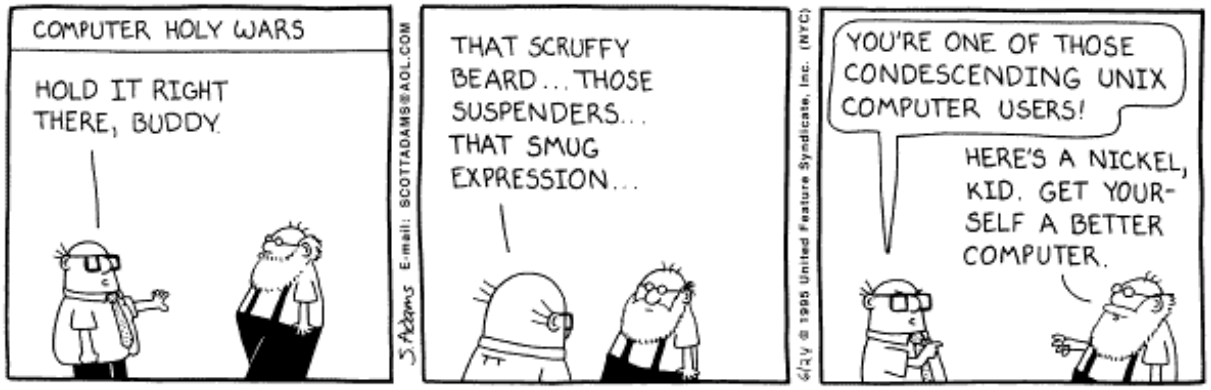


Unix 1

Basics, setups, some useful commands, and scripting

''Unix is user friendly --
It's just picky about who it's friends
are...''

Background: UNIX

- It's an operating system
  - OS – tells a computer how to operate, like Windows, iOS, etc. Macs have unix as their foundation, Linux machines run linux, which is a close approximation to unix)
- Developed in late 1960s at Bell Labs (linked to AT&T)
- Basic philosophy stresses use of fairly simple programs or tools to do one thing, then take output of that tool into another tool
  - Means that the tools are pretty simple (so less buggy) and can be joined together to do more complicated things
- So why doesn't everyone use UNIX?
  - Much steeper learning curve…..
  - Command-line interface and fairly cryptic program names

You've already worked with UNIX

- Initial setup lab had you exploring files, directories, and moving around in a structure
- You've also worked with unix in all of your command-line operations outside of the python windows

- Today will go through some background on UNIX, setups and environments, a few useful commands, and scripting

UNIX setup

- When you start up your computer, it will read specific files that define several important setup variables
- Most you don’t need to adjust, but some key ones you should know about
  - Defining the shell
  - Environment variables
  - aliases

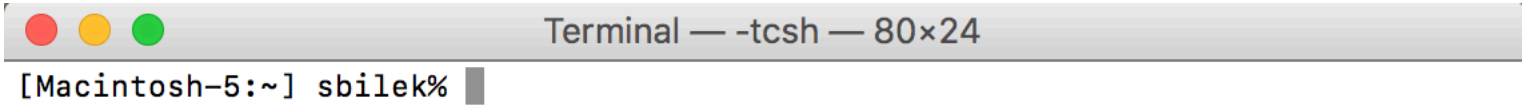
Shells

- User interface for the UNIX operating system
  - Interprets what you type
- Controls the syntax of what you type at the command line

- Variety of shells available
  - sh (Bourne shell)
  - bash (Bourne Again shell) - often default on linux and Mac
  - csh (C shell), tcsh (tenex C shell - very similar to csh)
    - Both similar to the C programming language in syntax
  - bash and tcsh are the ones most commonly used in 345 setups

What shell do you use?

- Check the top of your terminal window (usually tells you)



- In a terminal window:

  >> env $SHELL #this will return your login shell to the screen

- You can change shells in any window
  - Just type the shell name at the prompt
- Which one should I use?
  - User preference – pick one and learn its syntax

UNIX Environment

- The environment setup is important – defines specific environment variables for the OS so that each time you log in, you get the same behavior
- How to see what your environment variables are?

