OSU Oregon State UNIVERSITY

CENTER FOR GENOME RESEARCH & BIOCOMPUTING

“Introduction to Unix/Linux” INX_U18, Day 4, 2018-08-01

groups, permissions, executeables, $path, where, #!, script, interpreter

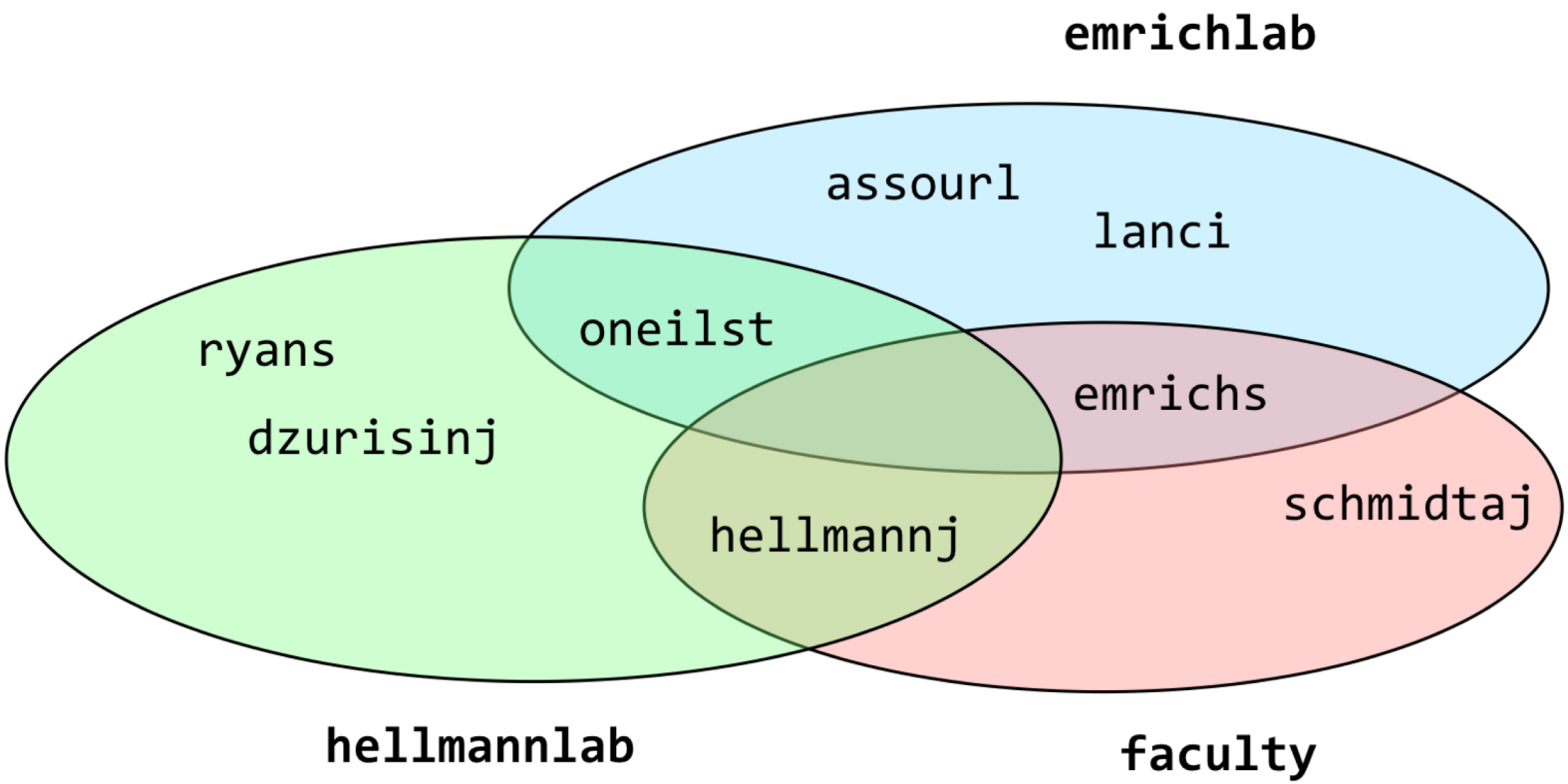
Learning Outcome(s):

Understanding directory structure/permissions

Purpose of the $path shell variable and determine a program's location.

