Lab Assignment – Terminal

CS380

In this lab you will get an introduction to the Terminal command line interface provided in Ubuntu. This is called a shell.

This week you will have a general introduction of this topic. Next week you will look at more details and complete an assignment on the topic.

Lets get started!

Shell

Command-line interface provided by Unix and Mac OS X is called a shell a shell:

- prompts user for commands
- interprets user commands
- passes them onto the rest of the operating system which is hidden from the user

How do you access a shell ?

- if you have an account on a machine running Unix or Linux , just log in.

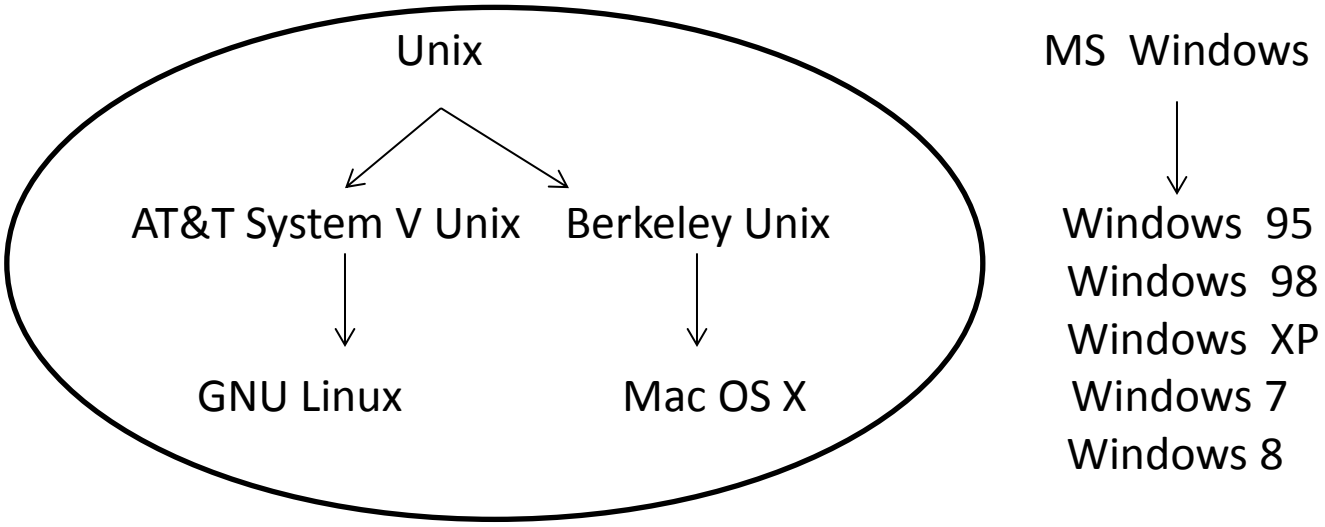
A default shell will be running.

- if you are using a Mac or Ubuntu, run the Terminal app. 

A default shell will be running.

Aside: A shell script is a sequence of shell commands written in an executable script file. Executing this file instructs the shell to execute all commands in the order of their appearance in the script file.

Examples of Operating Systems



- Even though there are differences between the various Unix operating systems, for the most part, we are going to ignore those differences, and just refer to "Unix" operating systems because the principles are largely the same.

- There are also many different Unix shells that are more alike than different:
  - sh (original Unix shell, by Stephen Bourne) `/bin/sh`
  - ksh (similar to sh, by David Korn) `/bin/ksh`
  - bash (Bourne again shell, part of GNU project) `/bin/bash`
  - csh (part of Berkely Unix, intended to by C-like, by Bill Joy) `/bin/csh`
  - tcsh (based on and compatible with csh) `/bin/tcsh`

`echo $SHELL` `(type this command to see what shell is on your computer)`

Unix Philosophy

- provide small programs that do one thing well
  and
  provide mechanisms for joining programs together

- "silence is golden"
  when a program has nothing to say, it shouldn't say anything

- users are very intelligent and do what they intend to do

Introduction to Terminal use

See this tutorial for an introduction to using Terminal:

https://www.youtube.com/watch?v=IVquJh3DXUA

The following pages will help you gain a deeper understanding of Terminal use. You should try out operations in Terminal & consult the web for further details as required.

