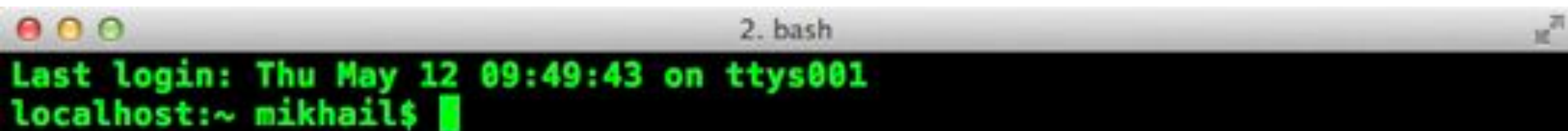


KNOW YOUR UNIX

Know your Unix

- Unix is a family of operating systems and environments that exploits the power of linguistic abstractions to perform tasks
- Unix users spend a lot of time at the **command line**
- In Unix, a word is worth a thousand mouse clicks

A screenshot of a terminal window. The title bar shows three window control buttons (red, yellow, green) on the left and the text "2. bash" in the center. The terminal content is displayed in green text on a black background. It shows the login message "Last login: Thu May 12 09:49:43 on ttys001" and the prompt "localhost:~ mikhail\$" followed by a green cursor block.

```
2. bash
Last login: Thu May 12 09:49:43 on ttys001
localhost:~ mikhail$ █
```

Getting to the command line

- Remote access, SSH, PuTTY.
- Mac OS X + Xcode development suite (free) + X11 server (free) + iTerm2 (optional)
- Ubuntu Linux (long-term support LTS version, XX.04)
- Windows users
 - Cygwin, <http://www.cygwin.com/>
 - Git Bash, <https://git-for-windows.github.io/>
 - Boot from a CD or USB
 - Install the whole Linux systems as a Virtual Machine in VirtualBox

<http://www.chiark.greenend.org.uk/~sgtatham/putty/>
<https://developer.apple.com/xcode/>
<https://www.xquartz.org/>
<https://iterm2.com/>
<http://www.ubuntu.com/download/desktop>
<https://www.virtualbox.org/>

Obtaining new software

Modern Unixes have package managers to that download install (free) software for you

- On a Mac, **MacPorts** is a popular package-management system, and **Homebrew** is gaining in popularity
- On Ubuntu, **apt** is the standard package manager, with both a command-line and graphical interface available
- On Windows, **Cygwin** installs everything precompiled through its setup file. Do not delete **setup-x86_64.exe** file after installing Cygwin, explore what Linux tools are available (a lot)

<https://www.macports.org/>

<http://brew.sh/>

https://en.wikipedia.org/wiki/Advanced_Packaging_Tool

<https://cygwin.com/install.html>

What is shell

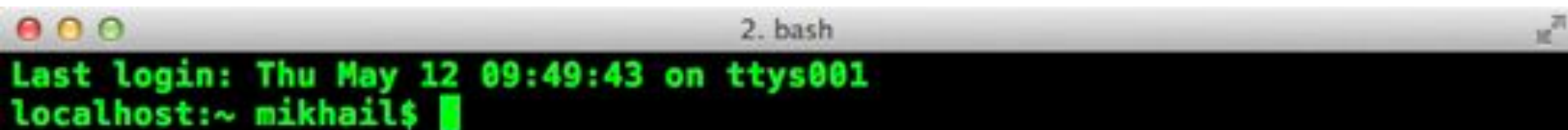
- Shell is an interactive environment with a set of commands to initiate and direct computations
- Shell encloses the complexity of OS, hence the name
 - You type in commands
 - Shell executes them

Most popular types of shell

- **bash** Bourne-Again shell
- **tcsh** TENEX C shell
- **zsh** Z shell
- Change shell - **chsh -s /bin/zsh**

Exercise:

- Check which shell you are using – **echo \$SHELL**



A terminal window titled "2. bash" with a dark background and green text. The window shows the following text: "Last login: Thu May 12 09:49:43 on ttys001" and "localhost:~ mikhail\$".

