GEOS 436 / 636
Programming and Automation for Geoscientists

– Week 08: Unix – Shell –

Ronni Grapenthin
rgrapenthin@alaska.edu
Elvey 413B
x7682

Is there a computer environment that makes it *easy* to automate processes??*

* hint: YES!!!!

Unix Philosophy

- minimalist
- modular
- robust
- funky user interface (the shell)

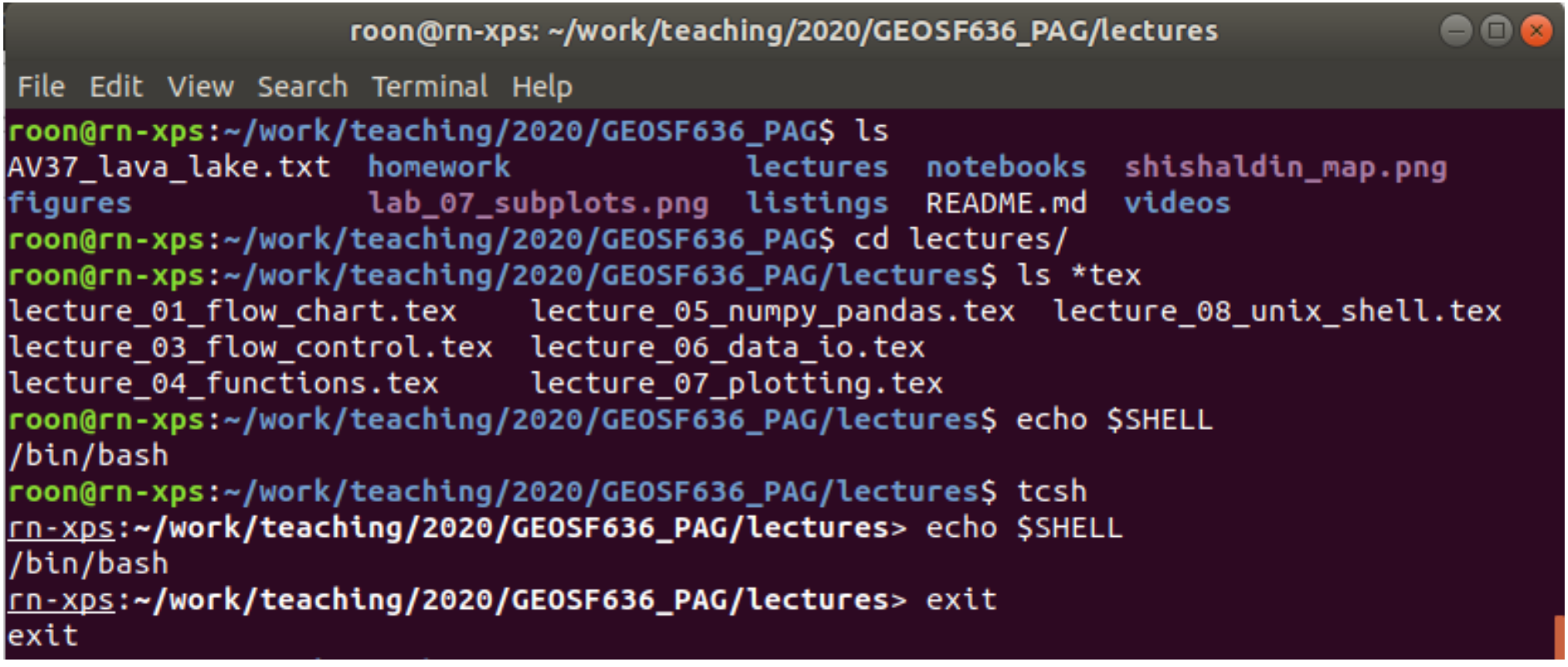
Peter Salus:

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

How do I get it?

