HARVARD UNIVERSITY

Extended Unix: sed, awk, grep, and bash scripting basics

Scott Yockel, PhD
Harvard - Research Computing

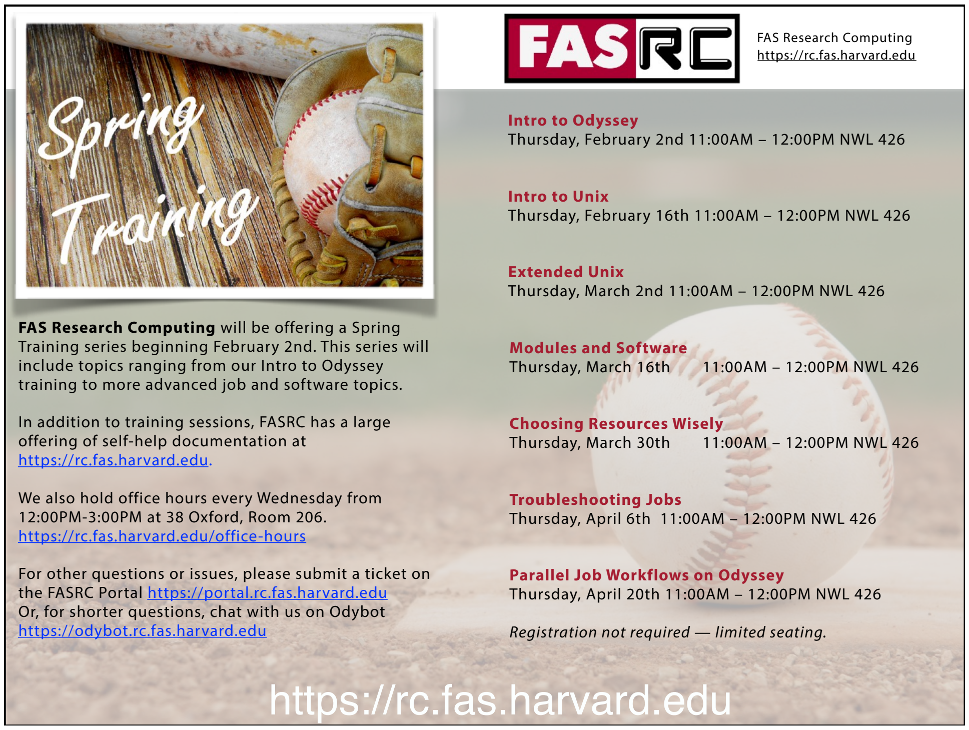
HARVARD UNIVERSITY

What is Research Computing?

Faculty of Arts and Sciences (FAS) department that handles non-enterprise IT requests from researchers. *(Contact HUIT for most Desktop, Laptop, networking, printing, and email issues.)*

- **RC Primary Services:**
  - Odyssey Supercomputing Environment
  - Lab Storage
  - Instrument Computing Support
  - Hosted Machines (virtual or physical)
- **RC Staff:**
  - 20 staff with backgrounds ranging from systems administration to development-operations to Ph.D. research scientists.
  - Supporting 600 research groups and 3000+ users across FAS, SEAS, HSPH, HBS, GSE.
  - For bio-informatics researchers the Harvard Informatics group is closely tied to RC and is there to support the specific problems for that domain.

2


Spring Training

FAS Research Computing
https://rc.fas.harvard.edu

FAS Research Computing will be offering a Spring Training series beginning February 2nd. This series will include topics ranging from our Intro to Odyssey training to more advanced job and software topics.

In addition to training sessions, FASRC has a large offering of self-help documentation at https://rc.fas.harvard.edu.

We also hold office hours every Wednesday from 12:00PM-3:00PM at 38 Oxford, Room 206. https://rc.fas.harvard.edu/office-hours

For other questions or issues, please submit a ticket on the FASRC Portal https://portal.rc.fas.harvard.edu
Or, for shorter questions, chat with us on Odybot https://odybot.rc.fas.harvard.edu

Intro to Odyssey
Thursday, February 2nd 11:00AM – 12:00PM NWL 426

Intro to Unix
Thursday, February 16th 11:00AM – 12:00PM NWL 426

Extended Unix
Thursday, March 2nd 11:00AM – 12:00PM NWL 426

Modules and Software
Thursday, March 16th 11:00AM – 12:00PM NWL 426

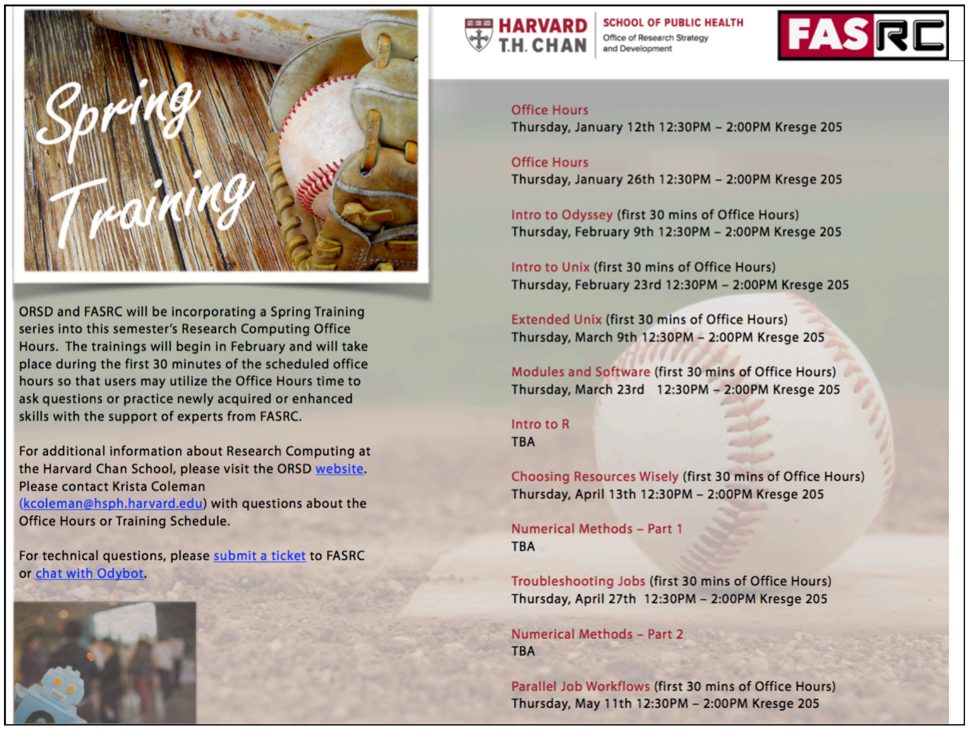
Choosing Resources Wisely
Thursday, March 30th 11:00AM – 12:00PM NWL 426

Troubleshooting Jobs
Thursday, April 6th 11:00AM – 12:00PM NWL 426

Parallel Job Workflows on Odyssey
Thursday, April 20th 11:00AM – 12:00PM NWL 426

Registration not required — limited seating.

https://rc.fas.harvard.edu


Spring Training

HARVARD T.H. CHAN SCHOOL OF PUBLIC HEALTH
Office of Research Strategy and Development

ORSD and FASRC will be incorporating a Spring Training series into this semester's Research Computing Office Hours. The trainings will begin in February and will take place during the first 30 minutes of the scheduled office hours so that users may utilize the Office Hours time to ask questions or practice newly acquired or enhanced skills with the support of experts from FASRC.

For additional information about Research Computing at the Harvard Chan School, please visit the ORSD website. Please contact Krista Coleman (kcoleman@hsph.harvard.edu) with questions about the Office Hours or Training Schedule.

For technical questions, please submit a ticket to FASRC or chat with Odybot.

