Getting Your Computation Running on Princeton's Cloud

Dawn Koffman
Office of Population Research
Princeton University
May 2015

Running Your Computation on Princeton's Cloud

significantly more computing resources available compared to your laptop or desktop

but often, particularly at first, it is much easier to develop and debug locally, **and then**

- connect
- transfer files
- submit jobs

and make use of full computing power available on these remote systems

Workshop Outline

- Princeton's Cloud
- Unix/Linux/Mac OS X Philosophy and User Interface
- Files
- Directories
- Commands
- Shell Programs
- [Stream Editor: sed]

Princeton's Cloud

- What is Princeton's Cloud?
- Overview of Princeton's Remote Systems
- Obtaining Accounts
- Connecting
- Transferring Files
- Running R scripts and Stata .do Files
- Using a Scheduler to Submit Computing Jobs

What is Princeton's Cloud?

- in this workshop, **Princeton's Cloud** refers to:
  remote computing systems managed by Princeton's Research Computing group
  http://www.princeton.edu/researchcomputing/

-hardware location:
  - High Performance Computing Research Center (HPCRC)
  - 47,000-square-foot facility opened in 2011

- computing systems make up TIGRESS:
  Terascale **I**nfrastructure for **G**roundbreaking **R**esearch in **E**ngineering and **S**cience

-in addition to remote computing systems, Research Computing also manages:
  software licenses and support (Stata, SAS, Matlab, …)
  visualization lab

How to Get Started

- request and obtain an account

- connect to the remote system

- transfer files (programming scripts, data, output)

- interact with remote system's operating system (usually Linux)

- execute computational jobs, often using a resource manager/scheduler

Requesting and Obtaining Accounts

two TIGRESS systems are available simply by registering on a webpage:

nobel

- load-balanced cluster of interactive computational Linux systems
- two Dell R610 servers named for Princeton Nobel Laureates, Compton and Davisson
- good entry point for researchers and students
- well-suited for:
  - access to commercially licensed software provided centrally
  - lower end computational tasks
  - teaching and coursework
- to register for an account, login to registration page:
  http://www.princeton.edu/researchcomputing/computational-hardware/nobel/

adroit

- 8 node cluster, adroit-01 through adroit-08
- 160 processors available, twenty per node
- each node contains 64 GB memory
- intended for developing small production jobs
- all jobs other than those that last for just a few minutes must be run through a scheduler
- to register for an account, login to registration page:
  http://www.princeton.edu/researchcomputing/computational-hardware/adroit/

configured just like the larger clusters; so moving from adroit to the larger clusters is easy

Other TIGRESS Systems

della

designed for serial jobs, small to medium parallel jobs,
and jobs requiring fairly large memory per task

tigressdata

single server useful for debugging, data access and visualization

tukey

main computational cluster for individuals in the Politics department

for more information about these and other TIGRESS systems, see
http://www.princeton.edu/researchcomputing/computational-hardware/ and
http://askrc.princeton.edu/question/23/how-do-i-get-an-account-on-a-tigress-system/

to request access to these other TIGRESS systems, see “For Prospective Users” at
http://www.princeton.edu/researchcomputing/access/

Connecting to Remote System

SSH (*secure shell*)

- provides access to remote systems over a network
- already installed on Mac OS X and Linux
- for Windows, can use ssh implementation at
  http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
  - look for "Windows installer for everything except PuTTYtel"
  - download installer and run it

connecting to remote system from a Windows system

- start a new PuTTY terminal session by finding PuTTY in "Start Menu"
- create a new connection
  hostname: `nobel.princeton.edu` or `adroit.princeton.edu`
  username: Princeton netid
- if successful: will see terminal window with machine name, user name and prompt

connecting to remote system from a Mac

- open a terminal application window
- at the prompt (usually **$** or **%**), enter
  `ssh nobel.princeton.edu` or
  `ssh adroit.princeton.edu`
- if successful, will see terminal window with machine name, user name and prompt

may need to first connect to nobel (via ssh) and from there, connect to adroit

Transferring Files

between local system and remote system, need to transfer:

- code (R scripts, Stata .do files, etc)
- data
- output

---

between local **Windows** system and remote Linux system:

- FileZilla
  - graphical file transfer program
  - open-source: https://filezilla-project.org/
- PSCP and PSFTP
  - command line tools from PuTTY

---

between local **Mac** system and remote Linux system:

- rsync
  - command line tool
  - already installed
- scp
  - command line tool
  - already installed
- FileZilla
  - graphical file transfer program
  - open source: https://filezilla-project.org/

Transferring Files

between local **Windows** system and remote Linux system -
using command line tool PSFTP from PuTTY

Select PSFTP from Start Menu

```
psftp: no hostname specified; use "open host.name" to connect
psftp> open nobel.princeton.edu
login as: dkoffman
dkoffman@nobel.princeton.edu's password: mypassword1234
Remote working directory is /n/homeserver/user/dkoffman
psftp> help
cd      change your remote working directory
lcd     change your local working directory
pwd     print your remote working directory
lpwd    print your local working directory
get     download a file from the server to your local machine
put     upload a file from your local machine to the server
exit    finish your SFTP session
psftp> put hello.R
psftp> put datafile.csv
psftp> exit
```

Transferring Files

between local **Mac** system and remote Linux system -
using **rsync** command **from terminal window**

SOURCE DESTINATION

```
$ rsync  ~/hello.R  dkoffman@nobel.princeton.edu:~/hello.R
dkoffman@nobel.princeton.edu's password:
$
```