Examples of Tasks for Command-Line Interface

data management:

- two types of administrative data – millions of observations of each type
- need to standardize addresses for merging (or fuzzy matching)

file management

- check number of lines in large file downloaded from the web

file management:

- split huge files into subsets that are small enough to be read into memory

basic web scraping

- list of names and dates of OPR computing workshops

basic web scraping

- list of UN countries and dates they became members

Command Execution Cycle and Command Format

1. Shell prompts user
2. User inputs or recalls command ending with <CR>
3. Shell executes command

```
$ command  [options]  [arguments]
```

command
- first word on line
- name of program

options
- usually begin with -
- modify command behavior

arguments
- "object" to be "acted on" by command
- often directory name, file name, or character string

use `man` command for options & arguments of each command

use `PS1="$ "` to change prompt string
(try typing this command to change the prompt string on your PC!)

```
$ date

$ who

$ who  -q

$ cal  2014

$ cal 8 2014

$ pwd

$ ls

$ mkdir  unix

$ cd unix

$ pwd

$ls
```

Using Command History

commands are saved and are available to recall

to re-execute a previously entered command:

step 1. press  to scroll through previously entered commands
step 2. press <CR> to execute a recalled command

OR

to re-execute a previously entered command:

```
$ history
$ !<command number>
```

Files

- Displaying File Contents
- File Management Commands
- File Access and Permission
- Redirecting Standard Output to a File
- File Name Generation Characters

Files

file names:

- ✓ should not contain spaces or slashes
- ✓ should not start with + or –
- ✓ best to avoid special characters other than _ and .
- ✓ files with names that start with . will not appear in output of ls command

created by:

- copying an existing file
- using output redirection
- executing some Unix program or other application
- using a text editor
- downloading from the internet

(try out the commands on the following slides; further details can be searched for on web if required)

Displaying File Contents

```
$ wc wdata

$ cat wdata

$ head wdata

$ head -1 wdata

$ tail wdata

$ tail -2 wdata

$ more wdata
```

(These commands assume you have a file called wdata saved in your working directory)

You should create a file called wdata using a text editor & put some sample content in it

File Commands

```
$ cp  wdata  wdata.old

$ mv  wdata.old  wdata.save

$ cp  wdata  wdata_orig

$ cp  wdata  wdata_fromweb

$ rm  wdata_orig   wdata_fromweb

$ diff  wdata  wdata.save
```

(These commands also assume you have a file called wdata saved in your working directory)

Redirecting Standard Output

most commands display output on terminal screen

```
$ date
```

command output can be redirected to a file

```
$ date > date.save
$ cat date.save
```

*** note: output redirection using > overwrites file contents if file already exists

```
$ date > date.save
$ cat date.save
```

use >> to append output to any existing file contents (rather than overwrite file)

```
$ date >> date.save
$ cat date.save
```

File Name Generation Characters

shell can automatically put file names on a command line if user uses file name generation characters

(below shows examples. Try some of these out considering the files you have in your working directory, e.g. your date.save file. What happens if you try the command 'ls d*' for example?)

? any single character

```
$ cat s?
```

* any number of any characters (including 0)

```
$ ls b*
```

```
$ ls *.R
```

```
$ wc -l *.do
```

```
$ ls *.dta
```

```
$ ls check_*.do
```