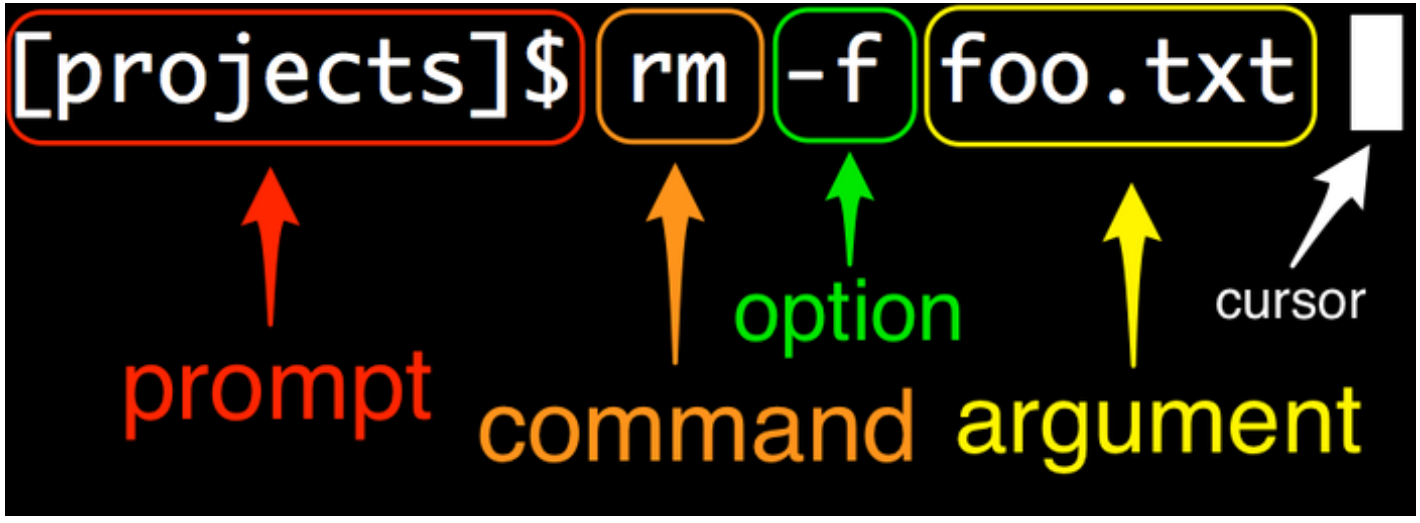
- Unix/Linux - comes with the package (Linux is a free Unix)
- Mac OS - derived from Unix; use Terminal.app
- Windows - various emulators (e.g., VirtualBox)
- We'll use a browser-based interface (JupyterLab) to ensure a homogeneous setup

Shell



- Text-based user interface
- Interpreter of your commands
- Most popular shells: `bash, (t)csh`
- Defines your working environment

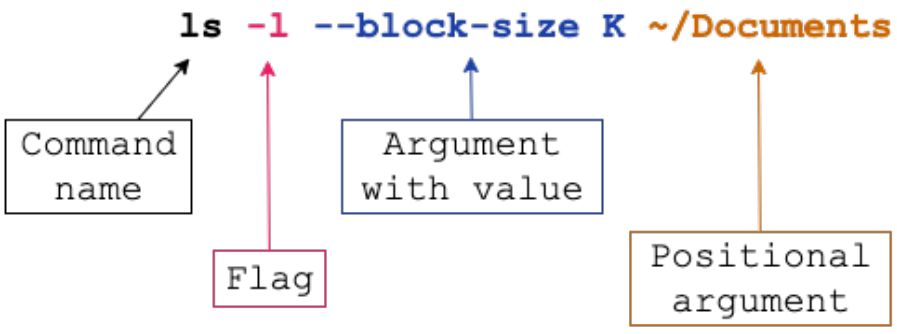
Shell Commandline Anatomy



https://www.learnenough.com/command-line-tutorial/basics

(prompt will differ)