- rsync: fast and flexible; many options can be used to adjust its behavior, for example:
  - **-r** recurse through sub-direcotires
  - **-v** verbosely print messages
  - **-z** compress data before transfer and decompress after transfer
- for (many) more options, see manual page (**man rsync**)
- often put **rsync** command with appropriate options within a shell script on local machine

Transferring Files

between local **Mac** system and remote Linux system -
using **scp** command **from terminal window**

SOURCE DESTINATION

```
$ scp  ~/hello.R  dkoffman@nobel.princeton.edu:~/hello.R
dkoffman@nobel.princeton.edu's password:
$
```

- differences between **rsync** and **scp**:
  - scp is more basic: regular, linear copy; fewer options for tweaking its behavior
  - rsync uses a delta transfer algorithm and some optimizations to make copying faster
    - for example, if destination file already exists, rsync will check file sizes and modification timestamps and skip further processing if both of those match

Running Stata or R Script Without a Scheduler

$ **stata -b do hello.do**

contents of results window sent to text file hello.log

$ Rscript -e 'a <- 5' -e 'a' > show_a.txt

$ Rscript hello.R > hello.txt

Submitting Computing Jobs to the Clusters

SLURM (**S**imple **L**inux **U**tility for **R**esource **M**anagement)
https://computing.llnl.gov/linux/slurm/quickstart.html
https://computing.llnl.gov/linux/slurm/sbatch.html

- cluster management and job scheduling system for large and small Linux clusters
- open source
- fault tolerant
- highly scalable

- submitting a job is similar to taking a ticket at a deli counter
- once machine is ready to run a job, it comes out of the queue

- submitting a job requires using specific commands in a specific format

Submitting a Serial Job

- create a job script for SLURM, here named `serial_ex2.cmd`

```
$ cat serial_ex2.cmd
#!/usr/bin/env bash

#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH -t 10:00
#SBATCH -o log.%j
#SBATCH -mail-type=begin
#SBATCH -mail-type=end

Rscript -e 'rnorm(1e3)'
```

- submit the job to the batch system (queue)

```
$ sbatch serial_ex2.cmd
submitted batch job 194717
$ ls log.*
log.194717
$
```

Email from SLURM

From: SLURM User [mailto:cses@princeton.edu]
Sent: Friday, May 01, 2015 2:45 PM
To: Dawn A. Koffman
Subject: SLURM Job_id=194717 Name=serial_ex2.cmd Began, Queued time 00:00:00

From: SLURM User [mailto:cses@princeton.edu]
Sent: Friday, May 01, 2015 2:46 PM
To: Dawn A. Koffman
Subject: SLURM Job_id=194717 Name=serial_ex2.cmd Ended, Run time 00:00:00

Job ID: 194717
Cluster: adroit
User/Group: dkoffman/pustaff
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:00:00
CPU Efficiency: 0.00% of 00:00:00 core-walltime Memory Utilized: 1.52 MB Memory Efficiency: 0.05% of 3.12 GB

Submitting a Serial Job

- create a job script for SLURM, here named `serial_ex3.cmd`

```
$ cat serial_ex3.cmd
#!/usr/bin/env bash

#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH -t 10:00
#SBATCH -o log.%j
#SBATCH -mail-type=begin
#SBATCH -mail-type=end

Rscript hello.R
```

- submit the job to the batch system (queue)

```
$ sbatch serial_ex3.cmd
submitted batch job 194718
$ ls log.*
log.194718
$
```

SLURM Commands

sbatch	- submit job
squeue	- display information about the jobs in the queue (jobid, name, user, time, nodes, nodelist (reason))
squeue –u *userid*	- display information about jobs in the queue for user = *userid*
srun	- submit a job for parallel execution