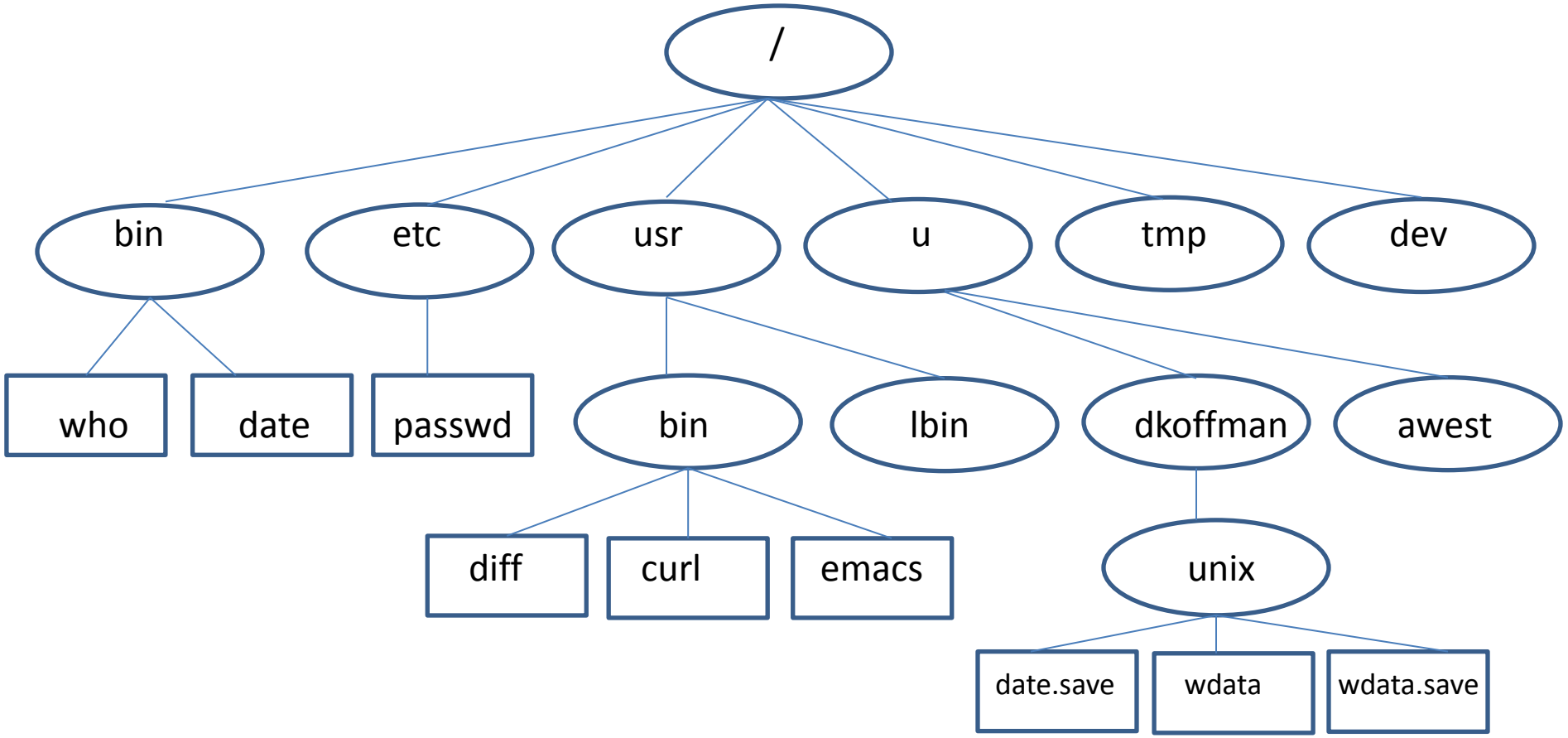
[...] any one of a group of characters

```
$ rm s[4-7]
```

Directories

- Directory Tree
- Pathnames: Absolute and Relative
- Copying, Moving and Removing Files & Directories

Directory Tree



`pwd` shows you where you are (present working directory)