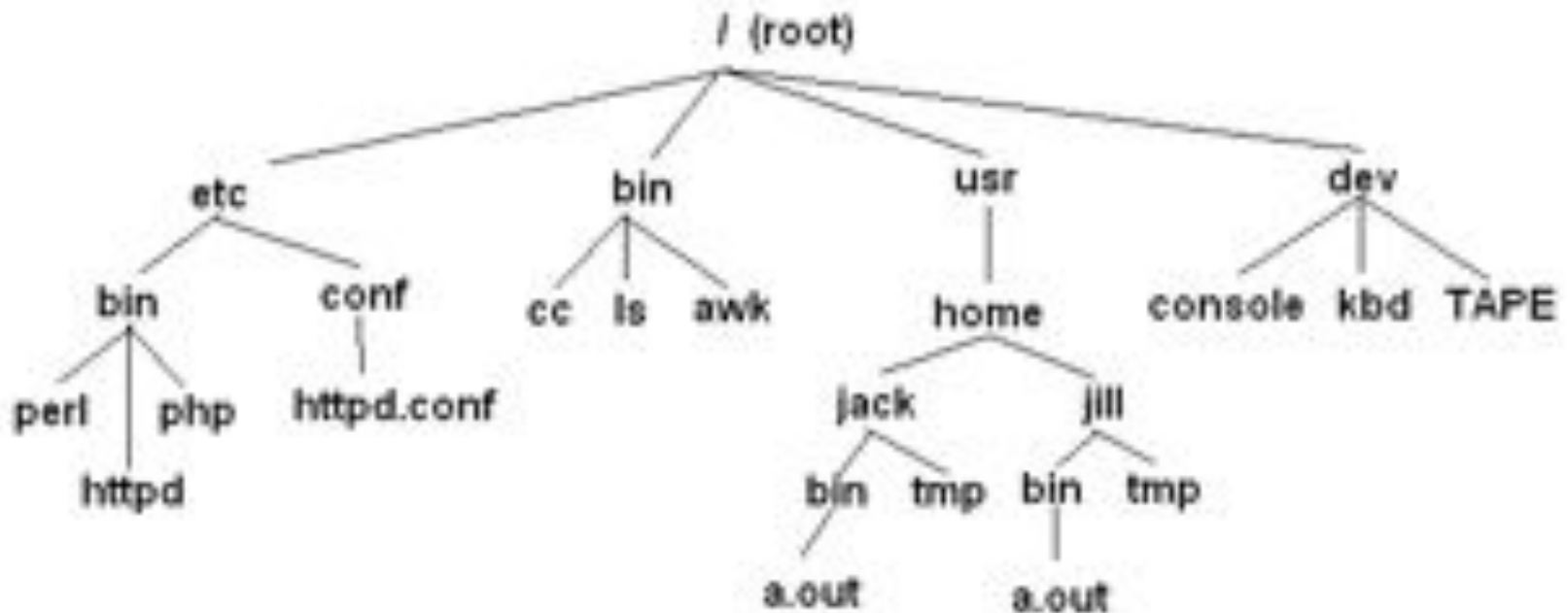
```
2. bash
Last login: Thu May 12 09:49:43 on ttys001
localhost:~ mikhail$
```

Interacting with shell

- Most commands take additional arguments that fine tune their behavior
- If you don't know what a command does, use the command `man <command>`.
- Some tools use an alternate documentation system called `info`
- Press `q` to quit the man page viewer
- Most often, you'll use `<command> -h` or `<command> --help`



File system: Full vs. relative paths



`cd /`

`cd /usr/home/jack/bin`

`cd ..`

`cd, or cd ~`

`cd --`

go to the root directory

go to the user's sub-directory

go to the upper level directory

go to the user's home directory

go to the last visited directory



Orienting yourself

The filesystem

- **ls** list files
- **cd** change directory
- **pwd** print working directory

ls list all files

ls -l list files in `one` column

ls -lah list files in long format, include special directories, sizes in human-readable format

ls -A list all entries in the directory



Wildcards and patterns

- ***** matches any character
- **?** matches a single character
- **[chars]** matches any character in chars
- **[a-b]** matches any character between a and b

```
ls *.md
```

```
ls [Rt]*
```



Looking inside files

- **cat <file>** prints out content of a file. If multiple files, consequently prints out all of them (concatenates)
- **zcat** prints out content of gzipped files
- **more/less <file>** shows the content of the file one screen at a time

Keyboard shortcuts

- **space** forward
- **b** backward
- **g** go to the beginning
- **G** go to the end
- **/<text>** starts forward search, **enter** to find next instance
- **q** quit



Creating, moving, copying, and removing files

- `touch <file>` creates an empty file
- `mkdir <dirname>` creates a directory
- `cp <source_file> <target_file>` copy a file to another location/file
- `mv <source_file> <target_file>` move a file
- `rm <file>` remove a file. If multiple files provided, removes all of them
- `rm -r <dirname>` recursive removal (deletes a directory)