Command Anatomy



https://betterdev.blog/command-line-arguments-anatomy-explained

Command Anatomy: man-pages

```
$>man mkdir
```

```
MKDIR(1)                                                User Commands

NAME
       mkdir - make directories

SYNOPSIS
       mkdir [OPTION]... DIRECTORY...

DESCRIPTION
       Create the DIRECTORY(ies), if they do not already exist.

       Mandatory arguments to long options are mandatory for short options too.

       -m, --mode=MODE
              set file mode (as in chmod), not a=rwx - umask

       -p, --parents
              no error if existing, make parent directories as needed

       -v, --verbose
              print a message for each created directory

       -Z     set SELinux security context of each created directory to the default type

       --context[=CTX]
              like -Z, or if CTX is specified then set the SELinux or SMACK security context to CTX

       --help display this help and exit

       --version
              output version information and exit

AUTHOR
       Written by David MacKenzie.
```

Command Anatomy: `--help` / `-h`

Built-in help often a sufficiently quick reminder:

```
roon@rn-xps:~/work/teaching/2020/GEOSF636_PAG/lectures$ mkdir --help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
  -m, --mode=MODE   set file mode (as in chmod), not a=rwx - umask
  -p, --parents     no error if existing, make parent directories as needed
  -v, --verbose     print a message for each created directory
  -Z                   set SELinux security context of each created directory
                         to the default type
      --context[=CTX]  like -Z, or if CTX is specified then set the SELinux
                         or SMACK security context to CTX
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
Full documentation at: <http://www.gnu.org/software/coreutils/mkdir>
or available locally via: info '(coreutils) mkdir invocation'
roon@rn-xps:~/work/teaching/2020/GEOSF636_PAG/lectures$
```

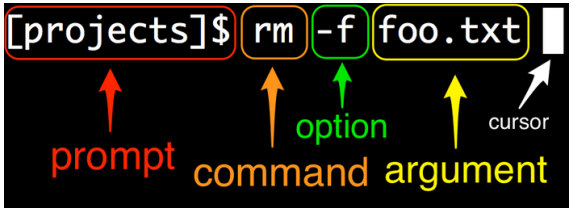
Environment

```
USERNAME=roon
XDG_VTNR=2
SSH_AUTH_SOCK=/run/user/1001/keyring/ssh
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
XDG_SESSION_ID=7
USER=roon
DESKTOP_SESSION=ubuntu
QT4_IM_MODULE=xim
TEXTDOMAINDIR=/usr/share/locale/
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/bc62458d_0cbf_40fc_811a_583b6b0564a9
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PWD=/home/roon/work/teaching/2020/GEOSF636_PAG/lectures
HOME=/home/roon
TEXTDOMAIN=im-config
SSH_AGENT_PID=2629
QT_ACCESSIBILITY=1
XDG_SESSION_TYPE=x11
XDG_DATA_DIRS=/usr/share/ubuntu:/home/roon/.local/share/flatpak/exports/share/:/var/lib/f
latpak/exports/share/:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop
XDG_SESSION_DESKTOP=ubuntu
GJS_DEBUG_OUTPUT=stderr
GTK_MODULES=gail:atk-bridge
WINDOWPATH=2
VTE_VERSION=5202
SHELL=/bin/bash
TERM=xterm-256color
QT_IM_MODULE=ibus
XMODIFIERS=@im=ibus
IM_CONFIG_PHASE=2
XDG_CURRENT_DESKTOP=ubuntu:GNOME
GPG_AGENT_INFO=/run/user/1001/gnupg/S.gpg-agent:0:1
GNOME_TERMINAL_SERVICE=:1.199
SHLVL=2
XDG_SEAT=seat0
GDMSESSION=ubuntu
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME=roon
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1001/bus
XDG_RUNTIME_DIR=/run/user/1001
XAUTHORITY=/run/user/1001/gdm/Xauthority
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
PATH=/home/roon/.local/bin:/home/roon/.local/bin:/home/roon/.local/bin:/home/roon/bin:/us
r/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sn
ap/bin
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
DESKTOP_MODE=1
SESSION_MANAGER=local/rn-xps:@/tmp/.ICE-unix/2516,unix/rn-xps:/tmp/.ICE-unix/2516
LESSOPEN=| /usr/bin/lesspipe %s
GTK_IM_MODULE=ibus
_=/usr/bin/env
```

