

# Workshop 1: Introduction to UNIX command-line

Serghei Mangul, PhD | [smangul@ucla.edu](mailto:smangul@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

# Copy the working materials



```
[serghei@login2 code] $ git clone  
https://github.com/QCB-  
Collaboratory/W1.UNIX.command.line.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/QCB-Collaboratory/W1.UNIX.command.line.git>**

The screenshot shows a GitHub repository page for 'QCB-Collaboratory/W1.UNIX.command.line'. At the top, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a prominent green 'Clone or download' button. Below this, a list of files is shown: 'figures', 'workshop.materials', 'workshop.slides', and 'README.md'. On the right side, there are two cloning options: 'Clone with HTTPS' (selected) and 'Use SSH', each with a corresponding URL input field containing 'https://github.com/QCB-Collaboratory/W1.UNIX.command.line.git'. Below the URLs are 'Open in Desktop' and 'Download ZIP' buttons.

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

# Working materials

- cd W1.UNIX.command.line/workshop.materials/
  - ls
- 
- emp.txt regex\_sort.txt toy.reads.fastq toy.ref.fasta.pac
  - file\_sed.txt regex.txt toy.ref.fasta toy.ref.fasta.sa
  - hg19.gtf reproduce.sh toy.ref.fasta.amb
  - numbers.txt sales.txt toy.ref.fasta.ann
  - regex2.txt sed\_file.txt toy.ref.fasta.bwt

# Relative vs. absolute path

- A file or a directory can be referred to by
  - Relative path
    - `../test.txt` **if you are at**  
`/u/home/s/serghei/test/new/`
  - Absolute path
    - `/u/home/s/serghei/test/test.txt`

```
[serghei@login2 test]$ less ./test.txt
[serghei@login2 test]$ ls -l ../
[serghei@login2 test]$ ls -l ~/
```

Absolute

**245 Highland Ave, Manhattan  
Beach, California 90266**

Relative



# 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 serghei eeskin    72 Mar 11 14:22 large.txt
-rw-r--r-- 1 serghei eeskin   263 Mar 11 15:18 new.tar
-rw-r--r-- 1 serghei eeskin   13 Mar 11 15:27 test.txt
drwxr-xr-x 2 serghei eeskin 4096 Mar 11 15:36
dfgdfgdfgdfgdfgdfgdfgdf
```

Type

User (you)



# File Permissions

```
-rw-r--r-- 1 serghei eeskin    72 Mar 11 14:22 large.txt
-rw-r--r-- 1 serghei eeskin   263 Mar 11 15:18 new.tar
-rw-r--r-- 1 serghei eeskin   13 Mar 11 15:27 test.txt
drwxr-xr-x 2 serghei eeskin 4096 Mar 11 15:36
dfgdfgdfgdfgdfgdfgdfgdf
```

Type

Group

```
-rw-r--r-- 1 serghei eeskin    72 Mar 11 14:22 large.txt
```

# File Permissions

```
-rw-r--r-- 1 serghei eeskin    72 Mar 11 14:22 large.txt
-rw-r--r-- 1 serghei eeskin   263 Mar 11 15:18 new.tar
-rw-r--r-- 1 serghei eeskin   13 Mar 11 15:27 test.txt
drwxr-xr-x 2 serghei eeskin 4096 Mar 11 15:36
dfgdfgdfgdfgdfgdfgdfgdf
```

Type

“The World”

```
-rw-r--r-- 1 serghei eeskin    72 Mar 11 14:22 large.txt
```

# Command: chmod

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

- Syntax:

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



```
[serghei@login2 test]$ ls -l
drwxr-xr-x 3 serghei eeskin 4096 Mar 11 15:23 archive
-rw-r--r-- 1 serghei eeskin    72 Mar 11 14:22 large.txt
-rw-r--r-- 1 serghei eeskin   263 Mar 11 15:18 new.tar
-rw-r--r-- 1 serghei eeskin    13 Mar 11 15:27 test.txt
[serghei@login2 test]$ chmod g+w large.txt
[serghei@login2 test]$ ls -l
drwxr-xr-x 3 serghei eeskin 4096 Mar 11 15:23 archive
-rw-rw-r-- 1 serghei eeskin    72 Mar 11 14:22 large.txt
-rw-r--r-- 1 serghei eeskin   263 Mar 11 15:18 new.tar
-rw-r--r-- 1 serghei eeskin    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!

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

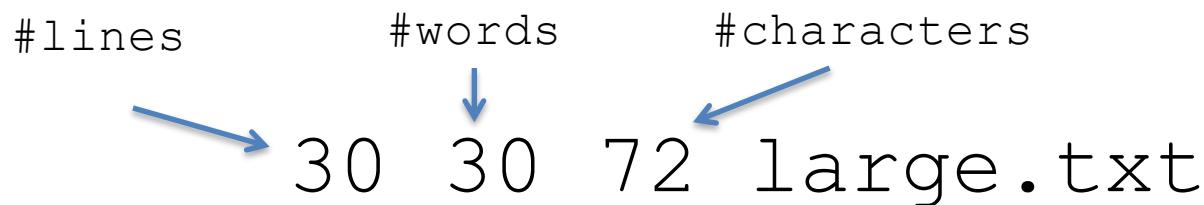
- To count the characters, words, and lines in a file use **wc**

```
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
30	30	72

large.txt





# Let's practice!

```
[serghei@login2 ~] $ wc test.txt
```

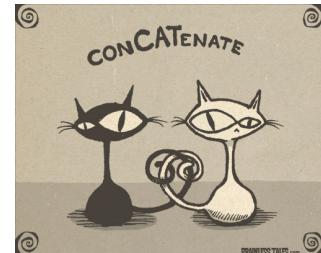
```
1 3 15 test.txt
```

```
[serghei@login2 ~] $ wc -l test.txt
```

```
1 test.txt
```

```
[serghei@login2 ~] $ ls | wc -l
```

```
5
```



# Command : cat

- Concatenate files together and displayed in the terminal.

```
cat <file1> <file2> ...
```



```
[serghei@login2 test]$ cat large.txt test.txt | wc -l  
21  
[serghei@login2 test]$ cat large.txt test.txt >all.txt
```