Matthew Peterson, OSU CGRB, matthew@cgrb.oregonstate.edu

Groups



http://library.open.oregonstate.edu/computationalbiology/chapter/permissions-and-executables/

Group membership

```
groups <username>
groups $USER
```

Want to know every user in a group?

```
getent group <groupname>
```

Permissions

Each file and directory is associated with:

- One **user**
- One **group** *

Each file and directory has permissions describing:

- What the **owner** can do
- What the **group** can do
- What everyone else (**other**) can do

* (multiple groups would require ACLs, which we don't use)

Oregon State UNIVERSITY

`ls -l` List permissions

File Type
(- for file,
d for directory)

Number of links to this file
in the filesystem

```
-rwxrwxrwx 1 oneils iplant-everyone
```

user (owner)

group

user
permissions
"u"

group
permissions
"g"

others
permissions
"o"

user **group** **other**

http://library.open.oregonstate.edu/computationalbiology/chapter/permissions-and-executables/

read, write, execute for files

Code	Meaning for files
r	Can read file contents
w	Can write to (edit) the file (rename and delete the file also)
x	Can (potentially) "execute" the file

```
-rwxrwxrwx 1 oneils iplant-everyone
```

user (owner)

group

user permissions "u"

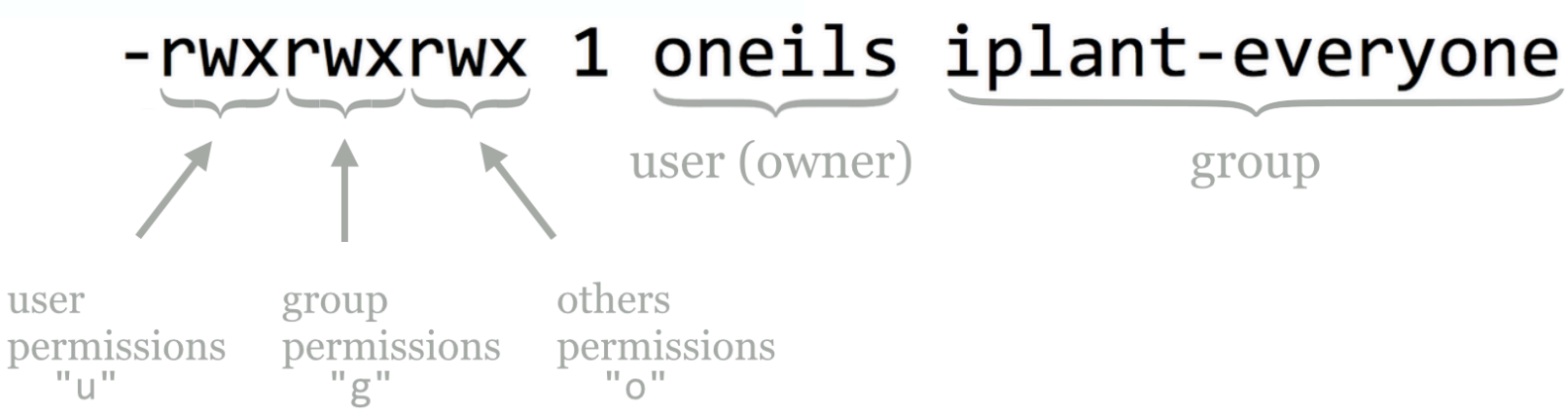
group permissions "g"

others permissions "o"

http://library.open.oregonstate.edu/computationalbiology/chapter/permissions-and-executables/

rwx for directories

Code	Meaning for directories
r	Can see contents of the directory (e.g. run `ls`)
w	Can modify contents of the directory (create or remove files/directories)
x	Can `cd` to the directory, and potentially access subdirectories



http://library.open.oregonstate.edu/computationalbiology/chapter/permissions-and-executables/

Quiz

What do these permissions represent?

```
drwxr-xr-x   projects

-rw-r--r--   todo.txt
```

u g o

chmod Add/remove permissions

"change mode"

```
chmod <ugo><+->><rwx> <target>
```

user, group, *and/or* others

add *or* remove

read, write, *and/or* execute

chmod File examples

`chmod go-w p450s.fasta`	Remove write for group and others
`chmod ugo+r p450s.fasta`	Add read for user, group, and others
`chmod go-rwx p450s.fasta`	Remove read, write, and execute for group and others
`chmod ugo+x p450s.fasta`	Add execute for user, group, and others
`chmod +x p450s.fasta`	Same as `chmod ugo+x p450s.fasta`