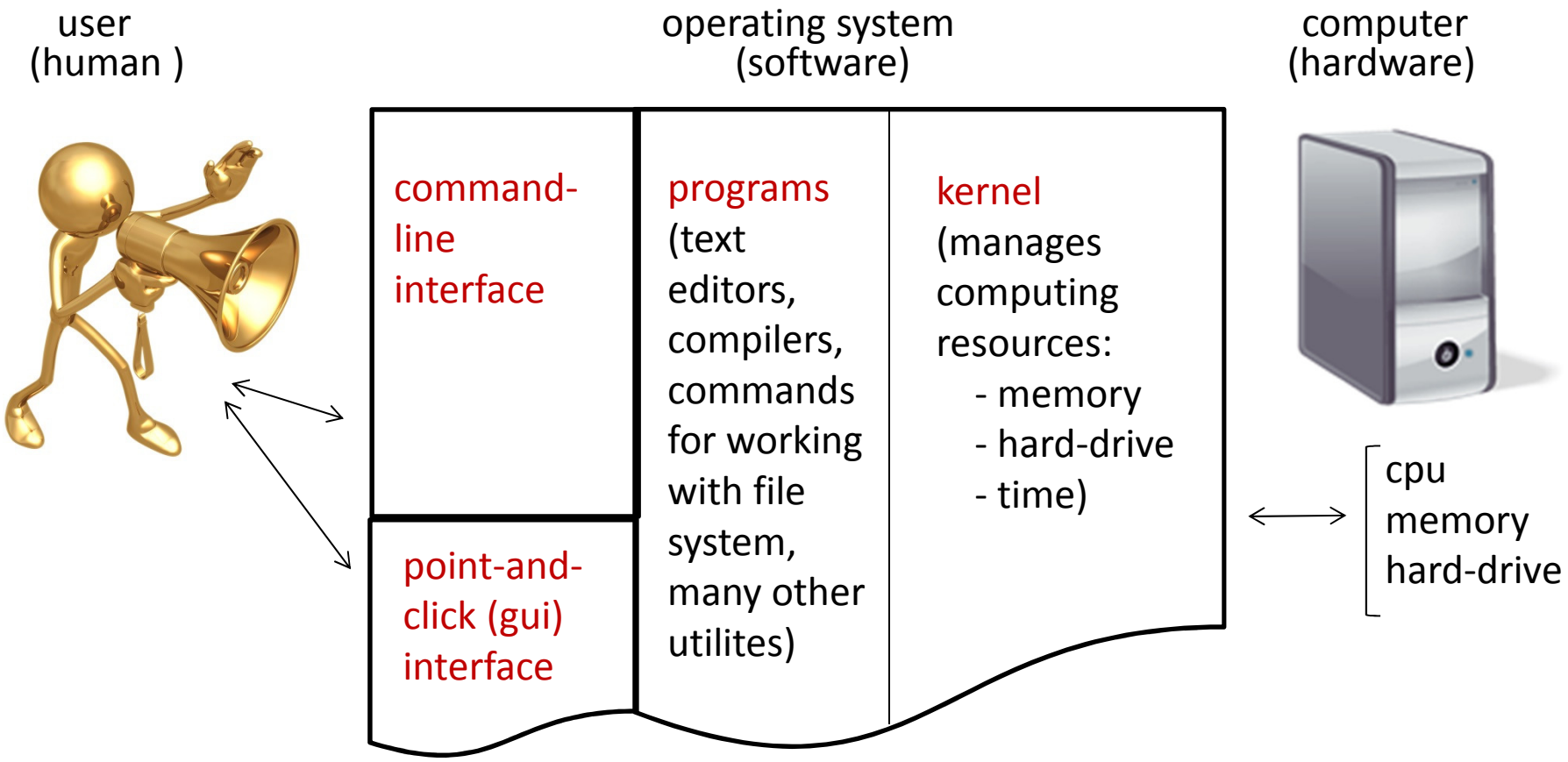
.
.
.

many other SLURM commands

Unix/Mac OS X User Interface and Philosophy

- Operating Systems
- Command-Line Interface
- Shell
- Unix Philosophy
- Command Execution Cycle
- Command History

User Interfaces



Comparison

command-line interface	point-and-click interface
- may have steeper learning curve, BUT provides constructs that can make many tasks very easy	- may be more intuitive, BUT can also be much more human-manual-labor intensive
- scales up very well when have lots of: data programs tasks to accomplish	- often does not scale up well when have lots of: data programs tasks to accomplish

Shell

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

a shell:

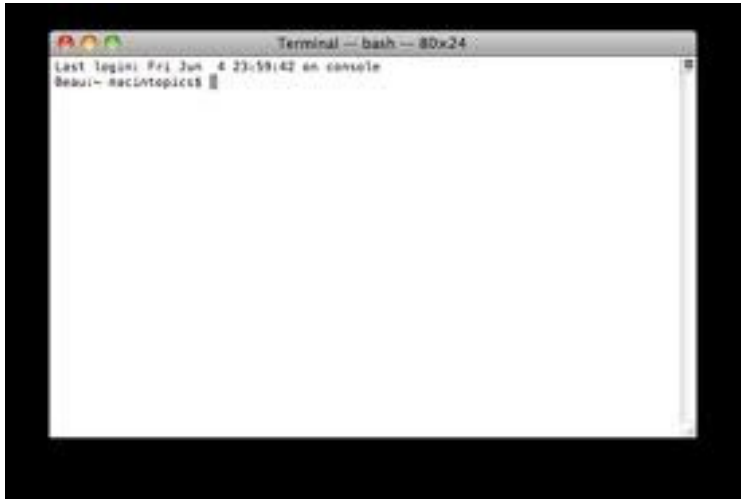
- prompts user for commands
- interprets user commands
- passes them onto the rest of the operating system which is hidden from the user

How do you access a shell ?

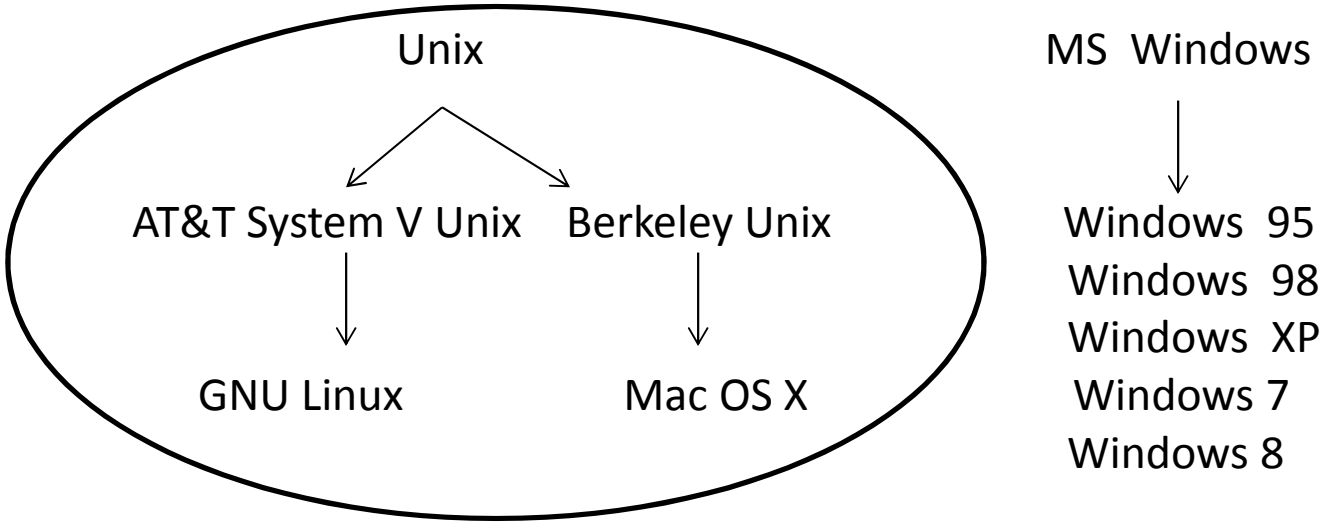
- if you have an account on a machine running Unix or Linux , just log in.
  A default shell will be running.