`cd` makes your "home" (login) directory your current directory

Changing Directory

absolute pathnames

```
$ pwd
$ cd /etc
$ cat passwd
$ cd /bin
$ ls e*
$ ls f*
$ cd /usr/bin
$ ls e*
$ ls f*
$ cd /u/dkoffman
$ cd /u/dkoffman/unix
```

relative pathnames

```
$ pwd
$ cd ../../../etc
$ cat passwd
$ cd ../bin
$ ls e*
$ ls f*
$ cd ../usr/bin
$ ls e*
$ ls f*
$ cd
$ cd  unix
```

.. refers to the parent directory

Accessing Files

absolute pathnames

```
$ pwd

$ cat /etc/passwd

$ ls /bin/e*
$ ls /bin/f*

$ ls /usr/bin/e*
$ ls f*
$ pwd
```

relative pathnames

```
$ pwd

$ cat ../../../etc/passwd

$ ls ../../../bin/e*
$ ls ../../../bin/f*

$ ls ../../../usr/bin/e*
$ ls ../../../usr/bin/f*
$ pwd
```

.. refers to the parent directory

Copying Files

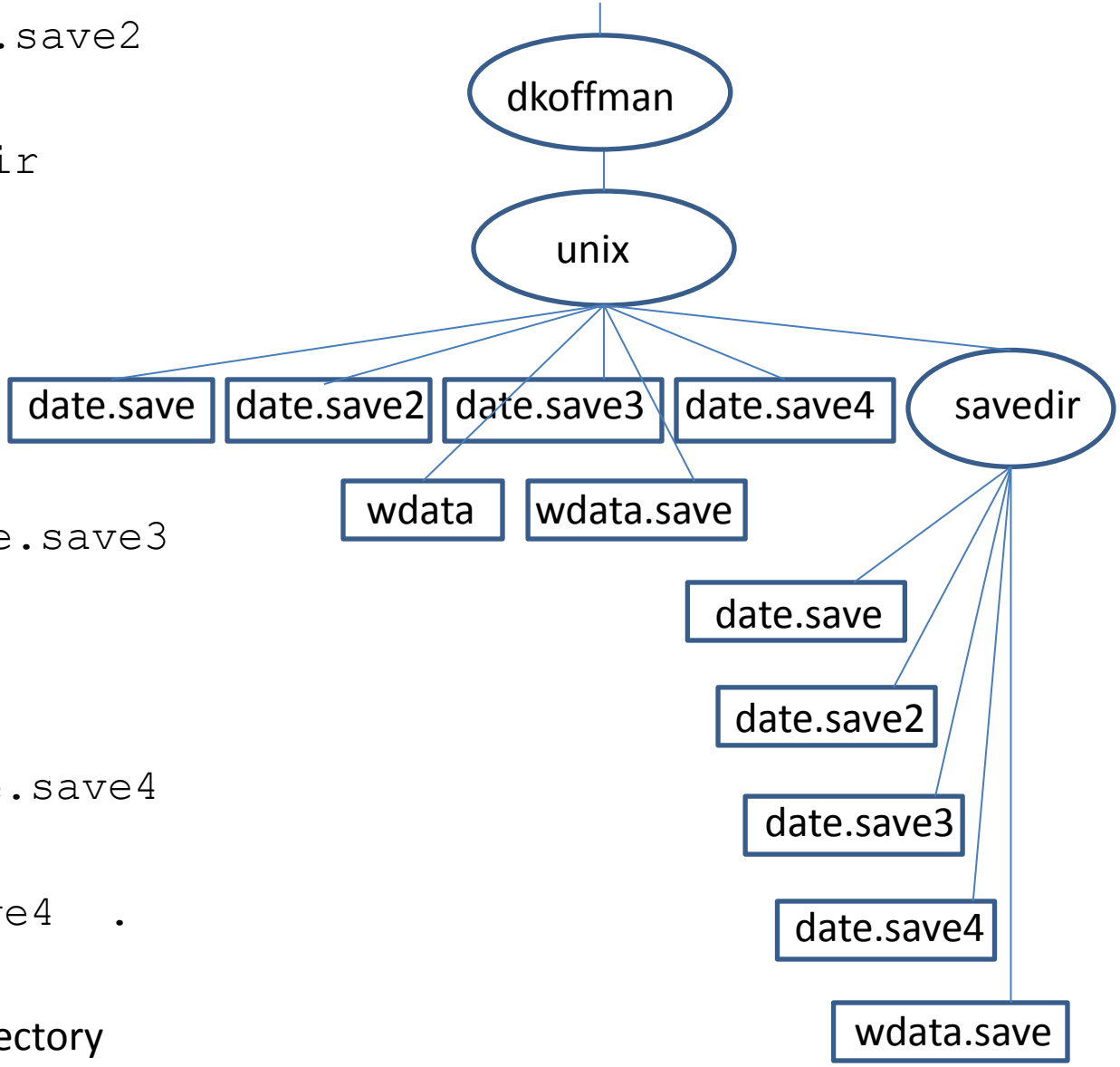
```
$ cp date.save  date.save2
$ mkdir   savedir
$ cp  *.save*  savedir
```