Office Hours
Thursday, January 12th 12:30PM – 2:00PM Kresge 205

Office Hours
Thursday, January 26th 12:30PM – 2:00PM Kresge 205

Intro to Odyssey (first 30 mins of Office Hours)
Thursday, February 9th 12:30PM – 2:00PM Kresge 205

Intro to Unix (first 30 mins of Office Hours)
Thursday, February 23rd 12:30PM – 2:00PM Kresge 205

Extended Unix (first 30 mins of Office Hours)
Thursday, March 9th 12:30PM – 2:00PM Kresge 205

Modules and Software (first 30 mins of Office Hours)
Thursday, March 23rd 12:30PM – 2:00PM Kresge 205

Intro to R
TBA

Choosing Resources Wisely (first 30 mins of Office Hours)
Thursday, April 13th 12:30PM – 2:00PM Kresge 205

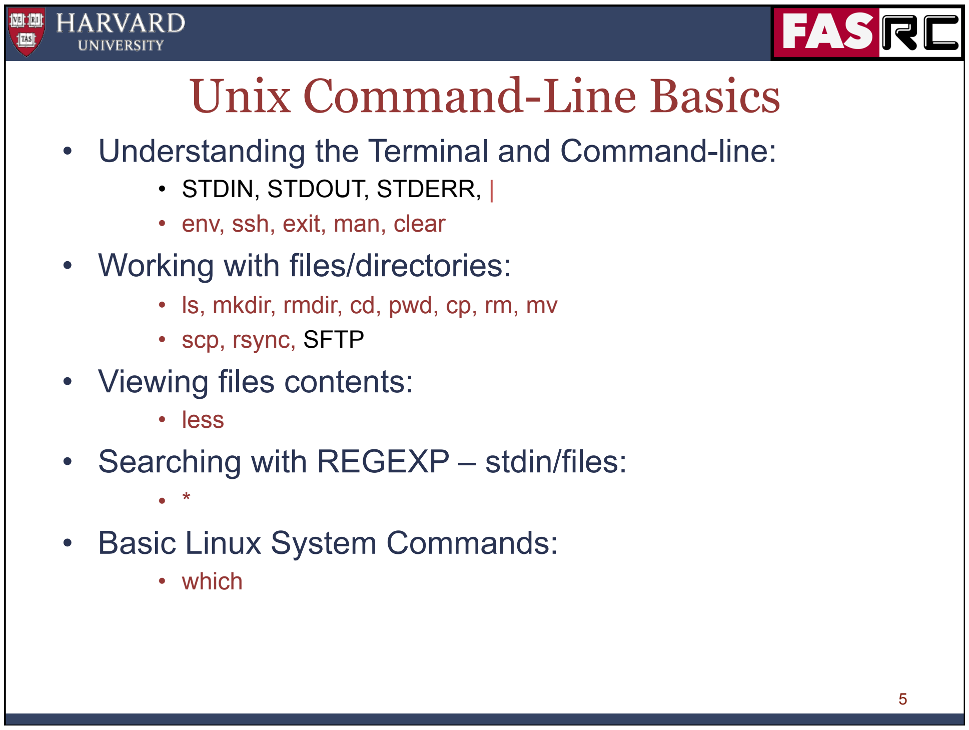
Numerical Methods – Part 1
TBA

Troubleshooting Jobs (first 30 mins of Office Hours)
Thursday, April 27th 12:30PM – 2:00PM Kresge 205

Numerical Methods – Part 2
TBA

Parallel Job Workflows (first 30 mins of Office Hours)
Thursday, May 11th 12:30PM – 2:00PM Kresge 205

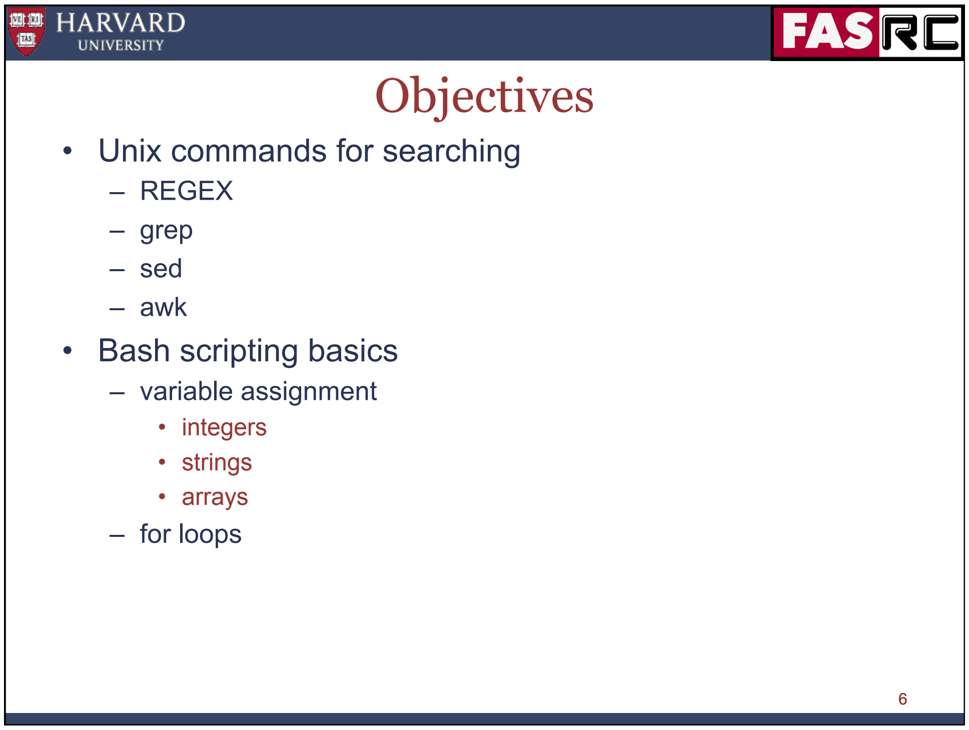
https://rc.fas.harvard.edu/training/spring-2017/



Unix Command-Line Basics

- Understanding the Terminal and Command-line:
    - STDIN, STDOUT, STDERR, |
    - env, ssh, exit, man, clear
- Working with files/directories:
    - ls, mkdir, rmdir, cd, pwd, cp, rm, mv
    - scp, rsync, SFTP
- Viewing files contents:
    - less
- Searching with REGEXP – stdin/files:
    - *
- Basic Linux System Commands:
    - which

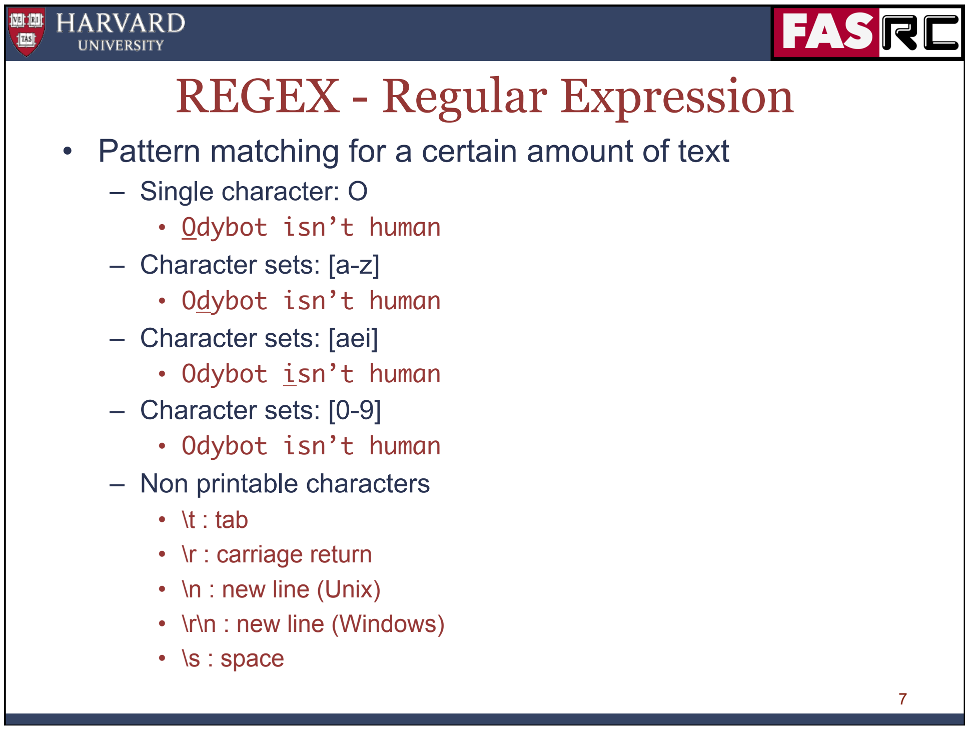
5



Objectives

- Unix commands for searching
    - REGEX
    - grep
    - sed
    - awk
- Bash scripting basics
    - variable assignment
        - integers
        - strings
        - arrays
    - for loops

