]{3}[-]{0,1}[0-9]{3}[-]{0,1}[0-9]{4}" f.txt`

sed : a "**s**tream **ed**itor"

- A non-interactive text editor
- Routine editing tasks
  - find, replace, delete, append, insert
- Input text flows through the program, is modified, and is directed to standard output.

```
sed [options] commands [file-to-edit]
```

Why use **sed**?

- Sed is designed to be especially useful in three cases:
  - files are too large for interactive editing
  - editing is too complicated for regular text editors
  - multiple editing in one pass

sed : Substitute command **s**

sed '**s**/old_word/new_word/’ [file-to-edit]

```
To bee, or not to bee
```

```
    sed 's/bee/be/‘  tobe.txt
```

To be, or not to **bee**

sed : **g** - Global replacement

- Normally, substitutions apply to only the first match in the string.

```
To bee, or not to bee
```

- To apply the substitution to **all** matches in the string use “**g**” options

```
sed 's/bee/be/g‘  tobe.txt
```

Edit matched text

- Put parentheses around the matched text:

```
sed -E 's/<pattern>/(&)/' annoying.txt
```

Let’s practice!

```
                            To bee, or not to bee
less tobe.txt                 ↓
sed 's/bee/be/' tobe.txt
To be, or not to bee
sed 's/bee/be/g' tobe.txt
To be, or not to be
```

```
sed  ’s/seven/nine/g’ file_sed.txt | sed ’s/nine/two/g’
sed ’s/a/o/g’ file_sed.txt
sed 's/^and/or/’ file_sed.txt
sed 's/s..../xxxxx/g’ file_sed.txt
sed 's/ago$/ago!/’ file_sed.txt

sed 's/[12]/3/g' regex2.txt
sed 's/[[:alpha:]]/B/g' regex2.txt
sed -E 's/[[:alnum:]]{2}/(&)/g' regex2.txt
```

Don’t read and write the same file!

- sed 's/seven/nine/g' **sed_file.txt >sed_file.txt**

Redirections are done by the shell, before the command runs. This means that the shell is told to write the file before **sed** gets a chance to read it. There is no way around this if you are using shell redirection.

Delete lines with sed

- Remove the 3rd line:
  - `sed '3d' fileName.txt`
- Remove the line containing the string "awk":
  - `sed '/awk/d' filename.txt`
- Remove the last line:
  - `sed '$d' filename.txt`

Let’s practice!

```
sed '3d’ regex2.txt
sed '/a/d' regex2.txt
sed '/[0-9]/d' regex2.txt
sed '$d' regex2.txt
```

Summary

- file permissions
- cat
- wc
- >, >>, <
- pipeline
- ln –s
- grep
- regex

Set up the alias for Mac OS/linux

- Go to home directory ON YOUR COMPUTER: `cd ~`
- Open file .bash_profile: `nano .bash_profile`
- Add in the end of the file:

-
  ```
  alias hoffman='pscott17@hoffman2.idre.ucla.edu'
  ```

- Restart the session

Run from the local session of the terminal. To open a local session : **Control-T**

Set up the alias for Cygwin

- Go to home directory :
  - `This PC / Windows (C:) / Cygwin64 / etc`
- Open file **ssh_config** in text editor
- Add in the end of the file:

-
  ```
  Host hoffman
   HostName hoffman2.idre.ucla.edu
   Port 22
   User username
  ```
- Restart the session

Run from the local session of the terminal. To open a local session : **Control-T**