list of files

```
$ cd savedir
$ ls
$ cp date.save2  date.save3
$ cp date.save3 ..

$ ls  ..
$ cp date.save2  date.save4
$ cd ..
$ cp savedir/date.save4  .
```

. refers to the current directory



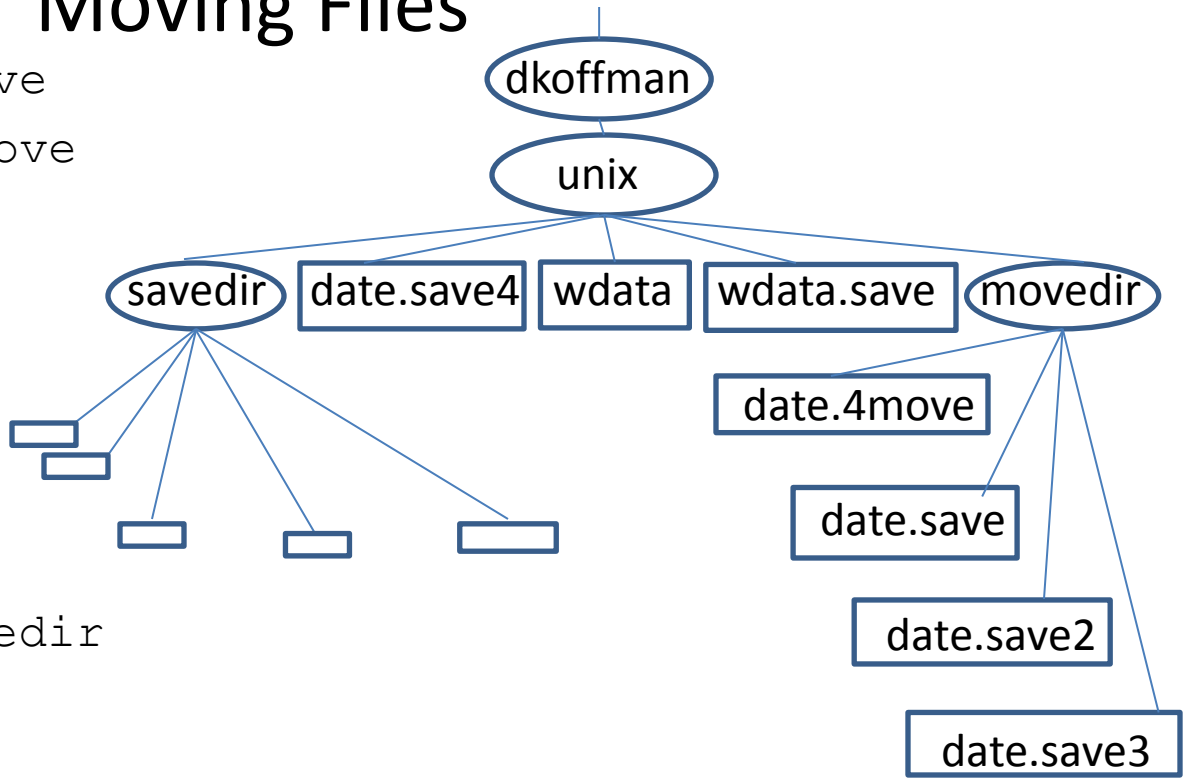
Moving Files

```
$ cp date.save  date4move
$ mv date4move  date.4move
$ ls
$ mkdir movedir
$ mv date.4move movedir
$ ls
$ ls movedir

$ mv date.save[23]  movedir
```