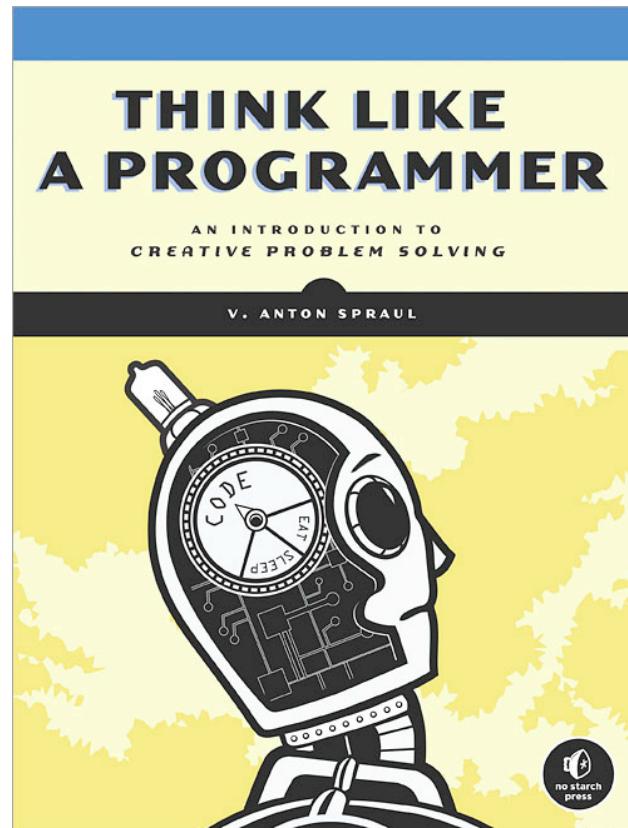
Directory

# 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
- **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!

```
[serghei@login4 test]$ grep "1" large.txt
```

```
1
```

```
10
```

```
...
```

```
19
```

```
[serghei@login4 test]$ grep -n "1" large.txt
```

```
1:1
```

```
10:10
```

```
...
```

```
19:19
```

```
[serghei@login4 test]$ grep -c "1" large.txt
```

```
13
```

```
[serghei@login4 test]$ grep -l "1" large.txt test.txt
```

```
large.txt
```

```
[serghei@login4 test]$ grep "1" large.txt test.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

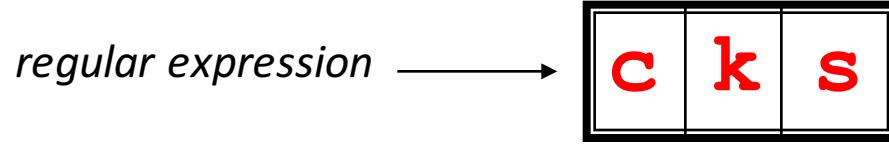
```
[serghei@login4 test]$ grep "chr2" hg19.gtf > chr2.txt
[serghei@login4 test]$ tail -n 1 chr2.txt
chr21 hg19_knownGene CDS 33066517 33066602 0.000000
    gene_id "uc002YPD.2"; transcript_id "uc002YPD.2";
```

# Regular Expression



Regular Expression  
/h[ə4@](((c<)((k|l|-))|((k)|l|<)|(x))s+\\  
((d)|((t|+|h))|3ea4@)|s+p[1]|a4@|n|3e|(t|+)|/i  
©2006 FTS Conventions - www.fscconventions.com

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



**UNIX Tools rocks.**

↑  
*match*

---

**UNIX Tools sucks.**

↑  
*match*

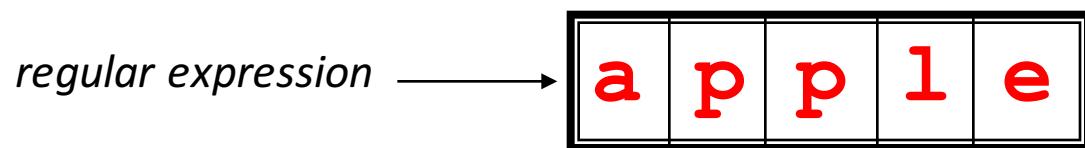
---

**UNIX Tools is okay.**

*no match*

# Regular Expressions

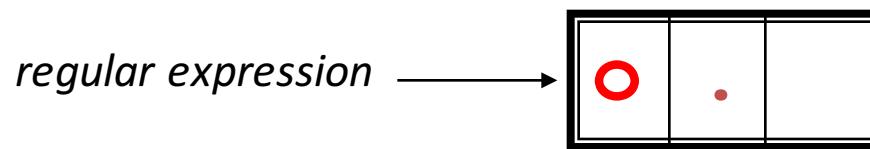
- A regular expression can match a string in more than one place.



Scrabble from the apple.

# Regular Expressions

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



For me to poop on.

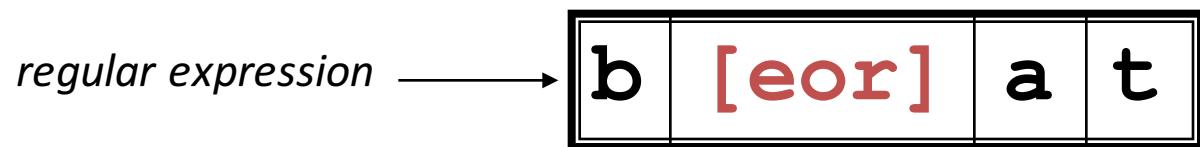
*match 1*                    *match 2*



The word "For" is enclosed in a dotted rectangle, with an arrow below it labeled "match 1". The word "on" is also enclosed in a dotted rectangle, with an arrow below it labeled "match 2".

# Character Classes

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

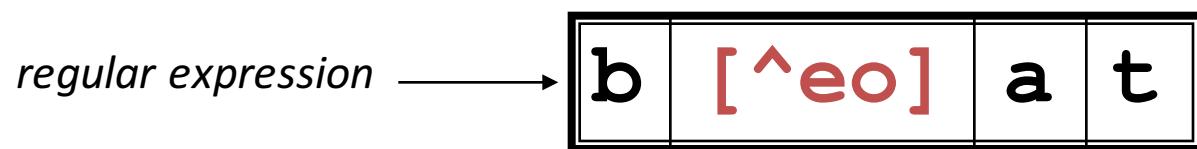


beat a brat on a boat

match 1              match 2              match 3

# Negated Character Classes

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



beat a **brat** on a boat

↑  
*match*



# Let's practice!

```
[serghei@login4 test]$ grep "b[eor]at" regex.txt
[serghei@login4 test]$ grep "b.at" regex.txt
[serghei@login4 test]$ grep "b[^eor]at" regex.txt
[serghei@login4 test]$ 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).