6

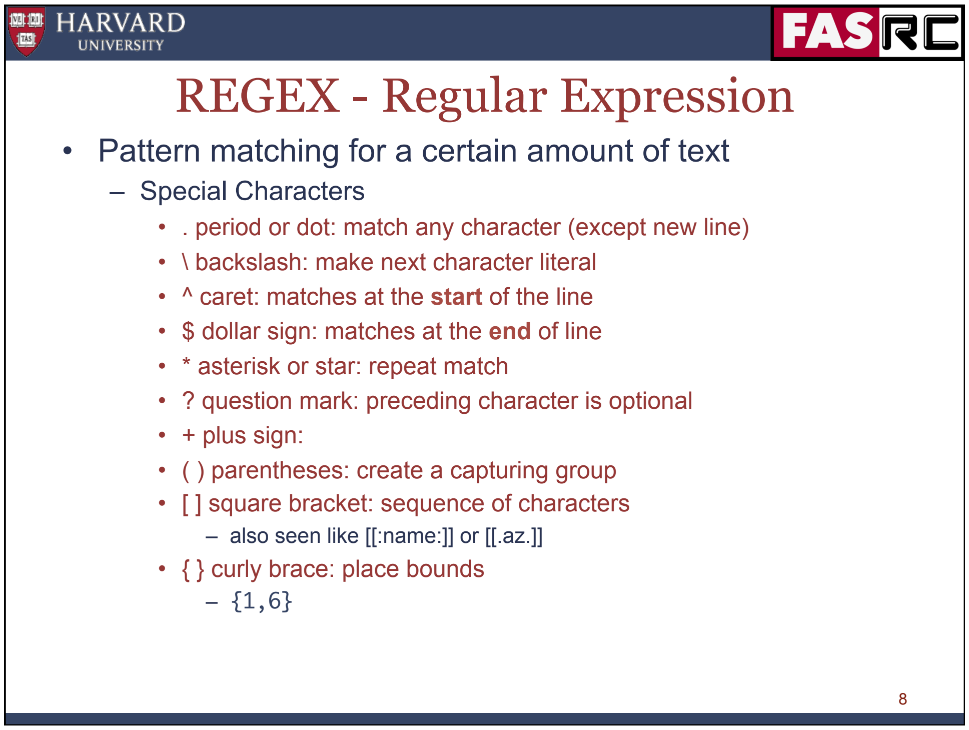


HARVARD UNIVERSITY FASRC

REGEX - Regular Expression

- Pattern matching for a certain amount of text
  - Single character: O
    - `Odybot isn’t human`
  - Character sets: [a-z]
    - `Odybot isn’t human`
  - Character sets: [aei]
    - `Odybot isn’t human`
  - Character sets: [0-9]
    - `Odybot isn’t human`
  - Non printable characters
    - \t : tab
    - \r : carriage return
    - \n : new line (Unix)
    - \r\n : new line (Windows)
    - \s : space

7

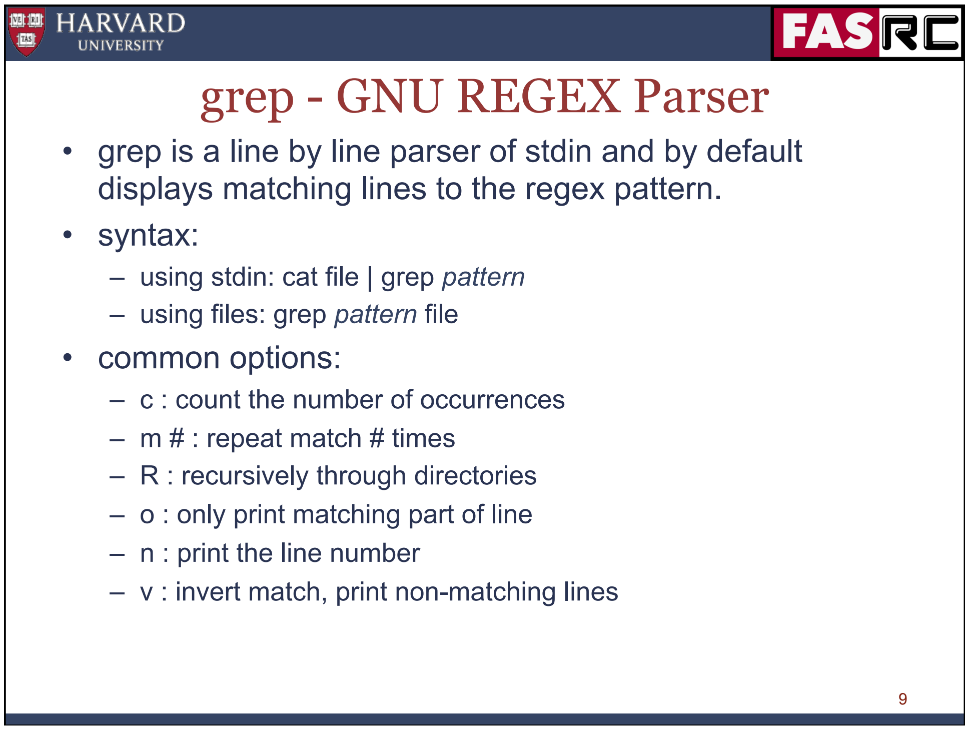


HARVARD UNIVERSITY FASRC

REGEX - Regular Expression

- Pattern matching for a certain amount of text
  - Special Characters
    - . period or dot: match any character (except new line)
    - \ backslash: make next character literal
    - ^ caret: matches at the **start** of the line
    - $ dollar sign: matches at the **end** of line
    - * asterisk or star: repeat match
    - ? question mark: preceding character is optional
    - + plus sign:
    - ( ) parentheses: create a capturing group
    - [ ] square bracket: sequence of characters
      - also seen like [[:name:]] or [[.az.]]
    - { } curly brace: place bounds
      - `{1,6}`

8

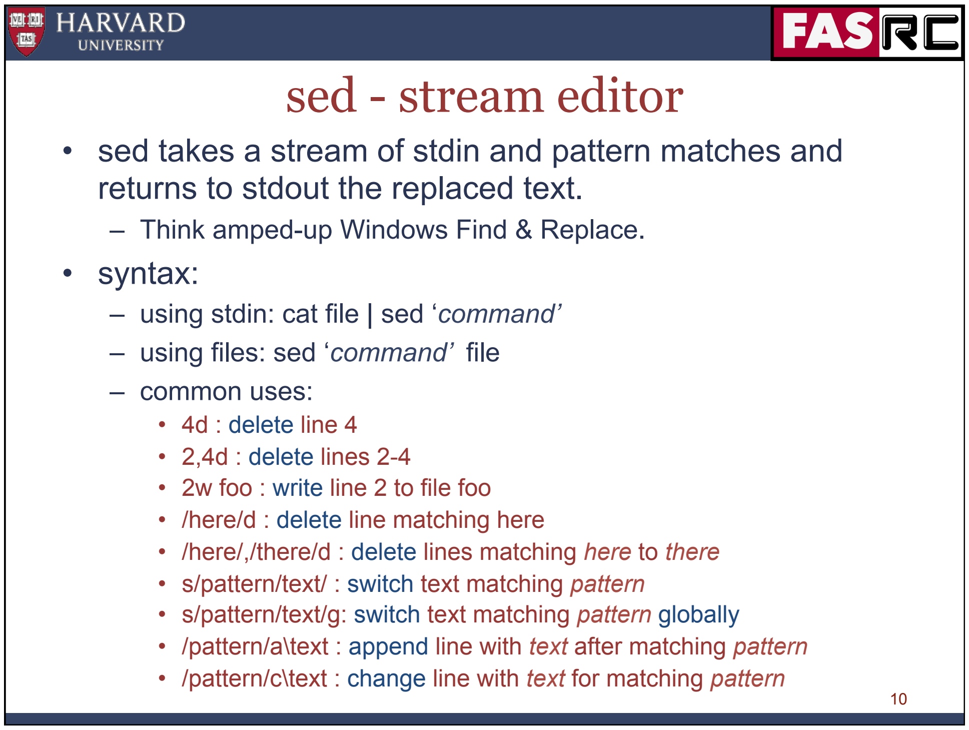


HARVARD UNIVERSITY FASRC

grep - GNU REGEX Parser

- grep is a line by line parser of stdin and by default displays matching lines to the regex pattern.
- syntax:
  - using stdin: cat file | grep *pattern*
  - using files: grep *pattern* file
- common options:
  - c : count the number of occurrences
  - m # : repeat match # times
  - R : recursively through directories
  - o : only print matching part of line
  - n : print the line number
  - v : invert match, print non-matching lines