- `env` command shows values of environment variables
- `PATH` probably the most important: specifies directory PATHs where shell tries to find programs

```
$> export PATH=/path/to/some/directory:$PATH (bash)
$> setenv PATH /path/to/some/directory:$PATH (tcsh)
```

Environment - The Path



https://www.learnenough.com/command-line-tutorial/basics

- Shell tries to execute what you type as a command
- Shell maintains a list of all locations of **executable** programs
- `PATH` specifies locations of where to find commands / programs
- Shell traverses this list in order. If you got two programs with the same name in different directories, only the one in the first directory will be called.
- Path can be permanently set in config file for shell, e.g. `.bashrc` or `.tcshrc` in home directory

```
$> export PATH=/path/to/some/directory:$PATH (bash)
$> setenv PATH /path/to/some/directory:$PATH (tcsh)
```

Directories

```
roon@rn-xps:~/work/RESEARCH/PROJECTS/valles$ tree
.
├── 2018_RESESS_intern_mentor_app.docx
├── literature
│   └── LANL_seismic_network_NMGS_abstract.pdf
├── maps
│   ├── sites.xy
│   ├── Valles.kmz
│   ├── Valles_sites_newman_copy.png
│   └── Valles_sites_newman_edit.png
└── permits
    ├── Application_2016.pdf
    ├── Grapenthin, Ronni - NMT - VALL-2016-SCI-0023.pdf
    └── Grapenthin, Ronni - Permit VALL-2018-SCI-0016.pdf

3 directories, 9 files
roon@rn-xps:~/work/RESEARCH/PROJECTS/valles$ ls
2018_RESESS_intern_mentor_app.docx  literature  maps  permits
```

- Files and directories are organized in a hierarchial structure
- Need to specify names by typing them (no clicking!), tab-completion is useful
- Case-sensitive: `case` ≠ `CaSe` ≠ `CASE`
- Special directory symbols:
  - `.` (the current directory)
  - `..` (the parent directory - one level up in tree)
  - `~` (the home directory)

Directory / File Navigation Commands

• `ls`	list directory contents
• `pwd`	**p**rint **w**orking **d**irectory
• `cd`	**c**hange **d**irectory
• `cd ..`	**c**hange **d**irectory one level up
• `mkdir test`	create directory ‘test’
• `mv test dummy`	rename ‘test’ into ‘dummy’
• `touch test-file`	create empty ‘test-file’ or update time stamp if file exists
• `mv test-file file2 dummy`	move ‘test-file’ and ‘file2’ into ‘dummy’ directory
• `cp dummy/file2 ./`	copy ‘file2’ from ‘dummy’ directory into current directory
• `cp -r dummy no-dummy`	copy ‘dummy’ directory and its contents into ‘no-dummy’ directory
• `rm -r dummy`	remove non-empty directory ‘dummy’ (**CAREFUL! NO TRASH BIN!**)