- if you are using a Mac, run the Terminal app.
  A default shell will be running.

-if Terminal app does not appear on the Shortcut Bar:
Go -> Applications -> Utilities -> Terminal



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
```

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

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

Recent Medicare Data Release

Recent Medicare Data Release

http://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/Medicare-Provider-Charge-Data/Physician-and-Other-Supplier.html

Data available in two formats:

- tab delimited file format (requires importing into database or statistical software; SAS® read-in language is included in the download ZIP package)

  Note: the compressed zip file contains a tab delimited data file which is 1.7GB uncompressed and contains more than 9 million records, thus importing this file into Microsoft Excel will result in an incomplete loading of data. Use of database or statistical software is required; a SAS® read-in statement is supplied.

- Microsoft Excel format (.xlsx), split by provider last name (organizational providers with name starting with a numeric are available in the "YZ" file)

Recent Medicare Data Release

```
$unzip Medicare-Physician-and-Other-Supplier-PUF-CY2012.zip

$wc -l Medicare-Physician-and-Other-Supplier-PUF-CY2012.txt
9153274  Medicare-Physician-and-Other-Supplier-PUF-CY2012.txt

$ tr "\t" "|" < Medicare-Physician-and-Other-Supplier-PUF-CY2012.txt  > medicare.pipe.txt

$ rm Medicare-Physician-and-Other-Supplier-PUF-CY2012.txt

$head -1 medicare.pipe.txt
npi|nppes_provider_last_org_name|nppes_provider_first_name|nppes_provider_mi|
nppes_credentials|nppes_provider_gender|nppes_entity_code|nppes_provider_street1|
nppes_provider_street2|nppes_provider_city|nppes_provider_zip|nppes_provider_state|
nppes_provider_country|provider_type|medicare_participation_indicator|place_of_service|
hcpcs_code|hcpcs_description|line_srvc_cnt|bene_unique_cnt|bene_day_srvc_cnt|
average_Medicare_allowed_amt|stdev_Medicare_allowed_amt|average_submitted_chrg_amt|
stdev_submitted_chrg_amt|average_Medicare_payment_amt|stdev_Medicare_payment_amt

$head medicare.pipe.txt

$tail medicare.pipe.txt
```

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

```
$ 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

```
$ pwd
/u/dkoffman/unix
$ wget http://opr.princeton.edu/workshops/201401/wdata.gz   # NOT standard  on  OS X
                        **** OR ****
$ curl http://opr.princeton.edu/workshops/201401/wdata.gz -o wdata.gz    # available on OS X

$ gunzip wdata.gz

$ ls
```

Displaying File Contents

```
$ wc wdata

$ cat wdata

$ head wdata

$ head -1 wdata

$ tail wdata

$ tail -2 wdata

$ more wdata
```

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
```

File Access and Permission

```
$ ls  -l

-rw-r--r--   1   dkoffman   rpcuser   8586 Apr 1 14:46   wdata
-rw-------   1   dkoffman   rpcuser   8586 Apr 1 14:27   wdata.save
```

User classes

- owner (user) = u
- group = g
- other = o
- all = a

File permissions

- read (cat, tail, cp, ...) = r
- write (vi, emacs) = w
- execute (run program) = x