9

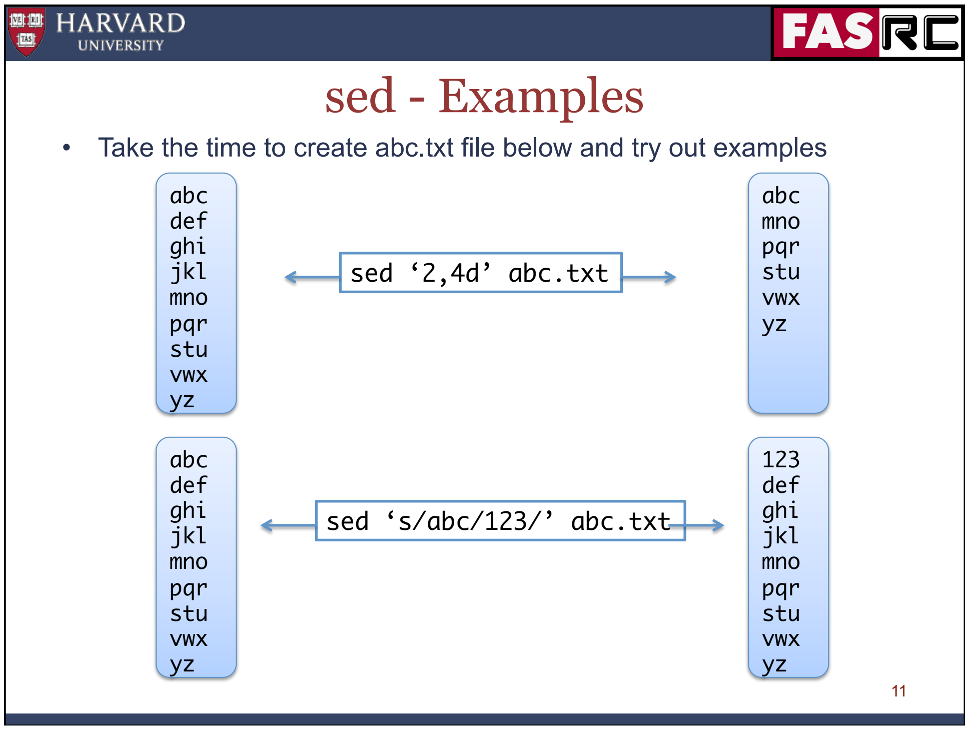


HARVARD UNIVERSITY FASRC

sed - stream editor

- sed takes a stream of stdin and pattern matches and returns to stdout the replaced text.
  - Think amped-up Windows Find & Replace.
- syntax:
  - using stdin: cat file | sed '*command*'
  - using files: sed '*command*' file
  - common uses:
    - 4d : delete line 4
    - 2,4d : delete lines 2-4
    - 2w foo : write line 2 to file foo
    - /here/d : delete line matching here
    - /here/,/there/d : delete lines matching *here* to *there*
    - s/pattern/text/ : switch text matching *pattern*
    - s/pattern/text/g: switch text matching *pattern* globally
    - /pattern/a\text : append line with *text* after matching *pattern*
    - /pattern/c\text : change line with *text* for matching *pattern*

10



sed - Examples

- Take the time to create abc.txt file below and try out examples

```
abc
def
ghi
jkl
mno
pqr
stu
vwx
yz
```

```
sed '2,4d' abc.txt
```

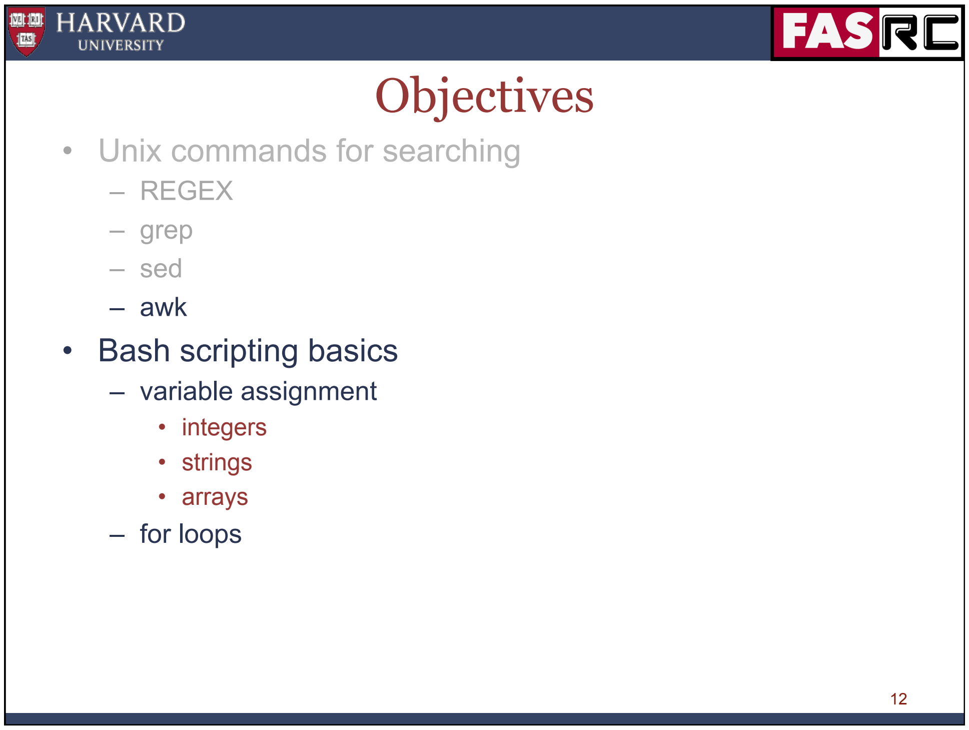
```
abc
mno
pqr
stu
vwx
yz
```

```
abc
def
ghi
jkl
mno
pqr
stu
vwx
yz
```

```
sed 's/abc/123/' abc.txt
```

```
123
def
ghi
jkl
mno
pqr
stu
vwx
yz
```

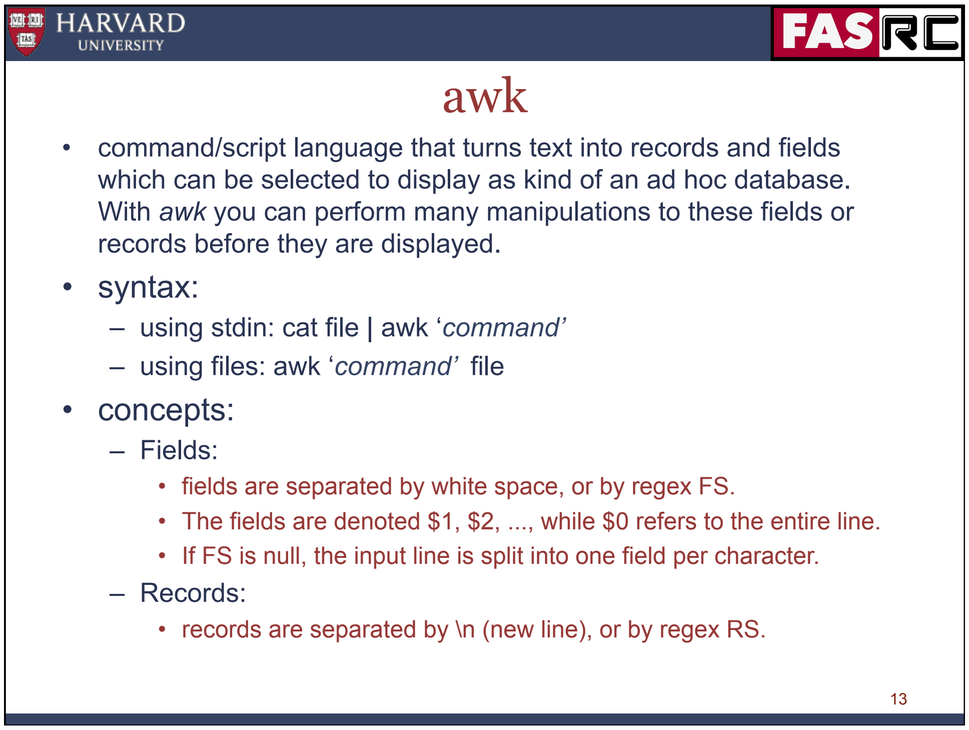
11



Objectives

- Unix commands for searching
  - REGEX
  - grep
  - sed
  - awk
- Bash scripting basics
  - variable assignment
    - integers
    - strings
    - arrays
  - for loops