list of files

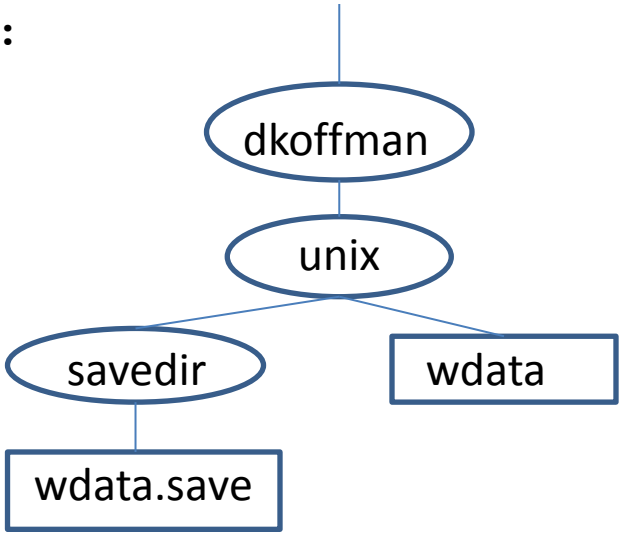
```
$ ls
$ cd movedir
$ ls
$ mv  ../date.save   .
$ ls
$ cd ..
```



Removing Files and Directories

```
$ cd
$ cd unix
$ rm date.save4 wdata.save
$ rmdir movedir
rmdir: failed to remove ‘movedir’:
Directory not empty
$ ls movedir
$ rm movedir/*   # BE CAREFUL!
$ rmdir movedir

$ rm savedir/date*
$ ls savedir
$ ls
```



Commands

(This 'Commands' section is more complicated.
More 'Commands' are provided here for those who would like to take a deeper look at command line use. Particularly recommended here are the last three pages of the document.)

- Review of Commands
- More Commands
- Sequential Execution
- Command Grouping
- Pipelines
- Foreground/Background Command Execution

Review of Commands

date

who

cal

pwd

ls

mkdir

cd

history

curl

wget

gunzip

cat

head

tail

more

cp

mv

rm

diff

chmod

rmdir

More Commands

```
$ tail -40 wdata

$ sort wdata

$ tail -40 wdata

$ sort wdata > wdata.sort

$ more wdata.sort


$ sort -r wdata > wdata.revsort

$ more wdata.revsort


$ wc wdata

$ wc -l wdata

$ wc -wc wdata
```

More Commands

```
$ head wdata

$ cut -d”,” -f1 wdata

$ head wdata

$ cut -d”,” -f1 wdata > wdata.countries

$ cut -c1,3-4 wdata

$ cut -d”,” -f5 wdata > wdata.le

$ paste wdata.le  wdata.countries

$ sort wdata.le > wdata.le.sort

$ uniq wdata.le.sort

$ uniq -c wdata.le.sort
```

More Commands

```
$ grep ",Oceania," wdata

$ grep ",Central America," wdata > wdata.centralamerica

$ grep pop2012 wdata

$ grep pop2012 wdata > wdata.hd

$ grep -v pop2012 wdata > wdata.clean

$ head wdata.clean

$ wc -l wdata.clean

$ grep  -n ",Oceania,"  wdata.clean

$ grep -n -i ",oceania,"  wdata.clean
```