```
$ chmod   g+w    wdata

$ ls -l

$ chmod   go-r   wdata.save

$ ls -l
```

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

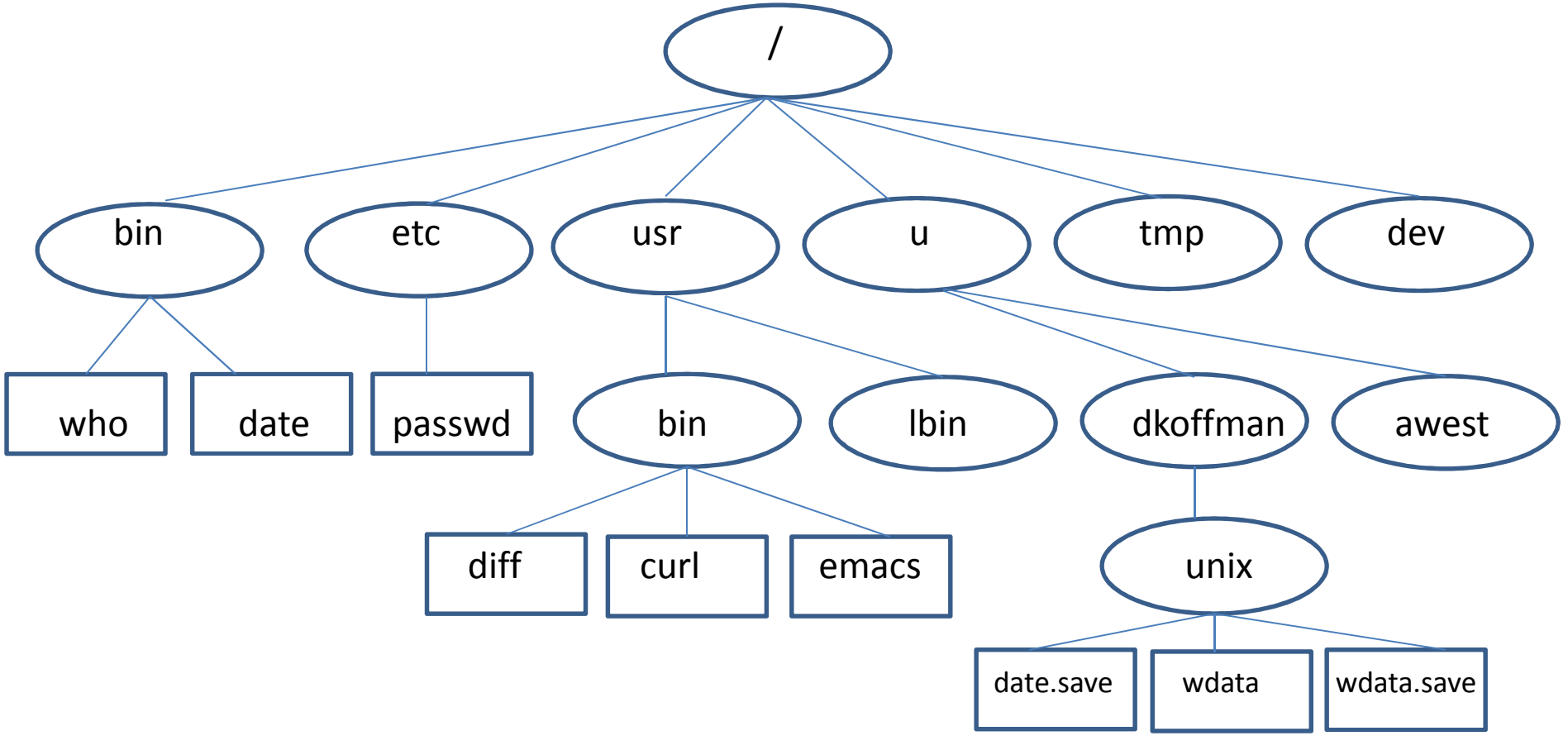
shell can automatically put file names
on a command line if user uses
file name generation characters

?	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

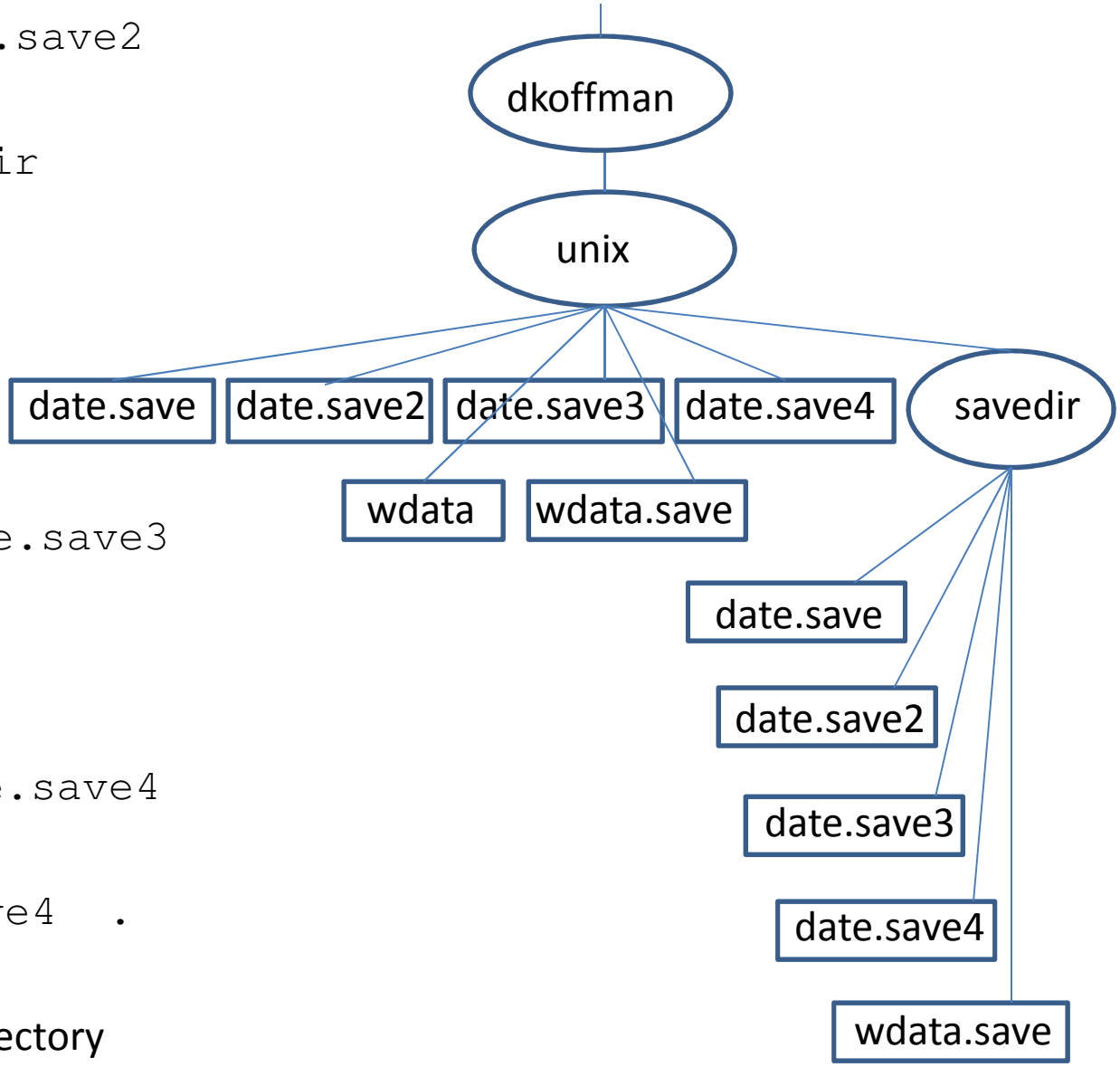
```
$ cp date.save  date.save2
$ mkdir   savedir
$ cp  *.save*  savedir
```

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



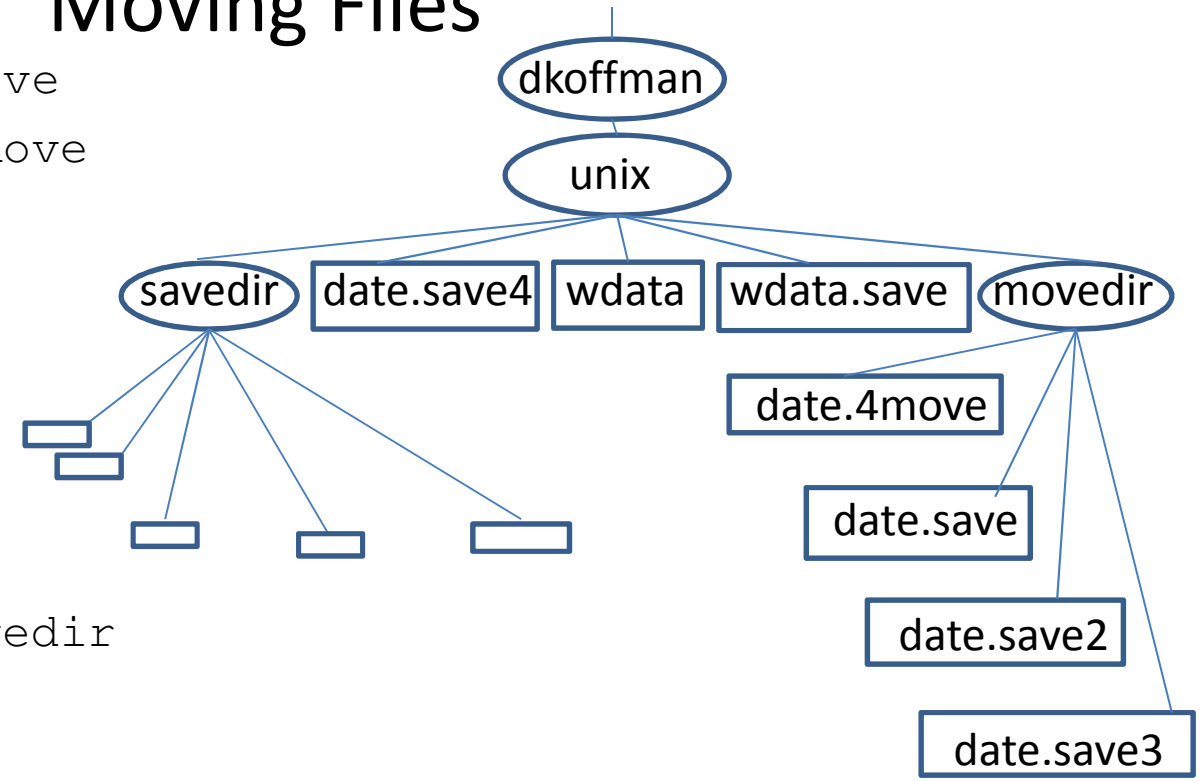
Moving Files