12

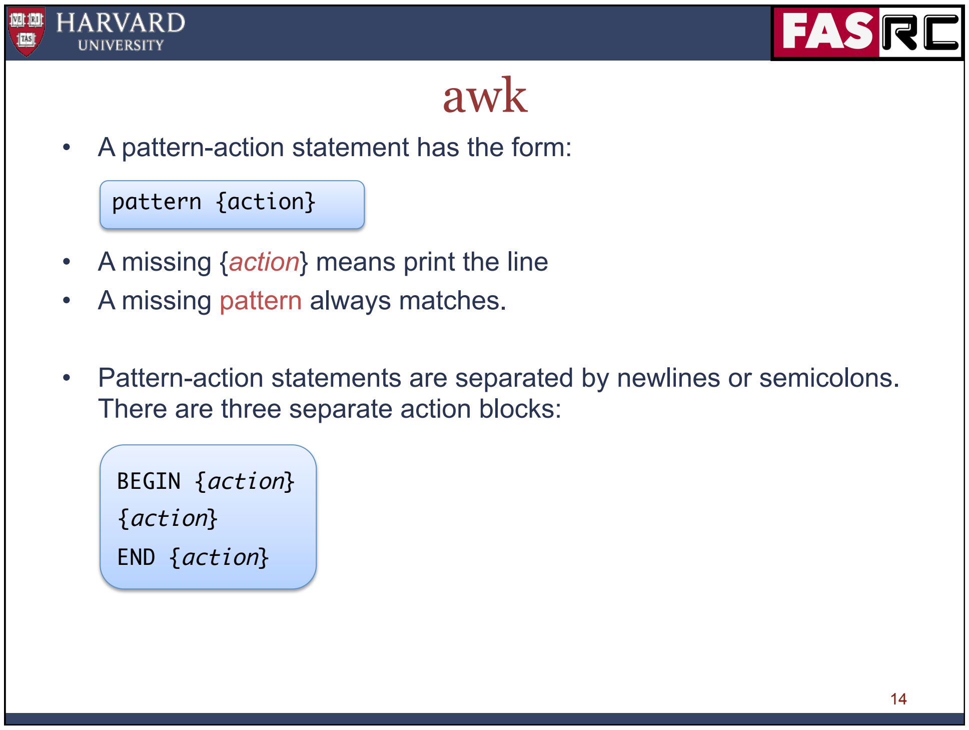


HARVARD UNIVERSITY FASRC

awk

- command/script language that turns text into records and fields which can be selected to display as kind of an ad hoc database. With *awk* you can perform many manipulations to these fields or records before they are displayed.
- syntax:
  - using stdin: cat file | awk '*command*'
  - using files: awk '*command*' file
- concepts:
  - Fields:
    - fields are separated by white space, or by regex FS.
    - The fields are denoted $1, $2, ..., while $0 refers to the entire line.
    - If FS is null, the input line is split into one field per character.
  - Records:
    - records are separated by \n (new line), or by regex RS.

13



HARVARD UNIVERSITY FASRC

awk

- A pattern-action statement has the form:

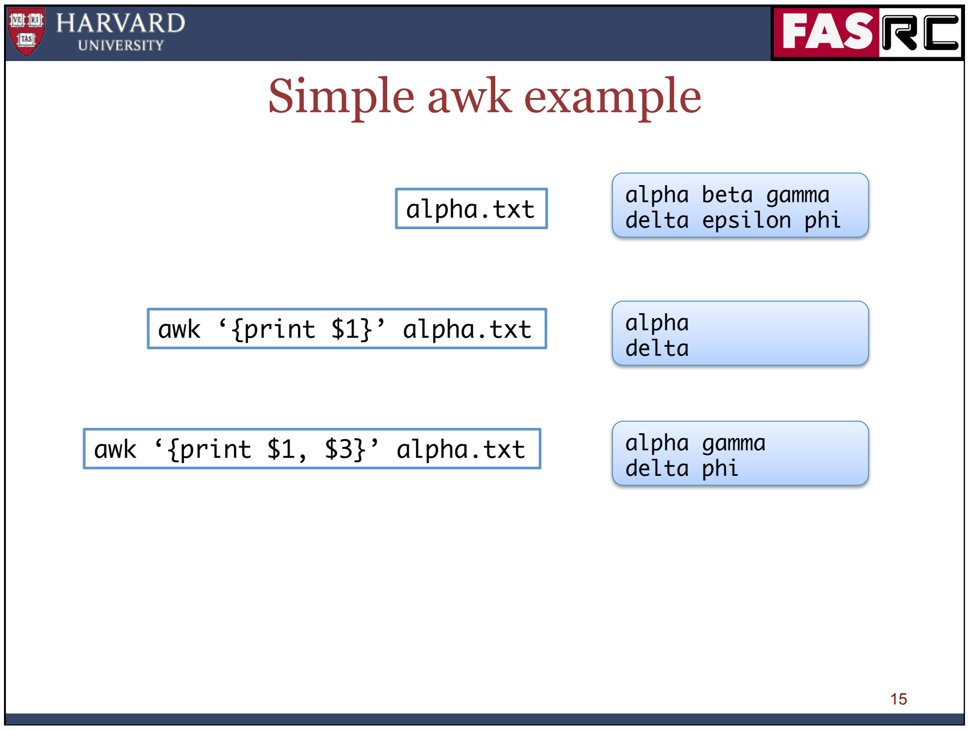
```
pattern {action}
```

- A missing {*action*} means print the line
- A missing pattern always matches.

- Pattern-action statements are separated by newlines or semicolons. There are three separate action blocks:

```
BEGIN {action}
{action}
END {action}
```

14



Simple awk example

```
alpha.txt
```

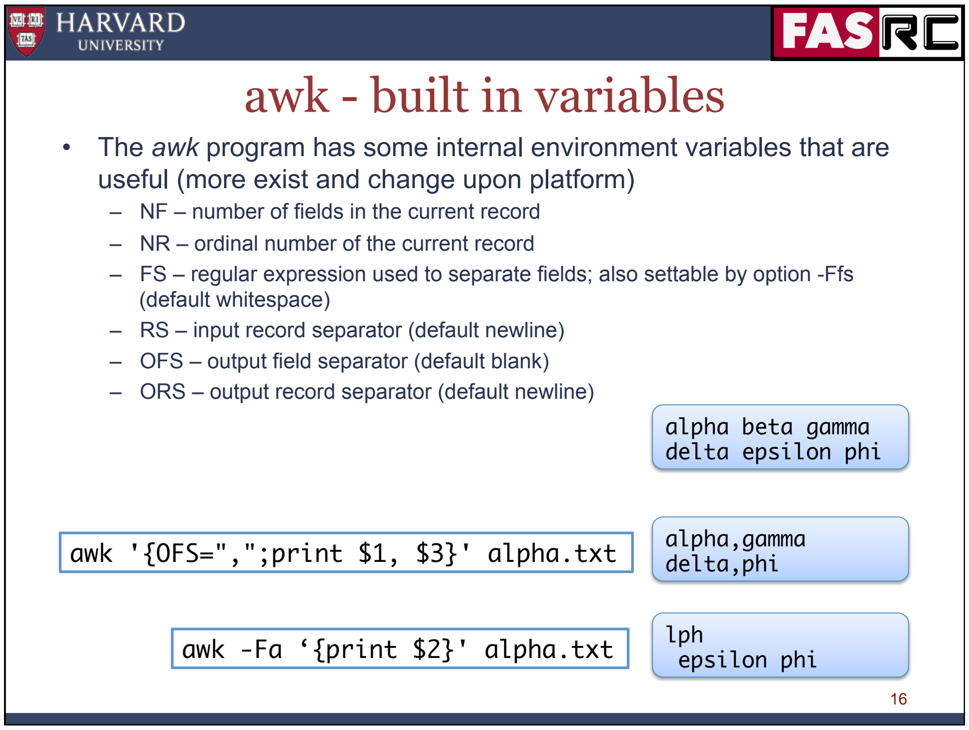
```
alpha beta gamma
delta epsilon phi
```

```
awk '{print $1}' alpha.txt
```

```
alpha
delta
```

```
awk '{print $1, $3}' alpha.txt
```

```
alpha gamma
delta phi
```



awk - built in variables

- The *awk* program has some internal environment variables that are useful (more exist and change upon platform)
  - NF – number of fields in the current record
  - NR – ordinal number of the current record
  - FS – regular expression used to separate fields; also settable by option -Ffs (default whitespace)
  - RS – input record separator (default newline)
  - OFS – output field separator (default blank)
  - ORS – output record separator (default newline)