  >> env

```
[Macintosh-5:NMTCourses/GEOP501/W6_DATA_STRUCT] sbilek% env
TMPDIR=/var/folders/45/23lkm3w93w78qt5ryt4wc3r00000gn/T/
TERM_PROGRAM_VERSION=388.1.1
Apple_PubSub_Socket_Render=/private/tmp/com.apple.launchd.J0a76irDKJ/Render
LANG=en_US.UTF-8
TERM_PROGRAM=Apple_Terminal
XPC_SERVICE_NAME=0
XPC_FLAGS=0x0
DISPLAY=/private/tmp/com.apple.launchd.W9zixbHfq6/org.macosforge.xquartz:0
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.6ybeY0xnPK/Listeners
TERM=xterm-256color
TERM_SESSION_ID=143C1932-AA06-48CE-A435-814ACDA6C87C
__CF_USER_TEXT_ENCODING=0x1F5:0x0:0x0
SHELL=/bin/bash
HOME=/Users/sbilek
LOGNAME=sbilek
USER=sbilek
PATH=/Users/sbilek/anaconda/bin:/usr/local/GMT4.4.0/bin:/Developer/Tools/:/Users/sbilek/bin
:/usr/bin:/usr/local/bin:/usr/local/sac/bin:/usr/X11R6/bin:/Applications/Absoft10/bin:/sw/b
in:/opt/passcal/bin:/Users/sbilek/PROGRAMS/sod/bin:/Users/sbilek/PROGRAMS/PROGRAMS.330/bin:
/Users/sbilek/PROGRAMS/Taup/TauP-2.1.0/bin:.:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/
opt/X11/bin:/usr/local/bin:/Users/sbilek/anaconda/bin:/opt/passcal/bin:/opt/passcal/other/b
in
HOSTTYPE=unknown
VENDOR=apple
OSTYPE=darwin
MACHTYPE=x86_64
SHLVL=1
PWD=/Users/sbilek/TEACHING/NMTCourses/GEOP501/W6_DATA_STRUCT
GROUP=staff
HOST=Macintosh-5.local
TAUP_HOME=/Users/sbilek/PROGRAMS/Taup/TauP-2.1.0
ABSOFT=/Applications/Absoft10
PASSCAL=/opt/passcal
PASSOFT=/opt/passcal
MANPATH=/usr/share/man:/usr/local/GMT4.4.0/man:/opt/passcal/man
NETCDFHOME=/usr/local/netcdf-3.6.3
SACAUX=/usr/local/sac/aux
GREENDIR=/Users/sbilek/PROGRAMS/PROGRAMS.330/GREEN
PERL5LIB=/Users/sbilek/libperl
CC=/Applications/Xcode.app/Contents/Developer/usr/bin/gcc
```

PATH

- PATH: tells the shell where to find applications or executable files

```
PATH=/Users/sbilek/anaconda/bin:/usr/local/GMT4.4.0/bin:/Developer/Tools/:/Users/sbilek/bin
:/usr/bin:/usr/local/bin:/usr/local/sac/bin:/usr/X11R6/bin:/Applications/Absoft10/bin:/sw/b
in:/opt/passcal/bin:/Users/sbilek/PROGRAMS/sod/bin:/Users/sbilek/PROGRAMS/PROGRAMS.330/bin:
/Users/sbilek/PROGRAMS/Taup/TauP-2.1.0/bin:.:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/
opt/X11/bin:/usr/local/bin:/Users/sbilek/anaconda/bin:/opt/passcal/bin:/opt/passcal/other/b
in
```

- : separates each full path name

- When you call a command, the shell will search through your list of paths, in order, until it finds the first occurrence to use
  - That first one it finds will be the one it uses
  - To force use of an executable at a different location, you need to define the full path on the command line

>> /Users/sbilek/sac # will use the program sac in my home directory rather than the one defined in my path at /usr/local/sac/bin/sac

How to modify environment (and other) variables

- For single use:
  - tcsh >> setenv PATH {$PATH}:/newloc
  - bash >> PATH=$PATH:/newloc

- For single use:
  - tcsh: >> set history = 1000 #saves last 1000 commands in the history list
  - bash: >> history=1000

But this gets old, having to type this stuff in to every terminal window…. Luckily there is a file for that!