```
$ 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

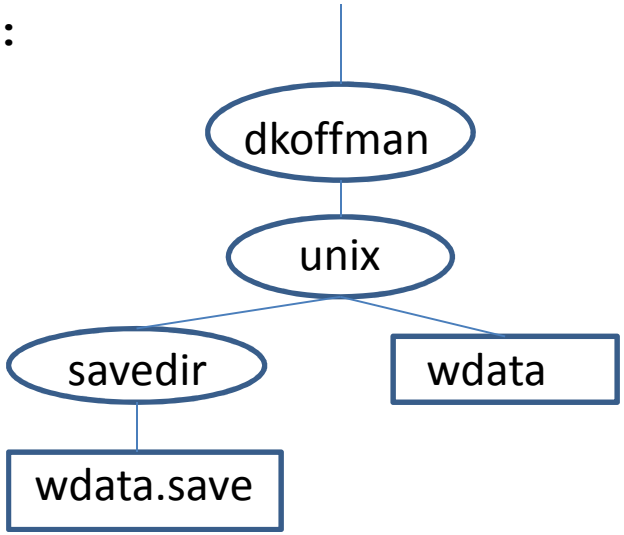
```
$ ls
$ cd movedir
$ ls
$ mv  ../date.save   .
$ ls
$ cd ..
```