Other Useful Commands

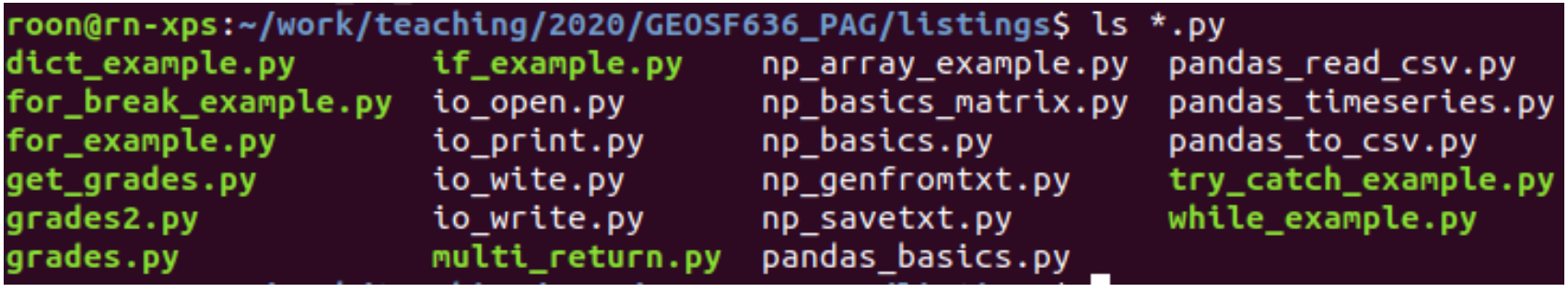
- `ls` — list directory contents
- `man cmd` — manual page for command ‘cmd’
- `less` — view a textfile (type ‘q’ to quit)
- `head -X` — display first X lines of a file
- `tail -X` — display last X lines of a file
- `wc file` — count words (-w) or lines (-l) or characters (-c) of a file
- `cat file (file2 ...)` — concatenate the contents of files and write out
- `sort` — sort contents of a file (even based on columns -k)

Wildcards

- Often you want to match multiple files at once, e.g., all files that end in `.py`

  `mv *.py listings`

- The wildcard '`*`' matches any number of characters:



- Would not match '`numpy`' as it requires a dot before '`py`'
- Wildcard recognition and expansion is called 'globbing' (stems from 'glob' (global) program which initially did this)

Wildcards

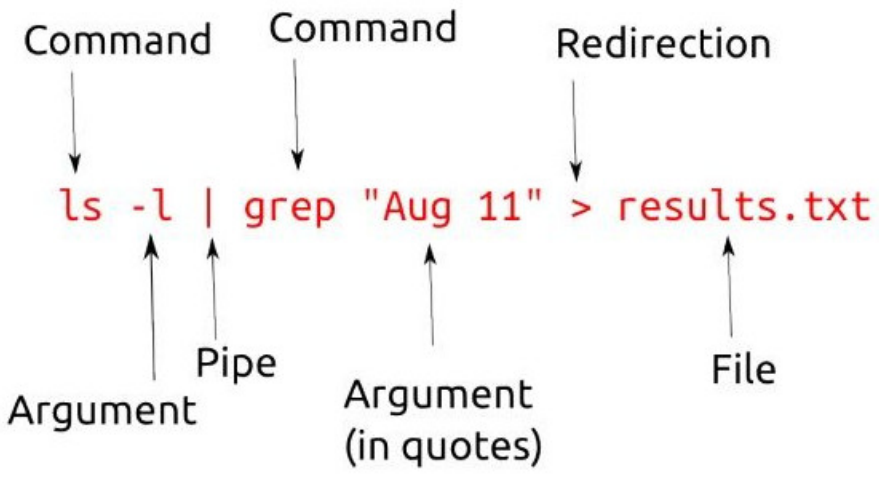
- `*` – Match zero or more characters
- `?` – Match exactly one character

```
roon@rn-xps:~/work/teaching/2020/GEOSF636_PAG/listings$ ls ???.*
bnf.txt  gmt.conf  gmt.history
roon@rn-xps:~/work/teaching/2020/GEOSF636_PAG/listings$ ls ?.log
x.log
```

- `[a-z]` – Match lower case letters
- `[A-Z]` – Match upper case letters
- `[a-zA-Z]` – Match upper and lower case letters
- `[0-9]` – Match numbers

```
roon@rn-xps:~/work/teaching/2020/GEOSF636_PAG/listings$ ls [a-g]*.py
dict_example.py  for_break_example.py  for_example.py  get_grades.py  grades2.py  grades.py
roon@rn-xps:~/work/teaching/2020/GEOSF636_PAG/listings$ ls *[0-9]*
grades2_output2.txt  grades_output2.txt              letter_grades2.bash
grades2_output.txt   grades_shell2_output_script.txt
grades2.py           instavel.gmt5
```