Modifying the default environment

- You can make changes to your environment so that each time you log in/start a new window, it contains your specific values
- These are contained in a “dot” file
  - these are typically invisible to the user when doing ls commands, live in your home directory
  - We’ll focus on the shell configuration file (.tcshrc or .profile)

Example for .tcshrc

```
#!/bin/tcsh

setenv TAUP_HOME /Users/sbilek/PROGRAMS/Taup/TauP-2.1.0
set path = ( ~/anaconda/bin /usr/local/GMT4.4.0/bin /Developer/Tools/ ~/bin /usr/bin /usr/l
ocal/bin /usr/local/sac/bin /usr/X11R6/bin /Applications/Absoft10/bin /sw/bin /opt/passcal/
bin ~/PROGRAMS/sod/bin /Users/sbilek/PROGRAMS/PROGRAMS.330/bin ${TAUP_HOME}/bin . $path /us
r/local/bin ~/anaconda/bin )
#add to path for gmt5 /Applications/GMT-5.4.2.app/Contents/Resources/bin
set noclobber
setenv ABSOFT /Applications/Absoft10
#setenv PASSCAL /opt/passcal/bin
setenv PASSCAL /opt/passcal
setenv PASSOFT ${PASSCAL}
source ${PASSCAL}/setup/setup.csh

setenv MANPATH  /usr/share/man:/usr/local/GMT4.4.0/man:$MANPATH

setenv LD_LIBRARY_PATH /Applications/Absoft10/lib/
setenv NETCDFHOME /usr/local/netcdf-3.6.3
#setenv SACAUX /usr/local/sac101.5c/aux
setenv SACAUX /usr/local/sac/aux
#setenv SAC_PPK_LARGE_CROSSHAIRS 1
#setenv PYTHONPATH /Users/sbilek/PROGRAMS/PSU_Finite_Fault_Codes/CommandLineTools/PySac_2.0
.2

setenv GREENDIR /Users/sbilek/PROGRAMS/PROGRAMS.330/GREEN
setenv PERL5LIB /Users/sbilek/libperl
#setenv VERSIONER_PERL_VERSION 5.12
setenv CC /Applications/Xcode.app/Contents/Developer/usr/bin/gcc

alias c 'clear'
alias cp 'cp -i'
alias mv 'mv -i'
alias rm 'rm -f'
alias l 'ls -F'
alias ll 'ls -l'
alias .. 'cd ..'
alias . 'echo $cwd'
alias fle "perl -pi -e 's/\r/\n/g' "
#alias sac '/usr/local/sac101.5c/bin/sac ~/macros/init.m'
alias sac '/usr/local/sac/bin/sac ~/macros/init.m'
~
~
~
~
~
~
~
~
~
~
~
"~/.tcshrc" 37L, 1459C
```

For .profile:

PATH=/usr/local/bin:/usr/bin:/usr/local/GMT/bin

export PATH

Example for .tcshrc

```
#!/bin/tcsh

setenv TAUP_HOME /Users/sbilek/PROGRAMS/Taup/TauP-2.1.0
set path = ( ~/anaconda/bin /usr/local/GMT4.4.0/bin /Developer/Tools/ ~/bin /usr/bin /usr/l
ocal/bin /usr/local/sac/bin /usr/X11R6/bin /Applications/Absoft10/bin /sw/bin /opt/passcal/
bin ~/PROGRAMS/sod/bin /Users/sbilek/PROGRAMS/PROGRAMS.330/bin ${TAUP_HOME}/bin . $path /us
r/local/bin ~/anaconda/bin )
#add to path for gmt5 /Applications/GMT-5.4.2.app/Contents/Resources/bin
set noclobber
setenv ABSOFT /Applications/Absoft10
#setenv PASSCAL /opt/passcal/bin
setenv PASSCAL /opt/passcal
setenv PASSOFT ${PASSCAL}
source ${PASSCAL}/setup/setup.csh

setenv MANPATH  /usr/share/man:/usr/local/GMT4.4.0/man:$MANPATH

setenv LD_LIBRARY_PATH /Applications/Absoft10/lib/
setenv NETCDFHOME /usr/local/netcdf-3.6.3
#setenv SACAUX /usr/local/sac101.5c/aux
setenv SACAUX /usr/local/sac/aux
#setenv SAC_PPK_LARGE_CROSSHAIRS 1
#setenv PYTHONPATH /Users/sbilek/PROGRAMS/PSU_Finite_Fault_Codes/CommandLineTools/PySac_2.0
.2

setenv GREENDIR /Users/sbilek/PROGRAMS/PROGRAMS.330/GREEN
setenv PERL5LIB /Users/sbilek/libperl
#setenv VERSIONER_PERL_VERSION 5.12
setenv CC /Applications/Xcode.app/Contents/Developer/usr/bin/gcc

alias c 'clear'
alias cp 'cp -i'
alias mv 'mv -i'
alias rm 'rm -f'
alias l 'ls -F'
alias ll 'ls -l'
alias .. 'cd ..'
alias . 'echo $cwd'
alias fle "perl -pi -e 's/\r/\n/g' "
#alias sac '/usr/local/sac101.5c/bin/sac ~/macros/init.m'
alias sac '/usr/local/sac/bin/sac ~/macros/init.m'
~
~
~
~
~
~
~
~
~
~
~
"~/.tcshrc" 37L, 1459C
```