Finding your files

find lists all files under the working directory (and its subdirectories) based on arbitrary criteria

find . prints the name of every file or directory, recursively. Starts from the current directory

find . -type f finds files only

find . -type d -maxdepth 1 finds directories only, at most 1 level down

find . -type f -name "*.mp3" finds only *.mp3 files

find . -type f -name "README.md" -exec wc -l {} \;
find files and execute a command on them



Permissions: chmod, chown and chgrp

In Unix, every file and directory has an owner and a group

- **Owner** is the one who created a file/directory
- **Group** defines rules of file operations and/or permissions
- **Every** user on a Unix machine can belong to one or more groups

Every file has **three permission levels**

- what the **user** can do
- what the **group** can do
- what the **all** can do

To see the owner, group and permissions associated with a file run `ls -lah`

```
localhost:BIOS692 mikhaill$ ls -lah
total 88
drwxr-xr-x 18 mikhaill staff 6128 May  2 15:37 .
drwxr-xr-x 18 mikhaill staff 6128 May  7 19:47 ..
-rw-r--r--@ 1 mikhaill staff 6.0K May  2 15:37 .DS_Store
drwxr-xr-x 13 mikhaill staff 4428 May  4 19:32 .git
-rw-r--r--  1 mikhaill staff  118 Apr 26 15:39 .gitignore
-rw-r--r--  1 mikhaill staff   198 Apr 26 12:52 Gemfile
-rw-r--r--  1 mikhaill staff 4058 Apr 26 14:41 License.md
-rw-r--r--  1 mikhaill staff  11K Apr 26 12:52 Rakefile
-rw-r--r--  1 mikhaill staff 4728 Apr 26 16:17 README.md
-rw-r--r--  1 mikhaill staff 3.5K Apr 26 15:11 _config.yml
-r-xr-xr-x  4 mikhaill staff 1368 Apr 26 12:52 _includes
-r-xr-xr-x  5 mikhaill staff 1708 Apr 26 12:52 _layouts
-r-xr-xr-x  3 mikhaill staff 1028 Apr 26 12:52 _plugins
-r-xr-xr-x  7 mikhaill staff 2388 Apr 26 16:18 _site
```



Permissions: chmod, chown and chgrp

The first column tells you about the permissions on the file

- The very first character in the permissions column tells you what kind of file it is. A `-` means it's a regular file. A `d` means it's a directory
- The next **nine** characters come in **three** classes, each has three characters. The three classes are owner/group/world permissions
- Inside a permission class, `r` means that class can read the file; `w` means that class can write the file; `x` means that class can execute the file
- If a file is a directory, `x` grants the permission to access inside the directory, while `r` grants permission to list its contents

The third and fourth tell you the owner and group respectively

```
localhost:BIOS692 mikhaill$ ls -lah
total 88
drwxr-xr-x  18 mikhaill  staff   6128 May  2 15:37 .
drwxr-xr-x  18 mikhaill  staff   6128 May  7 19:47 ..
-rw-r--r--@  1 mikhaill  staff   6.0K May  2 15:37 .DS_Store
drwxr-xr-x  13 mikhaill  staff   4428 May  4 19:32 .git
-rw-r--r--   1 mikhaill  staff    118 Apr 26 15:39 .gitignore
-rw-r--r--   1 mikhaill  staff    198 Apr 26 12:52 Gemfile
-rw-r--r--   1 mikhaill  staff   4058 Apr 26 14:41 License.md
-rw-r--r--   1 mikhaill  staff   11K Apr 26 12:52 Rakefile
-rw-r--r--   1 mikhaill  staff   4728 Apr 26 16:17 README.md
-rw-r--r--   1 mikhaill  staff   3.5K Apr 26 15:11 _config.yml
-rw-r-x   4 mikhaill  staff   1368 Apr 26 12:52 _includes
-rw-r-x   5 mikhaill  staff   1708 Apr 26 12:52 _layouts
-rw-r-x   3 mikhaill  staff   1028 Apr 26 12:52 _plugins
-rw-r-x   7 mikhaill  staff   2388 Apr 26 16:18 _site
```



Chaining commands: pipes

One of the most useful capabilities of Unix is the ability to redirect the STDOUT of one command into the STDIN of another

The “|” (pipe) character feeds output from the first program (to the left of the “|”) as input to the second program on the right.