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

- 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
```

Foreground and Background Command Processing

Foreground command processing

- one command line must complete execution before next command line begins execution
- “normal” way commands are processed

Background command processing

- next command line begins execution before background command completes
- any standard output is usually redirected to a file
- <BRK> and <DEL> are ignored
- identification number is displayed after background command is entered ... process id
- can stop a command running in the background using the kill command and the process id

```
$ command  arg1  arg2  > outfile &
 10411
$ kill 10411
$
```

execute command in the background

Background Command Processing

- normally, a hang-up signal (logging off) is not ignored by a command executing in the background, and will cause it to terminate

- nohup prefix allows a command to continue running even if a hang-up signal is received

```
$ nohup  cmd  arg1  arg2  ... &
```

- to check to see if a background command is still running and to obtain its process id, use ps command

Shell Programs

- Creating and Executing Shell Programs
- emacs Text Editor
- Adding Comments

How to Create and Execute a Shell Program

- Use a text editor such as emacs or vi to create a new file
- Enter a “shebang” (#!) indicating which shell (sh, bash, csh, ...) should execute the program
- Enter shell command lines (and optionally, shell control structures for branching and looping)
- Save the new file and exit the text editor

- Turn on execute permission for your new file
- Make sure the new file is in a directory where the shell looks for commands (PATH variable)

- Invoke the shell program by using the new file name as a command name

Text Editors

vi: visual text editor (wysiwyg) compared to older line-oriented editors (ex and ed)

"moded" editor ... need to use a command to allow adding text to a file

vim: vi improved

has both a command line interface and a graphical user interface

emacs: text editor known for being customizable and extensible

nice interface to R, LaTex, C/C++

"non-moded" editor ... entered text becomes part of file ...

control sequences are used as editing commands

aquamacs: "a modern editor based on emacs that makes Mac users feel at home"

*** here we briefly illustrate basic emacs, which is available on both Linux and Mac OS X

resources for learning emacs

- interactive tutorial: within emacs, use `<CTRL>h t`
- manual: http://www.gnu.org/software/emacs/manual/
- aquamacs: http://aquamacs.org/

Basic emacs Text Editing Commands

enter emacs to edit existing file	`emacs <file.existing>`
enter emacs to create a new file	`emacs <file.new>`
save file	`<CTRL>x <CTRL>s`
exit emacs	`<CTRL>x <CTRL>c`
move cursor one character forward	`<CTRL>f`
move cursor one character backward	`<CTRL>b`
move cursor to next line	`<CTRL>n`
move cursor to previous line	`<CTRL>p`
delete current line	`<CTRL>k`
delete current character	`<CTRL>d` or `<Delete>` or `<Backspace>`
undo last edit	`<CTRL>u`
access help	`<CTRL>h`
access emacs interactive tutorial	`<CTRL>h t`

Creating and Executing a New Shell Program

```
$ emacs myprog
 #!/bin/bash
 echo hello
 date
 who am i
 echo have a good day
 <CTRL>x <CTRL>s
 <CTRL>x <CTRL>c
$ chmod +x myprog
$ echo ${PATH}
/usr/local/bin:/bin:/usr/bin
$ pwd
/u/dkoffman/unix
$ PATH=${PATH}:/u/dkoffman/unix
$ myprog
 hello
 Thu Apr 10 13:00:46 EDT 2014
 dkoffman  pts/80   2014-04-10  12:59  (abc-xyz-princeton.edu)
 have a good day
$
```

Comments

starts a comment

<CR> ends a comment

```
$ cat wdata_le_part_scan
#
# Output consists of the first 4 lines
# of all wdata_le_part_[a-z][a-z] files
# in the current directory.
#
# Output is placed in a single file
# called wdata_le_part_scan.out
# in the current directory.
#
#!/bin/bash
head -4 wdata_le_part_[a-z][a-z] > wdata_le_part_scan.out
$
```

Stream Editor: sed

- Examples
- File Containing Edits
- Selecting Lines
- Writing Lines to Separate Files
- Using sed to Create a sed Script

Stream Editor: sed

- modifies text files using a list of editing commands, modifications not performed interactively
- original files remain unchanged ... modified versions are written to standard output
- sed is a filter, works similar to `cut` and `tr`

```
$ sed “s/stan/STAN/” wdata.clean

$ sed “s/,/|/”   wdata.clean

$ sed “s/,/|/g”  wdata.clean

$ sed “s_/_~~_”  wdata.clean

$ sed –e “s/stan/STAN/” –e “s/,/|/g” –e “s_/_~~_”  wdata.clean

$ sed “s/,.*,//” wdata.clean

$ sed “s/,.*,/:/” wdata.clean
```

sed Example

```
$ cat oprworkshops
#!/bin/bash
curl –s  http://opr.princeton.edu/workshops/   -o wwpage
grep 'h5 class="title"' wwpage | sed  –e 's_.*">__‘   –e 's_</a.*__'   –e s/,// >wtitles
grep '>Date' wwpage | sed -e's/.*em>: //' -e's_</p>__'  -es/,// -e's/;.*//' -e's/^[MWTFS].*day //' >wdates
paste -d"," wtitles  wdates | tee wtitlesdates.csv
rm wwpage  wtitles  wdates
$ oprworkshops
Tour of the Terminal: Using Unix or Mac OS X Command-Line,May 5 2014
Introduction to Python,May 9 2014
Data Management with pandas (Python),May9 2014
Introduction to Python,January 14 2014
Introduction to ggplot2,January 9 2014
Introduction to Stata,September 17 2013
Introduction to Stata Data Management,September 18 2013
Introduction to Stata 13 Graphics,September 19 2013
Graphical Models for Causal Inference with Observational Data,May 21 2013
Data Science for Social Scientists,May 24 2013
Stata 12 Graphics,May 7 2013
$
```

sed Example