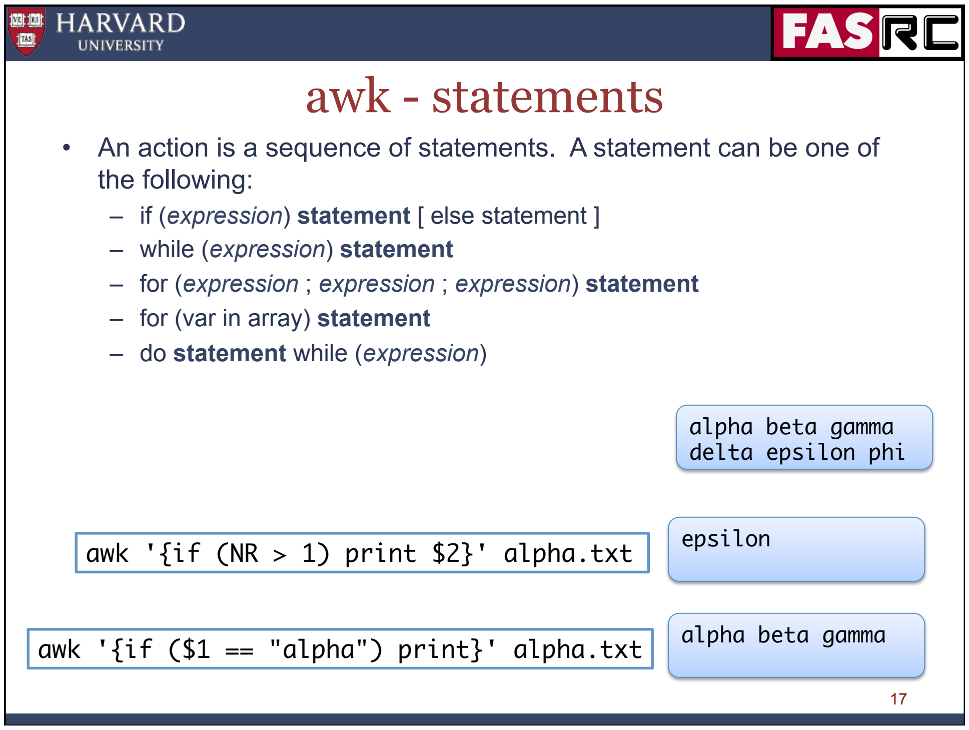
```
alpha beta gamma
delta epsilon phi
```

```
awk '{OFS=",";print $1, $3}' alpha.txt
```

```
alpha,gamma
delta,phi
```

```
awk -Fa '{print $2}' alpha.txt
```

```
lph
 epsilon phi
```



HARVARD UNIVERSITY FASRC

awk - statements

- An action is a sequence of statements. A statement can be one of the following:
  - if (*expression*) **statement** [else statement]
  - while (*expression*) **statement**
  - for (*expression* ; *expression* ; *expression*) **statement**
  - for (var in array) **statement**
  - do **statement** while (*expression*)

```
alpha beta gamma
delta epsilon phi
```

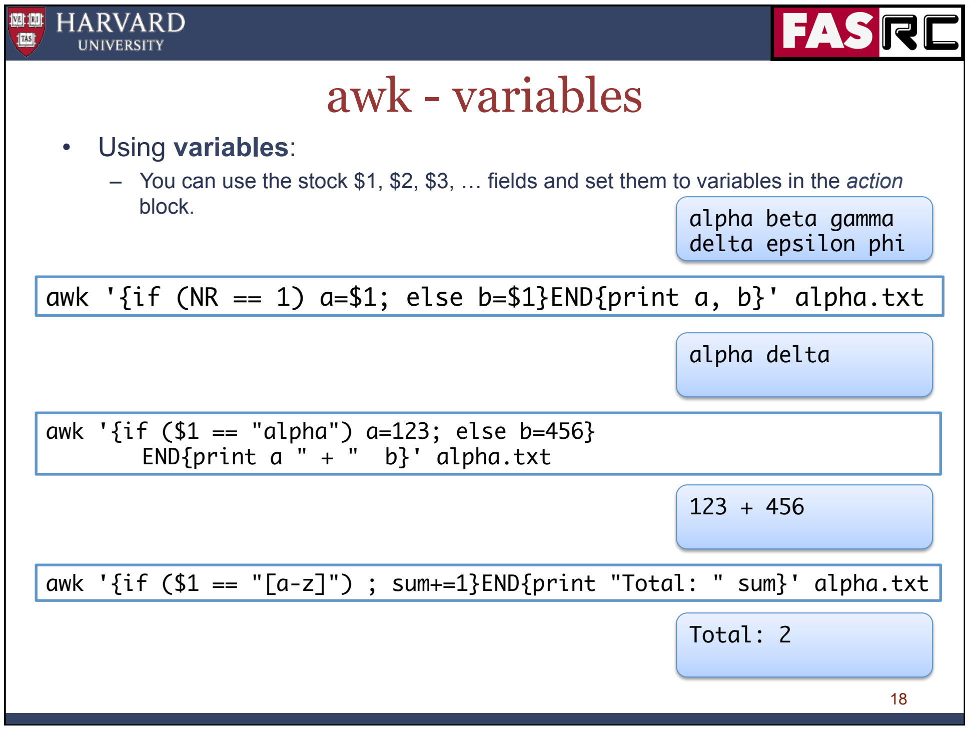
```
awk '{if (NR > 1) print $2}' alpha.txt
```

```
epsilon
```

```
awk '{if ($1 == "alpha") print}' alpha.txt
```

```
alpha beta gamma
```

17



HARVARD UNIVERSITY FASRC

awk - variables

- Using **variables**:
  - You can use the stock $1, $2, $3, … fields and set them to variables in the *action* block.

```
alpha beta gamma
delta epsilon phi
```

```
awk '{if (NR == 1) a=$1; else b=$1}END{print a, b}' alpha.txt
```

```
alpha delta
```

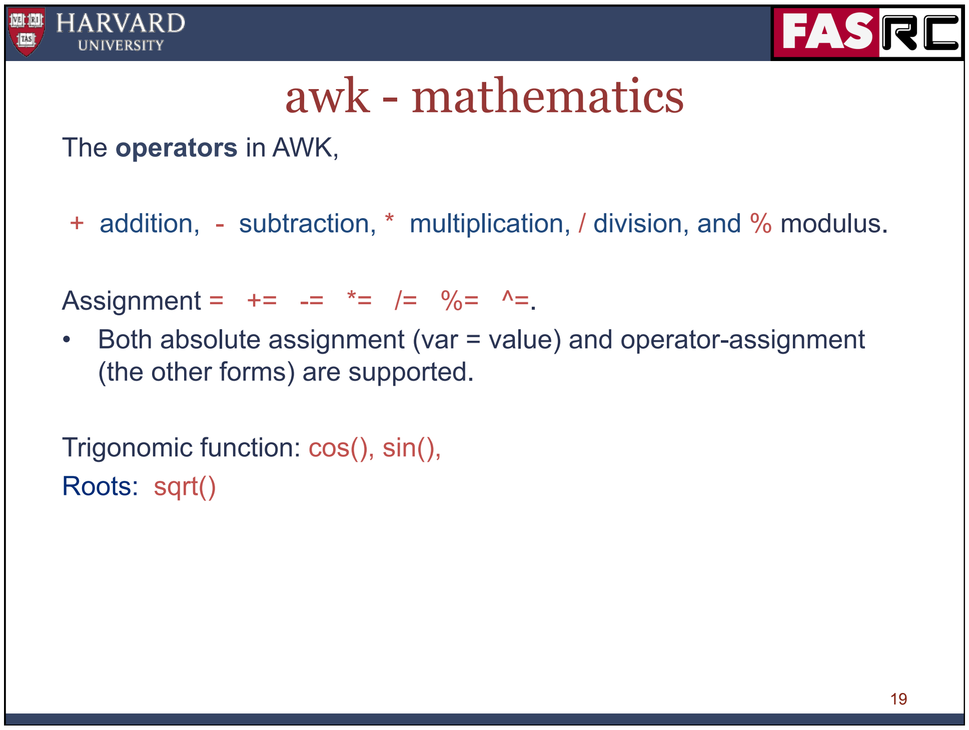
```
awk '{if ($1 == "alpha") a=123; else b=456}
        END{print a " + "  b}' alpha.txt
```

```
123 + 456
```

```
awk '{if ($1 == "[a-z]") ; sum+=1}END{print "Total: " sum}' alpha.txt
```

```
Total: 2
```

18



HARVARD UNIVERSITY FASRC

awk - mathematics

The **operators** in AWK,

+ addition, - subtraction, * multiplication, / division, and % modulus.