Regular Expressions

describe a sequence of characters (pattern) to be matched

<u>basics</u>

.	(dot)	matches any single character: 1.6
[ ]	(brackets)	match any one of the enclosed characters: [aeiou] can use – (dash) to indicate at range of characters: [A-Za-z] [24-6]
[^]		match any character **except** the enclosed characters: [^Zz]
*	(asterisk)	matches zero or more of the preceding character: b* vs bb*
^	(caret)	pattern must occur at the beginning of a line (anchor): ^ABC
$	(dollar sign)	pattern must occur at the end of a line (anchor): ABC$ vs ^ABC$
\	(backslash)	turns off (escapes) the special meaning of the next character: \.*

enclose regular expressions in single quotes to stop shell from expanding special characters

Using Regular Expressions

```
$ grep stan wdata.clean

$ grep ‘^B’ wdata.clean

$ grep ‘^....,’ wdata.clean

$ grep ‘/’ wdata.clean

$ grep –i ira[qn] wdata.clean

$ grep ‘^.*,.*West’ wdata.clean

$ grep ‘4.,[A-Z]’ wdata.clean

$ grep ‘[56].,[A-Z]’ wdata.clean

$ grep ‘[67].,[A-Z]..*Americas’ wdata.clean
```

More Commands

```
$ split -l20 wdata.clean

$ ls

$ wc -l xa?

$ tail xah

$ cat xa? > wdata.clean.copy

$ wc -l wdata.clean.copy

$ tr “abcdefghijklmnopqrstuvwxyz” “ABCDEFGHIJKLMNOPQRSTUVWXYZ” < wdata

$ tr  [:lower:]  [:upper:]  < wdata.clean  > wdata.clean.uc

$ tr -d ‘:”’ < wdata.clean

$ tr -s “ ”  < wdata.clean
```

Sequential Execution

```
cmd1 arg1 arg2 ...; cmd2 arg1 arg2 ...; cmd3 arg1 arg2  ...
```

- series of commands on a single line separated by semicolons

- commands are executed left-to-right, one at a time

```
$ sort wdata.clean > wdata.clean.s;  echo SORT DONE
```

Command Grouping

```
(cmd1 arg1 agg2 ...; cmd2 arg1 arg2 ...; cmd3 arg1 arg2 ...)
```

- allows several commands to be treated as one with respect to standard output

```
$ date > log

$ who am  i  >> log

$ (
 > date
 > who am i
 > ) > log
$

$ (date; who am i) > log
```

Pipeline

```
cmd1 arg1 ... | cmd2 arg1 ... | cmd3 arg1 ...
```

- series of commands separated by |

- output of one command used as input for next command

- commands run in parallel when possible!

- avoids use of temporary files ... faster!

```
$ who | sort
```

```
$ who > tempfile
$ sort < tempfile
$ rm tempfile
```

Pipeline Examples

```
$ who | wc –l

$ ls –l | grep "^d"

$ grep Africa wdata.clean | sort

$ sort wdata.le | uniq | wc –l

$ grep Americas wdata.clean | cut –d"," –f5 | sort

$ grep Americas wdata.clean | cut –d"," –f5 | sort | uniq

$ grep Americas wdata.clean | cut –d"," –f5 | sort | uniq | wc –l

$ sort wdata.clean | tr [:lower:] [:upper:] | cut –d"," –f1

$ sort wdata.clean | cut –d"," –f1,5

$ sort wdata.clean | cut –d"," –f1,5 | tr –d '".:' | split –l20 – wdata_le_part_
```

Writing to a File And to Standard Output

tee command

- reads from standard input
- writes to a file and standard output
- very useful for saving intermediate “results” in a pipeline
- use –a option to append to a file rather than overwrite

```
$ sort wdata.le | uniq | tee wdata.le.uniq | wc -l

$ cat wdata.le.uniq

$ sort wdata.le | uniq | tee wdata.le.uniq | wc -l > le.uniq.count

$ cat le.uniq.count

$ sort wdata.clean | cut -d”,” -f1,5 | tee c.le | split -l20 - wdata_le_part_

$ cat c.le
```