Aliases are awesome!

Can be used in a terminal window for short term use or set up here for continual use

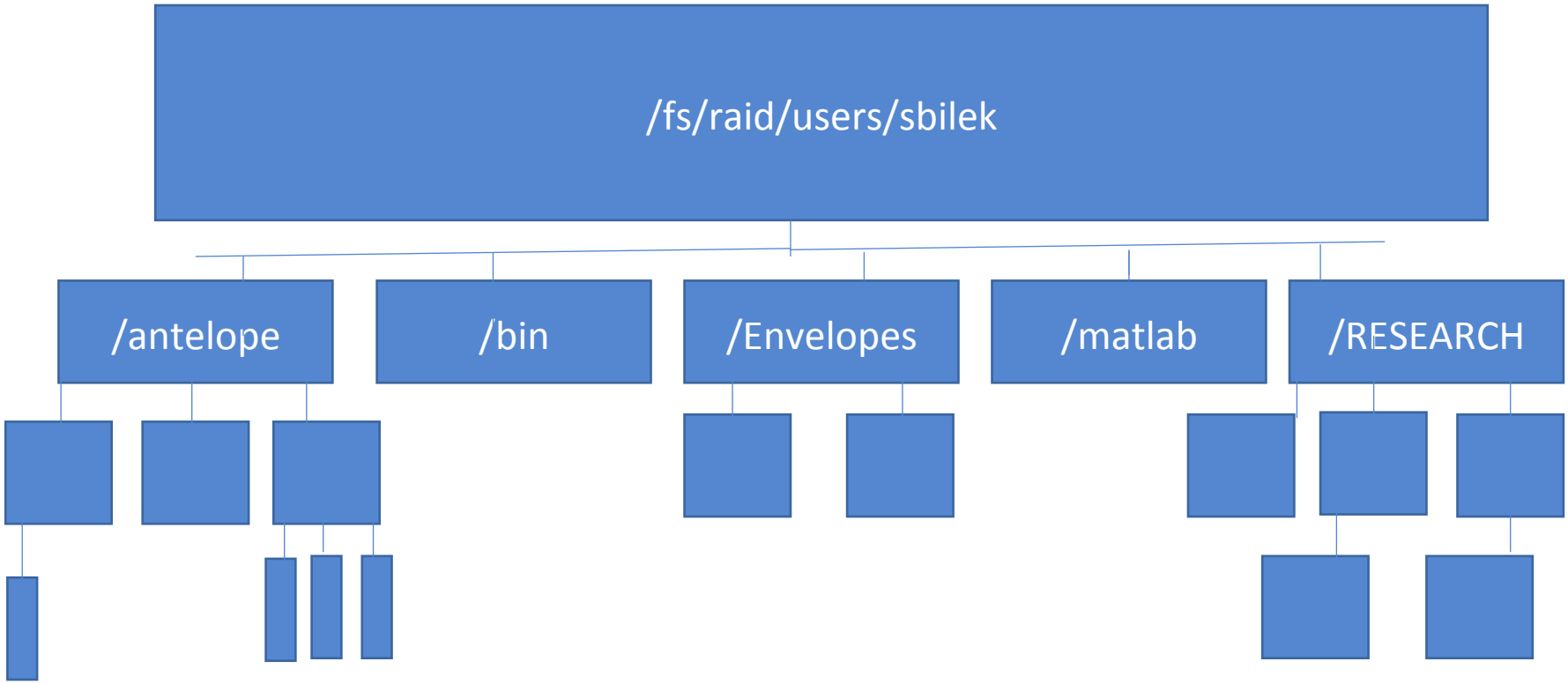
Force execution of config files: source

- If you modify your configuration files and want those changes to be implemented in your current terminal window, need to execute the file