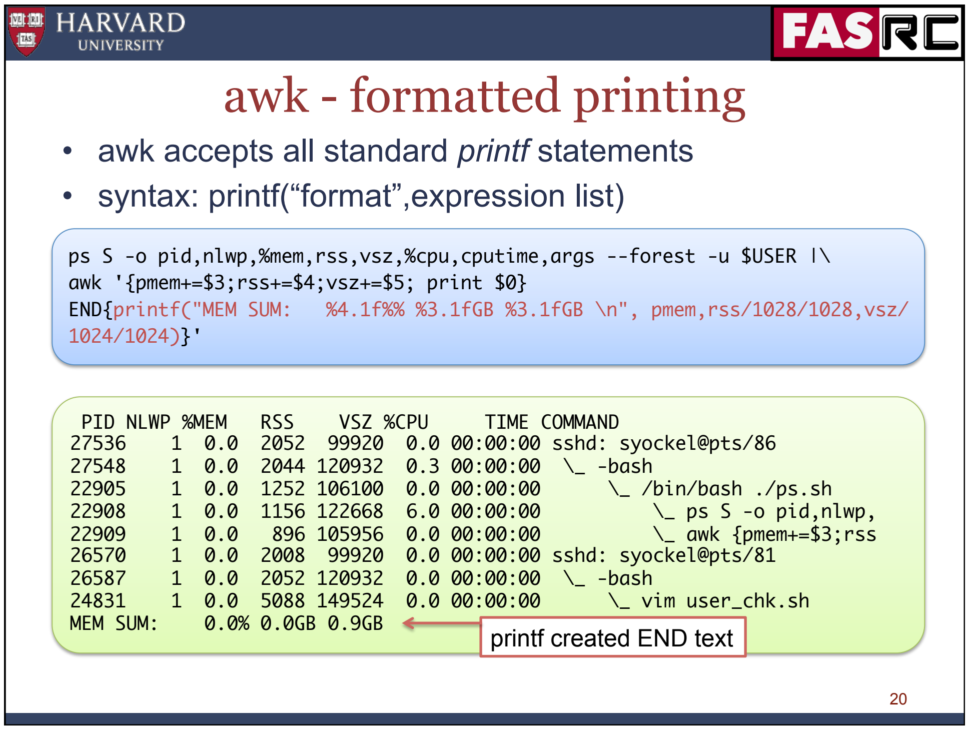
Assignment = += -= *= /= %= ^=.

- Both absolute assignment (var = value) and operator-assignment (the other forms) are supported.

Trigonomic function: cos(), sin(),

Roots: sqrt()

19



HARVARD UNIVERSITY FASRC

awk - formatted printing

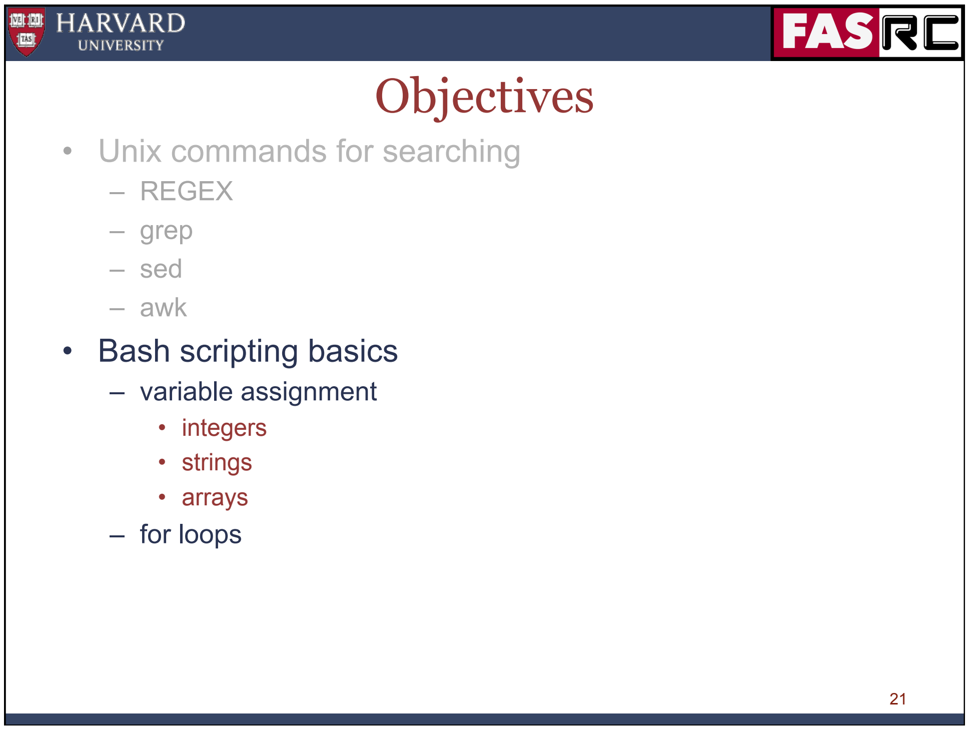
- awk accepts all standard *printf* statements
- syntax: printf("format",expression list)

```
ps S -o pid,nlwp,%mem,rss,vsz,%cpu,cputime,args --forest -u $USER |\
awk '{pmem+=$3;rss+=$4;vsz+=$5; print $0}
END{printf("MEM SUM:   %4.1f%% %3.1fGB %3.1fGB \n", pmem,rss/1028/1028,vsz/
1024/1024)}'
```

```
  PID NLWP %MEM   RSS    VSZ %CPU     TIME COMMAND
27536    1  0.0  2052  99920  0.0 00:00:00 sshd: syockel@pts/86
27548    1  0.0  2044 120932  0.3 00:00:00  \_ -bash
22905    1  0.0  1252 106100  0.0 00:00:00      \_ /bin/bash ./ps.sh
22908    1  0.0  1156 122668  6.0 00:00:00          \_ ps S -o pid,nlwp,
22909    1  0.0   896 105956  0.0 00:00:00          \_ awk {pmem+=$3;rss
26570    1  0.0  2008  99920  0.0 00:00:00 sshd: syockel@pts/81
26587    1  0.0  2052 120932  0.0 00:00:00  \_ -bash
24831    1  0.0  5088 149524  0.0 00:00:00      \_ vim user_chk.sh
MEM SUM:     0.0% 0.0GB 0.9GB
```

printf created END text

20

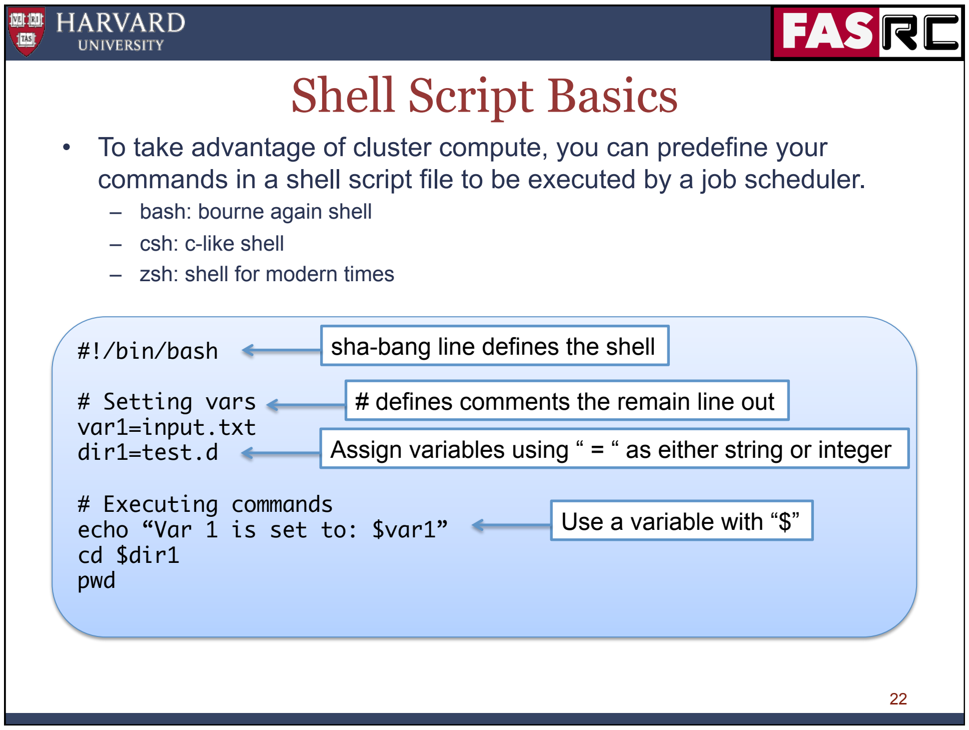


HARVARD UNIVERSITY FASRC

Objectives

- Unix commands for searching
  - REGEX
  - grep
  - sed
  - awk
- Bash scripting basics
  - variable assignment
    - integers
    - strings
    - arrays
  - for loops