Input/Output Redirection



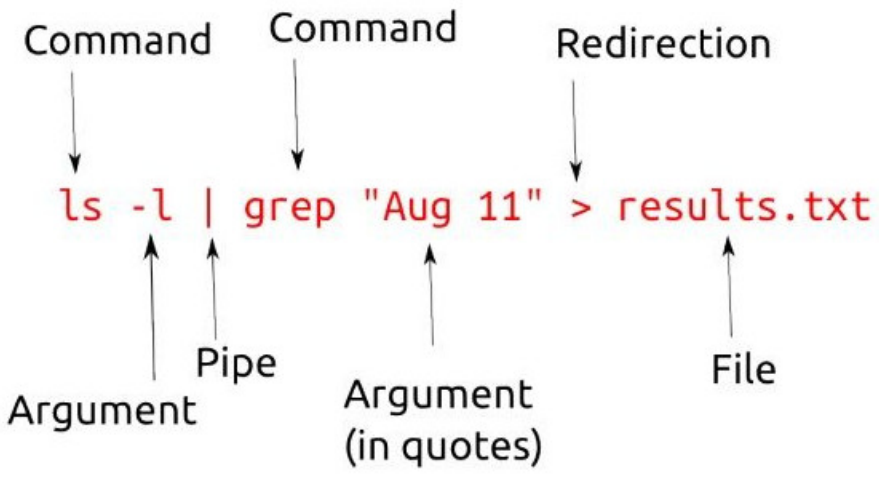
The Internet

Input/Output Redirection

- Many programs designed to take input from **standard input**, write output to **standard output**
- By default std-input is keyboard, std-output is the screen
- I/O redirection allows you to change that
- < redirects the input
- > redirects the output
- Can be used at the same time

- Send output to a file:
  `ls *.py > py.lst`
- Use file as input:
  `sort -r < py.lst`
- . . . save output to new file
  `sort -r < py.lst > pyr.lst`

Piping



The Internet

Piping

THE INCREDIBLE CONSTRUCT enabling the UNIX philosophy!

- Use **output** of one program as **input** for another: use the pipe.
- Use vertical bar `|` to realize this
- Can pipe together as many programs as you like, each needs to read from standard input and write to standard output.

- Count files of a given type:
  `ls *.py | wc -l`
- Sort listing alphabetically for 9th column
  `ls -l | sort -k 9`
- . . . scroll manually
  `ls -l | sort -k 9 | less`

Everything is Scriptable – Shell Scripting

- Do NOT type out / copy'n'paste long series of commands repeatedly
- DO bundle sets of commands in an executable script for repeated use
- Scripts don't have to be complicated, just need to reliably do something more easily than typing it out
- There's no limit to the number of scripts you can write; just remember the names
- Here's a script I use to connect to a server (I only need to type `ees.redoubt`, usually, I type `ees.r<tab><enter>`)

```
#!/bin/tcsh
ssh -2Y roon@redoubt.ees.nmt.edu
```

Everything is Scriptable – Shell Scripting

- Any series of commands you type at the prompt can be bundles in a script
- The shell provides you with variables (e.g., pre-defined `PATH`, or your own)
- Flow control allows to write programs (condition testing, loops)

Shell Scripting – Variables

- Regular variables (visible just in current shell), spaces are important!!
  `set txt = "hello world!"` (tcsh)
  `txt="hello world!"` (bash)
- Accessing variables
  `echo $txt` (tcsh, bash)
- BACKTICKS (execute command in subshell)
  `files=\`ls *.py\`` (bash, works in tcsh too)
- Environment Variables (visible in shell and scripts called from this shell)
  `setenv PATH /usr/bin` (tcsh)
  `export PATH=/usr/bin` (bash)