- >> source ~/.tcshrc

- Note that if you just open new terminal windows, these changes will automatically take effect

Directory Structure



Working down from home directory – subdirectories containing other directories and files.
Move around using 'cd' -- change directory, 'pwd' - check current directory

'.' – indicates current directory '..' indicates the directory directly above, '~' indicates home directory

Other useful commands - directories

- mkdir name : make a directory
- rmdir name : remove a directory if it is empty
- rm name : removes file **** be careful
- rm −r name : removes directory **** be careful
- cp : copy files
- cp −r : copy directory and all files/etc inside it
- mv : move files or directories

Other useful commands

- ls : list everything in the current directory
- man cmd : bring up a manual page for the command ‘cmd’
- more : opens a text file for viewing, use space to scroll
- head −nX : displays the first X lines in a file
- tail −nX : displays the last X lines in a file
- wc : line, word, and character count in a file
- sort : sort function (handles both letters and numbers)

Wildcards

- * and ? Can be used as wildcards, very useful for being able to match filenames
  - * match any number of characters
  - ? Match 1 character

```
>>> cp station_list* ./INPUTS
```

(copies station_list1.dat, station_list10.dat, station_list2.dat, station_list3.dat, station_list.txt to directory ./INPUTS)

```
>>> cp station_list1?.dat ./INPUTS
```

(copies only station_list10.dat to ./INPUTS)

- [] : placeholder that holds a range of characters or numbers

```
>>> cp station_list[1-3]* ./INPUTS
```

(copies station_list1.dat, station_list2.dat, station_list3.dat to ./INPUTS)

Command line redirection/piping

- | (vertical line): pipe
  - Uses the output from a command on the left side of pipe as input into in the right side of the pipe

```
>>> ls ./Seismograms | head −n5
BAR.EHZ
CAR.EHZ
CBET.EHZ
DAG.EHZ
MLM.EHZ
```

Command line redirection/piping

- > : redirect screen output to a defined file

```
>>> ls ./Seismograms | head −n5 > station_list.txt
>>> more station_list.txt
    BAR.EHZ
    CAR.EHZ
    CBET.EHZ
    DAG.EHZ
    MLM.EHZ
```

- Be careful with redirects in tcsh (will not overwrite an existing file, will give warning instead, need to use >! to overwrite) and bash (will overwrite a file with no warning)

- >> : concatenate screen output to end of an existing file

```
>>> ls ./Seismograms | tail −n1 >> station_list.txt
>>> more station_list.txt
    BAR.EHZ
    CAR.EHZ
    CBET.EHZ
    DAG.EHZ
    MLM.EHZ
    WTX.EHZ
```

Shell scripting

- What is it? Program using shell commands (like you would use in on the command line in the shell)
- Similar to python scripts you've been writing so far
- Useful to capture series of commands that accomplish a task and allow you to repeat your work
- Also fairly portable

Simple shell script

Script that

- copies some files into a directory and uses loops to
- sort a list of values and grabs out the last line to write to a new file,
- run a perl script with output going to a defined file, moving files to new directories

```
#!/bin/csh

mkdir DATA_FILES
cp /Users/sbilek/RESEARCH/MEXICO_PROJECT/swarms/NEEDED_FILES_FOR_SPEC_RAT/get_cluster_wfdisc.csh .
cp /Users/sbilek/RESEARCH/MEXICO_PROJECT/swarms/NEEDED_FILES_FOR_SPEC_RAT/station.txt .
cp /Users/sbilek/RESEARCH/MEXICO_PROJECT/swarms/NEEDED_FILES_FOR_SPEC_RAT/get_parameters.csh .
cp /Users/sbilek/RESEARCH/MEXICO_PROJECT/swarms/NEEDED_FILES_FOR_SPEC_RAT/runenvdel_*pl .

rm tmp*
cp *dat ./DATA_FILES


@ i = 1
while ($i <= 10)
  sort -k 2,2 moments.$i.dat | tail -1 > moments.${i}_used
@ i +=1
end


cp *used ./DATA_FILES


@ i = 1
while ($i <= 10)
  echo "working on cluster " $i
  perl runenvdel_${i}.pl > test_${i}.out
  mv test_$i.out ./Cluster$i
@ i +=1
end
```