The last example, **+x** with no "**ugo**" is a shortcut; leaving off the "**ugo**" implies it is for "*everyone,*" i.e., **u**, **g**, and **o**

http://library.open.oregonstate.edu/computationalbiology/chapter/permissions-and-executables/

`chmod` Applied to directories

To modify permissions to a directory and everything inside, add the **`-R`** flag (recursive).

```
chmod -R ugo+r projects
```

chmod Sharing a directory

Note: If you want to share a sub-directory in your $HOME directory, it will not work if your $HOME directory is "locked up" tight, e.g.,

```
chmod -R ugo+r $HOME/projects
```

and

```
chmod go-rwx $HOME
```

Oregon State UNIVERSITY

chgrp Change group

You can change the group of a file or directory <u>only</u> if you are already a member of that group.

```
chgrp <group> <fileordir>
```

```
oneils@atmosphere ~/apcb/intro$ groups $USER
oneils : iplant-everyone users community de-preview-access atmo-user dnasubway-users myplant-users
oneils@atmosphere ~/apcb/intro$ chgrp community p450s.fasta
```


http://library.open.oregonstate.edu/computationalbiology/chapter/permissions-and-executables/

executeable programs

- In Unix a "program" is a file that has the executable permission set.
- **x** is often set on binary programs (0's and 1's)
- Example of default executable programs:

```
cd /bin
ls -l
less echo
```

Note: less will generate a warning

Oregon State UNIVERSITY

Rules for running programs

- To run a program (e**x**ecutable file) we give the shell the absolute or relative path to it.

```
/bin/ls
/bin/echo hello
```

Q: We can run `ls` and `echo` without specifying an absolute or relative path, how?

A: The shell look for the programs in its **`$path`**

```
echo $path
```

$path

- **$path** is a shell variable* that contains a list of absolute and relative paths to search first when trying to run (e**x**ecute) a program.
- The shell searches the paths <u>one at a time</u> and e**x**ecutes <u>the first match</u> it finds (even if multiple matches exist!)

* Shell variables are similar to environment variables but only the shell (and not other programs can see them)

where is my command coming from?

where `<command>`

where `ls`

- It tells us **where** the program is and if there is an alias for it in our shell (tcsh)
- Where does **where** live? (and is it a program?)

where `where`

- Programs are *usually* e**x**ecutable files located by the shell in the **$path**, but not everything is a program, some are "**built-in**" to the shell itself, e.g., `cd`, `history`, `setenv`, `exit`

Oregon State UNIVERSITY

#! Making files executable

```
nano -w myprog.sh
#!/bin/tcsh

echo 'this is a test'
echo 'all done'
```

#! = 'shebang'

(also shabang)
"SHArp BANG"
"hasSH BANG"

```
  GNU nano 2.0.9            File: myprog.sh

#!/bin/tcsh

echo 'this is a test'
echo 'all done'

^G Get Help  ^O WriteOut  ^R Read File^Y Prev Page^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

Making a file's permission executable

```
chmod +x myprog.sh
```

This is an executable (text) file, but is it a program?

```
$HOME/myprog.sh
./myprog.sh
```

How about running just: `myprog.sh`
Why does that work (with no absolute or relative path specified)?

Oregon State UNIVERSITY

Programs (scripts) in other languages

```
nano -w myprog.py
#!/usr/bin/python

print("hi")
print(2+abs(-6))
chmod +x myprog.py
```

```
  GNU nano 2.0.9            File: myprog.py

#!/usr/bin/python

print("hi")
print(2+abs(-6))

                          [ Wrote 5 lines ]
^G Get Help  ^O WriteOut  ^R Read File^Y Prev Page^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

These programs are: **interpreted** (as text) and not **compiled** into binary, i.e., 0's and 1's

Q: Do the extensions **.sh** and **.py** matter?

A: No.

Command / Concept Review

- groups
- finger
- getent group
- chmod / chgrp
- where

user

group

other

rwx

$path

#!