```
$ cat countries
#!/bin/bash
curl –s http://www.un.org/en/members/index.shtml/  -o unmemberswebpage
grep 'title=' unmemberswebpage | sed -e's_.*title="__' -e's_".*__' -e's/,/:/g' -e"s/^M$//" -e"s/ $//" | uniq >uncountries
grep 'joindate' unmemberswebpage | sed -e's/.*">//'  -e's_<.*__'  -e's_/_-_g' >unjoindates
paste -d"," uncountries unjoindates | sort | tee uncountriesjoindates.csv
$ countries
Afghanistan,19-11-1946
Albania,14-12-1955
Algeria,08-10-1962
Andorra,28-07-1993
Angola,01-12-1976
        .
        .
        .
United States of America,24-10-1945
Uruguay,18-12-1945
Uzbekistan,02-03-1992
Vanuatu,15-09-1981
Venezuela,15-11-1945
Viet Nam,20-09-1977
Yemen,30-09-1947
Zambia,01-12-1964
Zimbabwe,25-08-1980
$
```

Stream Editor sed: File Containing Edits

- if there are many modifications to be made, a file can be used to store edits

```
$ cat sedscript
 s/stan/STAN/
 s/,/|/g
 s_/_~~_
$ sed –f sedscript wdata.clean
$
```

sed: Editing Select Lines Using Line Numbers

-can specify which lines should be a “operated on” by sed commands using line numbers

- line number
- range of line numbers

```
$ sed “92 s/stan/STAN/” wdata.clean

$ sed “92,99 s/stan/STAN/” wdata.clean

$ sed “1,99 s/,/|/g” wdata.clean

$ sed “100,$ s/,/|/g” wdata.clean
```

sed: Editing Select Lines Using Regular Expressions

-can specify which lines should be a “operated on” by sed commands using regular expressions

- lines containing a pattern
- range of line from first line up through a line containing a pattern
- range of lines from a line containing a pattern through the last line
- all lines between two lines containing particular patterns

```
$ sed “/^K/s/stan/STAN/” wdata.clean

$ sed “1,/^Kaz/ s/stan/STAN/” wdata.clean

$ sed “/^Kaz/,$ s/stan/STAN/” wdata.clean

$ sed “/Benin/,/Zimbabwe/ s/,/|/g”  wdata.clean
```

sed: Writing Lines to Separate Files

```
$ cat sedscript_w
/Africa/w wdata.Africa
/Europe/w wdata.Europe
/Americas/w wdata.Americas
/Asia.*Oceania/w wdata.Asia.Oceania
$ sed -n -f sedscript_w wdata.clean
$ wc -l wdata.clean wdata.Africa wdata.Europe wdata.Americas wdata.Asia.Oceania
     158 wdata.clean
      48 wdata.Africa
      36 wdata.Europe
      25 wdata.Americas
      49 wdata.Asia.Oceania
     316 total
$
```

sed: Writing Lines to Separate Files

```
$ cat sedscript_w
/CT|US/w medicare.CT.txt
/NY|US/w medicare.NY.txt
/NJ|US/w medicare.NJ.txt
/PA|US/w medicare.PA.txt
/MD|US/w medicare.MD.txt
/VA|US/w medicare.VA.txt
/CA|US/w medicare.CA.txt
/FL|US/w medicare.FL.txt
/TX|US/w medicare.TX.txt
/OH|US/w medicare.OH.txt
/IL|US/w medicare.IL.txt
$ sed -n -f sedscript_w medicare.pipe.txt
$ wc -l medicare.[A-Z][A-Z].txt
   716330 medicare.CA.txt
   125891 medicare.CT.txt
   667995 medicare.FL.txt
   387623 medicare.IL.txt
   187979 medicare.MD.txt
   306379 medicare.NJ.txt
   592577 medicare.NY.txt
   339208 medicare.OH.txt
   403924 medicare.PA.txt
   628122 medicare.TX.txt
   241626 medicare.VA.txt
  4597654 total
$
```

Using sed to Create a sed Script

```
$ cat sub.states
NY
CT
PA
CA
IL
OH
FL
MD
VA
NJ
TX
$cat make_subsed
#!/bin/bash
sed  -e "s/.*/&:&/"  -e 's_^_/_' -e 's_:_|US/w medicare._' -e 's/$/.txt/'  sub.states > sedscript
cat sedscript
sed -n -f sedscript medicare.pipe.txt
rm sedscript
$
```

Review of Commands

date	gunzip	rmdir	tr
who	cat	man	echo
cal	head	sort	tee
pwd	tail	wc	nohup
ls	more	cut	kill
mkdir	cp	paste	ps
cd	mv	uniq	emacs
history	rm	grep	sed
curl	diff	split	
wget	chmod		

The End!