Allows me to run a large set of calculations without me having to sit and wait for each run to finish and move files around

Shell scripting

- Some of the python scripts we've already done can be re-written in shell scripts
  - Key differences in syntax between python and shell (and between different shells)
  - Go back to "Flow control" lecture for some specific examples/comparisons
- Specific UNIX constructs exist
  - For example, use of ` ` (back or accent quotes) – command substitution
    - Tells the shell to run what is inside the back quotes and put the output of the command back into the quotes

Variables in shell scripts

- Used to store information (character, string, value)
- Can define it using = and return it using $

- csh

```
>>> set b = ”This is a test”
>>> set a = `echo $b | wc`
>>> echo $a
1 4 15   (1 line, 4 words, 15 characters)
```

bash

```
>>> b=“This is a test”
>>> a=`echo $b | wc`
>>> echo $a
1 4 15
```

Note the subtle differences: set in csh, no spaces around = in bash
Also note use of the command substitution in back quotes

Quotes

- ‘…’ : single quotes forces literal interpretation of whatever is the the quotes
- “…” : double quotes group words/characters together, but escape characters, variables are still recognized (so can be an issue with $ for example)

- csh

```
>>> set b = “Hello world $”
>>> echo $b
Illegal variable name
>>> set b = ‘Hello world $’
>>> echo $b
Hello world $
```

bash

```
>>> b=“Hello world $”
>>> echo $b
Illegal variable name
>>> b=‘Hello world $’
>>> echo $b
Hello world $
```

Basic calculations - Arithmetic

- Shell arithmetic is integer only,
    - + : addition
    - - : subtraction
    - * : multiplication
    - / : division
    - % : remainder or modulus

- bash: $(( )) used to calculate expressions

```
>>> echo $(( 10/3 ))
3
```

- csh:

Assignment Operators

- = : set variable equal to value on the right
- += : set variable to itself plus value on the right
- -= : set variable to itself minus value on the right
- *= : set variable to itself times value on the right
- /= : set variable to itself divided by value on the right
- %= : set variable equal to remainder of itself divided by the value on the right

Relational Operators

- All relational operators are left to right associative
  - < : test for less than
  - <= : test for less than or equal to
  - > : test for greater than
  - >= : test for greater than or equal to
  - == : test for equal to
  - != : test for not equal

Logical Operators

- Logical operators return 1 for true and 0 for false
  - && : logical AND; tests that both expressions are true
    - left to right associative

    %echo $(((3 < 4) && (10<15)))

    1

    %echo $(((3<4) && (10>15)))

    0
  - || : logical OR ; tests that one or both of the expressions are true
    - left to right associative

    %echo $(((3<4) || (10>15)))

    1
  - ! : logical negation; tests that expression is true

Testing

- useful flags return true (0) if
    - -d : expression is a directory
    - -f : expression is a regular file
    - -e : expression is any type of file
    - -w : expression is a writable file
    - -x : expression is an executable (file or directory)
    - -n : expression is a nonzero length string
    - -z : expression is a zero length string
- Particularly useful to test for this (in if statements) at start of codes to make sure files exist, produce error message if not.

Flow control

- if/then/endif
- if/then/else/endif
- if/then/elseif/endif
- do/done (in bash, used in for, while loops)
- For loop: iterates over an array of objects
- While loop: continues to loop as long as condition is met
- Foreach (tcsh command): allows for iteration over files

```
>>>
foreach file (*.BHZ)
   echo $file
   cp $file $file_copy.BHZ
end
```

Reading command line arguments

- Shell scripts can take inputs from the command line

- Script: name.sh

Terminal — vi name.sh — 80×24

```
#!/bin/bash
echo "Hi there.  My name is $HOST.  Who are you?"
echo "Nice to meet you $1."
~
```

- >>> ./name.sh Sue

```
Hi there.  My name is Macintosh-5.local.  Who are you?
Nice to meet you Sue.
```

Next time:

- Live shell scripting
- 2 weeks from now: more UNIX using other powerful tools like grep, awk, sed