21



HARVARD UNIVERSITY FASRC

Shell Script Basics

- To take advantage of cluster compute, you can predefine your commands in a shell script file to be executed by a job scheduler.
  - bash: bourne again shell
  - csh: c-like shell
  - zsh: shell for modern times

```
#!/bin/bash

# Setting vars
var1=input.txt
dir1=test.d

# Executing commands
echo “Var 1 is set to: $var1”
cd $dir1
pwd
```

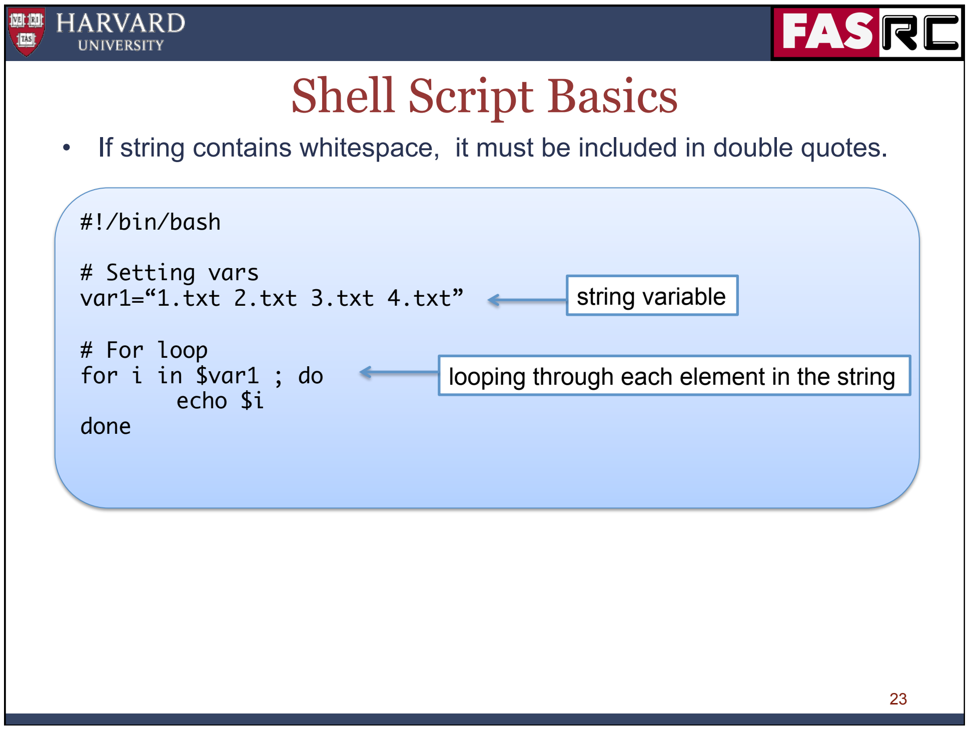
sha-bang line defines the shell

# defines comments the remain line out

Assign variables using “ = “ as either string or integer

Use a variable with “$”

22



HARVARD UNIVERSITY FASRC

Shell Script Basics

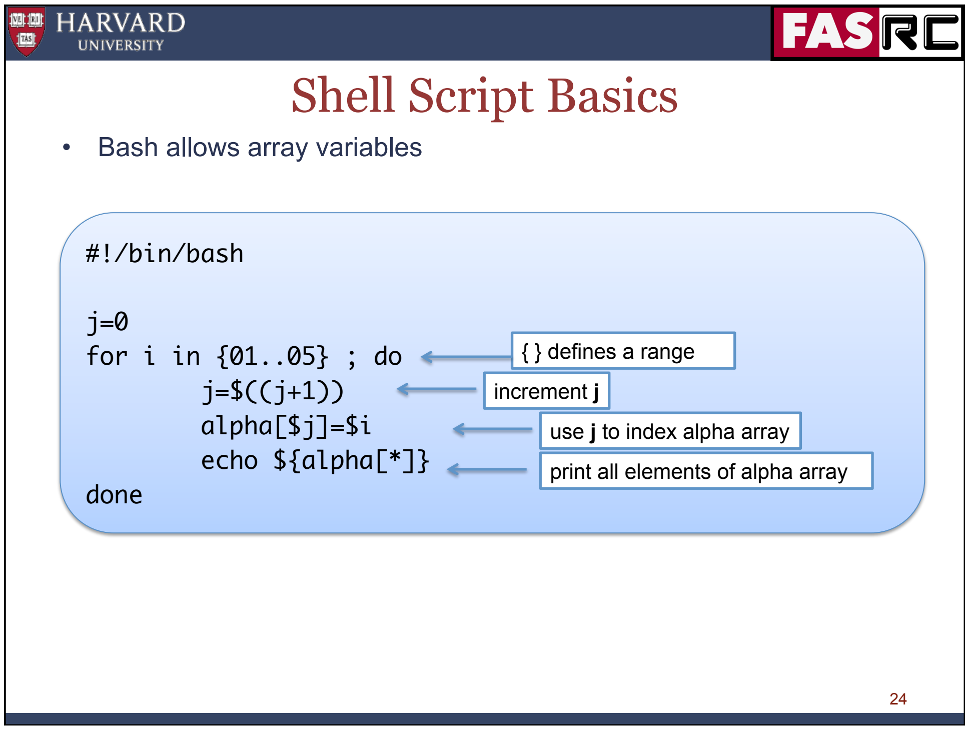
- If string contains whitespace, it must be included in double quotes.

```
#!/bin/bash

# Setting vars
var1=“1.txt 2.txt 3.txt 4.txt”

# For loop
for i in $var1 ; do
       echo $i
done
```

string variable

looping through each element in the string

23



HARVARD UNIVERSITY FASRC

Shell Script Basics

- Bash allows array variables

```
#!/bin/bash

j=0
for i in {01..05} ; do
        j=$((j+1))
        alpha[$j]=$i
        echo ${alpha[*]}
done
```

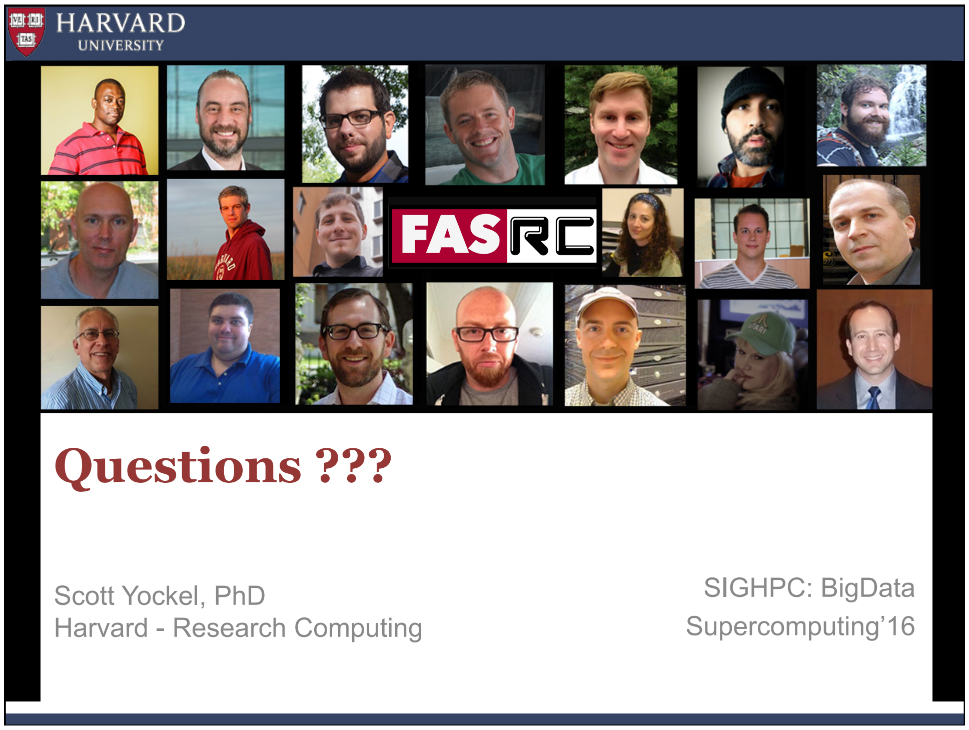
{ } defines a range

increment **j**

use **j** to index alpha array

print all elements of alpha array

24



HARVARD UNIVERSITY

Questions ???

Scott Yockel, PhD
Harvard - Research Computing

SIGHPC: BigData
Supercomputing'16