^K means beginning of the line

\$ means end of the line

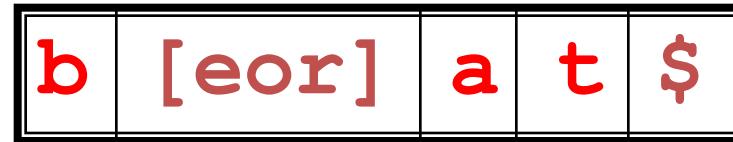
*regular expression*



**beat** a brat on a boat

*match*

*regular expression*



beat a brat on a **boat**

*match*

^word\$

^\$

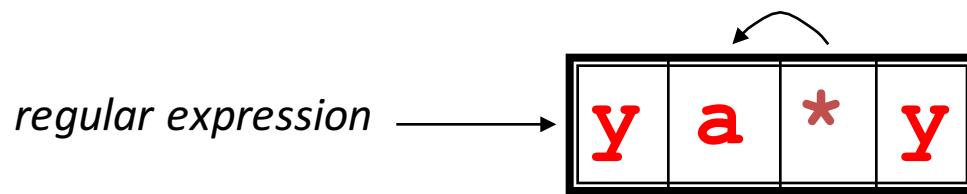
# Let's practice!



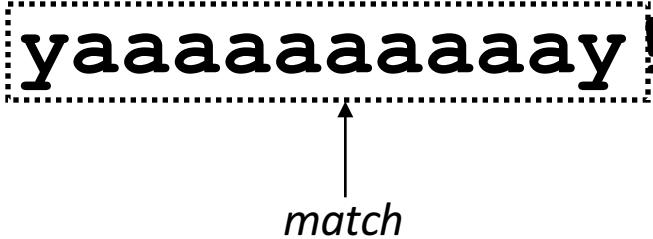
```
grep "[Aa]1" regex2.txt
grep "^[Aa]1" regex2.txt
grep "[Aa][0-9]\$" regex2.txt
grep "[0-9]" regex2.txt
grep "[[:alnum:]]" regex2.txt
grep "[[:alpha:]]" regex2.txt
```

# Repetition operators

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

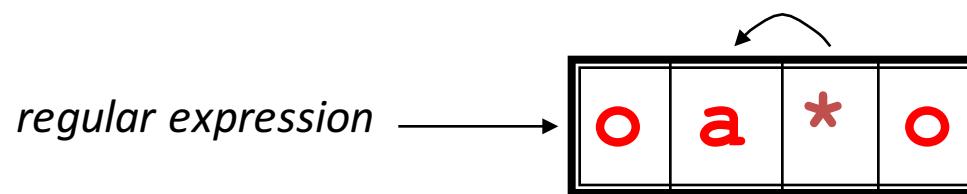


I got mail, **yaaaaaaaaay!**

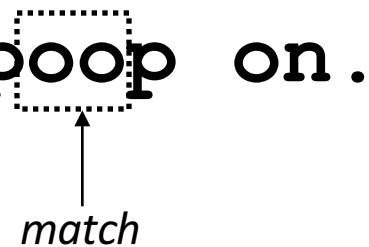


match

---



For me to **poop** on.



match

---

• \*

# Repetition operators

- \* Zero or more...
- ? Zero or one... (i.e. optional element)
- + One or more...
- E** {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

grep -E <pattern> <filename>

# Special characters

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



# Let's practice!

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



# Lines corresponding to chr2

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

# Repetition operators

- If you want to group part of an expression so that  $*$  or  $\{ \}$  applies to more than just the previous character, use  $( )$  notation
- Subexpressions 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  $abcababc$



# 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}[\s\-\s]{0,1}[0-9]{3}[\s\-\s]{0,1}[0-9]{4}" f.txt

# sed : a “stream editor”



- 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/' file_sed.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  
be ↑ ? ↑

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

```
sed 's/bee/be/g' file_sed.txt
```

# Edit matched text

- Put parentheses around the matched text:

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



# Let's practice!

To bee, or not to bee

```
vi 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$/!/' 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 truncate 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

- Go to home directory : cd
- Open file .bash\_profile: vi .bash\_profile
- Add in the end of the file:  

```
alias hoffman2='ssh serghei@hoffman2.idre.ucla.edu'
```
- Restart the session



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