Shell Scripting – Quotes

- `'...'` - Single quotes force literal interpretation of what's in the quotes

tcsh:

```
$>set output = 'I want to say $txt'
$>echo $output
I want to say $txt
```

bash:

```
$>output='I want to say $txt'
$>echo $output
I want to say $txt
```

- `"..."` - Double quotes group words, but variables are still recognized and interpreted

tcsh:

```
$>set output = "I want to say $txt"
$>echo $output
I want to say hello world!
```

bash:

```
$>output="I want to say $txt"
$>echo $output
I want to say hello world!
```

Shell Scripting – Control Structures (bash only)

Conditions

Note that `[[` is actually a command/program that returns either `0` (true) or `1` (false). Any program that obeys the same logic (like all base utils, such as `grep(1)` or `ping(1)`) can be used as condition, see examples.

`[[ -z STRING ]]`	Empty string
`[[ -n STRING ]]`	Not empty string
`[[ STRING == STRING ]]`	Equal
`[[ STRING != STRING ]]`	Not Equal
`[[ NUM -eq NUM ]]`	Equal
`[[ NUM -ne NUM ]]`	Not equal
`[[ NUM -lt NUM ]]`	Less than
`[[ NUM -le NUM ]]`	Less than or equal
`[[ NUM -gt NUM ]]`	Greater than
`[[ NUM -ge NUM ]]`	Greater than or equal
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions

More conditions

`[[ -o noclobber ]]`	If OPTIONNAME is enabled
`[[ ! EXPR ]]`	Not
`[[ X && Y ]]`	And
`[[ X || Y ]]`	Or

File conditions

`[[ -e FILE ]]`	Exists
`[[ -r FILE ]]`	Readable
`[[ -h FILE ]]`	Symlink
`[[ -d FILE ]]`	Directory
`[[ -w FILE ]]`	Writable
`[[ -s FILE ]]`	Size is > 0 bytes
`[[ -f FILE ]]`	File
`[[ -x FILE ]]`	Executable
`[[ FILE1 -nt FILE2 ]]`	1 is more recent than 2
`[[ FILE1 -ot FILE2 ]]`	2 is more recent than 1
`[[ FILE1 -ef FILE2 ]]`	Same files

devhints.io/bash

Shell Scripting – Control Structures (bash only)

Conditionals

```
# String
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
else
  echo "This never happens"
fi
```

```
# Combinations
if [[ X && Y ]]; then
  ...
fi
```

```
# Equal
if [[ "$A" == "$B" ]]
```

```
# Regex
if [[ "A" =~ . ]]
```

```
if (( $a < $b )); then
   echo "$a is smaller than $b"
fi
```

```
if [[ -e "file.txt" ]]; then
  echo "file exists"
fi
```

devhints.io/bash

Shell Scripting – Control Structures (bash only)

Conditionals

```
# String
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
else
  echo "This never happens"
fi
```

```
# Combinations
if [[ X && Y ]]; then
  ...
fi
```

```
# Equal
if [[ "$A" == "$B" ]]
```

```
# Regex
if [[ "A" =~ . ]]
```

```
if (( $a < $b )); then
   echo "$a is smaller than $b"
fi
```

```
if [[ -e "file.txt" ]]; then
  echo "file exists"
fi
```

devhints.io/bash

Loops

Basic for loop

```
for i in /etc/rc.*; do
  echo $i
done
```

C-like for loop

```
for ((i = 0 ; i < 100 ; i++)); do
  echo $i
done
```

Ranges

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

With step size

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

Reading lines

```
cat file.txt | while read line; do
  echo $line
done
```

Forever

```
while true; do
  ...
done
```

devhints.io/bash

Shell Scripting – Reading Command Line Arguments

- Shell scripts can take input from the command line
- Use special variables `$1, $2, ... $n` to get first, second, ... argument
- Use special variable `$#` (bash), `$#argv` (tcsh) to get number of arguments

```
#!/usr/bin/env bash

echo "Argument 1: $1"
echo "Argument 2: $2"
echo "..."
echo "Argument 9: $9"
```

```
$> ./arguments.sh a b c d e f g h i j
Argument 1: a
Argument 2: b
...
Argument 9: i
```