Therefore, you can string all sorts of commands together using the pipe

- `find . | wc -l`
- `cat names.txt | sort | uniq -c`

Executing one command AFTER another completed successfully

- `<command> && <command>`
`mkdir music && mv *.mp3 music/`



Chaining commands: redirections

Nearly every command in Unix makes use of a convention to have a "standard input" (also called `stdin` or `STDIN`, or `channel 0`) and "standard output" (also called `stdout` or `STDOUT`, or `channel 1`)

There is also a "standard error" (`stderr` or `STDERR`, or `channel 2`) output that is, by convention, reserved for error messages

If you want to dump the standard output into a file, use command `> file` (overwrites the file). `>> file` (appends to the file)

- `find / 2> error.log` capture `STDERR` into a file
- `find / 2> /dev/null` suppress `STDERR` messages
- `find / 2>&1` add `STDERR` to `STDOUT`

Redirection works in another direction

- `grep CC0 < License.md`

Or, the content of a command into another command

- `join <(sort file1) <(sort file2)`



Other essential commands

- `head/tail`
- `for`
- `sort`
- `uniq`
- `wc`
- `tr`
- `grep`
- `join`
- `cut`
- `comm`
- `echo`
- `basename`
- `dirname`
- `history`
- `which`
- `who`

Shell conveniences

- Tab completion
- **Ctrl-c** cancel the command you are writing
- **Ctrl-r** reverse search through your command
line history
- **history** shows your previous commands
- **!**<history number>**** repeats specific command
- **!!** repeats the last command



SHELL SCRIPTING

Workflow scripts

- A script is a **file** with a list of shell commands executed by an interpreter
- Shebang (**#!**) defines the interpreter on the first line
 - **#!/bin/bash** commands interpreted by bash
 - **#!/usr/bin/python** interpreted by Python

Exercise: Create file `hello_world.sh`

```
#!/bin/bash  
echo Hello World
```

- Should have **x** permissions, `chmod u+x hello_world.sh`
- Running a script
`./hello_world.sh`



Variables

- Set a variable
 - `count_of_files=3`
(wrong: `count_of_files = 3`)
 - `file="/home/mdozmorov/work/README.md"`
 - `count_of_files=$file`
- Use a variable
 - `echo $file`



Capturing output with `backticks`

Often, one wants to capture the output of a command as a variable

Wrap a command into “`” backticks - the backwards apostrophes that appear on a US English keyboard at the upper left, under the ~ (tilde)

```
echo `date`  
CURRENT_DIR=`pwd`  
file_name=`basename /bin/mkdir`
```



Arguments of a script as variables

```
> ./hello_world.sh "Hello World!"
```

- `echo $0` prints the script name
- `echo $1` prints the first argument
- `echo $2` prints the second argument
- `echo ${10}` prints the tenth argument
- `echo $#` prints the number of arguments



Internal variables

- Set system's parameters. Can be defined in system's configuration files `.bashrc`, `.bash_profile`
 - **DISPLAY** tells X11 on which display to open windows
 - **EDITOR** default text editor; usually **emacs** or **vim**
 - **HOME** path to user's home directory; same as `~`
 - **PATH** path to executable programs
 - **PWD** current directory, same as `pwd`
 - **SHELL** path to the current shell
 - **TERM** current terminal type
 - **USER** account name of current user, same as `whoami`

Exercise:

- Check the `$PATH` (or, pick any) variable (hint: use `echo`)



Aliases

To avoid retyping commands - use an alias. Can be defined in system's configuration files `.profile` (Linux), `.bash_profile` (Mac), `.bashrc`

```
alias lah='ls -lah'
```

```
alias ..='cd ..'
```

```
# get top process eating memory
```

```
alias psmem='ps auxf | sort -nr -k 4'
```

```
alias psmem10='ps auxf | sort -nr -k 4 | head -10'
```

```
# get top process eating cpu
```

```
alias pscpu='ps auxf | sort -nr -k 3'
```

```
alias pscpu10='ps auxf | sort -nr -k 3 | head -10'
```

```
# Find files eating space in the current directory
```

```
alias spacehogs='du -cks * | sort -rn'
```



Conditional execution

- if-then-else

```
if [ ! -e $results_dir ]; then
    mkdir $results_dir;
fi
```

- Some popular operators for checking a condition include:
 - **-e <file>** TRUE if a specific file/directory exists
 - **-s <file>** TRUE if non-empty file
 - **-z <string>** TRUE if the given string is empty
 - **<string1> = <string2>** TRUE if the two strings are equal

help test see all operators

Loops

- for-do-done

```
for file in `ls *.txt`; do
    echo $file;
    gzip $file;
done
```

- while-do-done