Introduction of Linux & Shell Programming

饶卫雄

wxrao@tongji.edu.cn

2023-09

内容

- Linux Introduction
- Connecting
- Linux Interaction
- I/O Redirection
- The Filesystem
- Processes & Job Control
- Editors
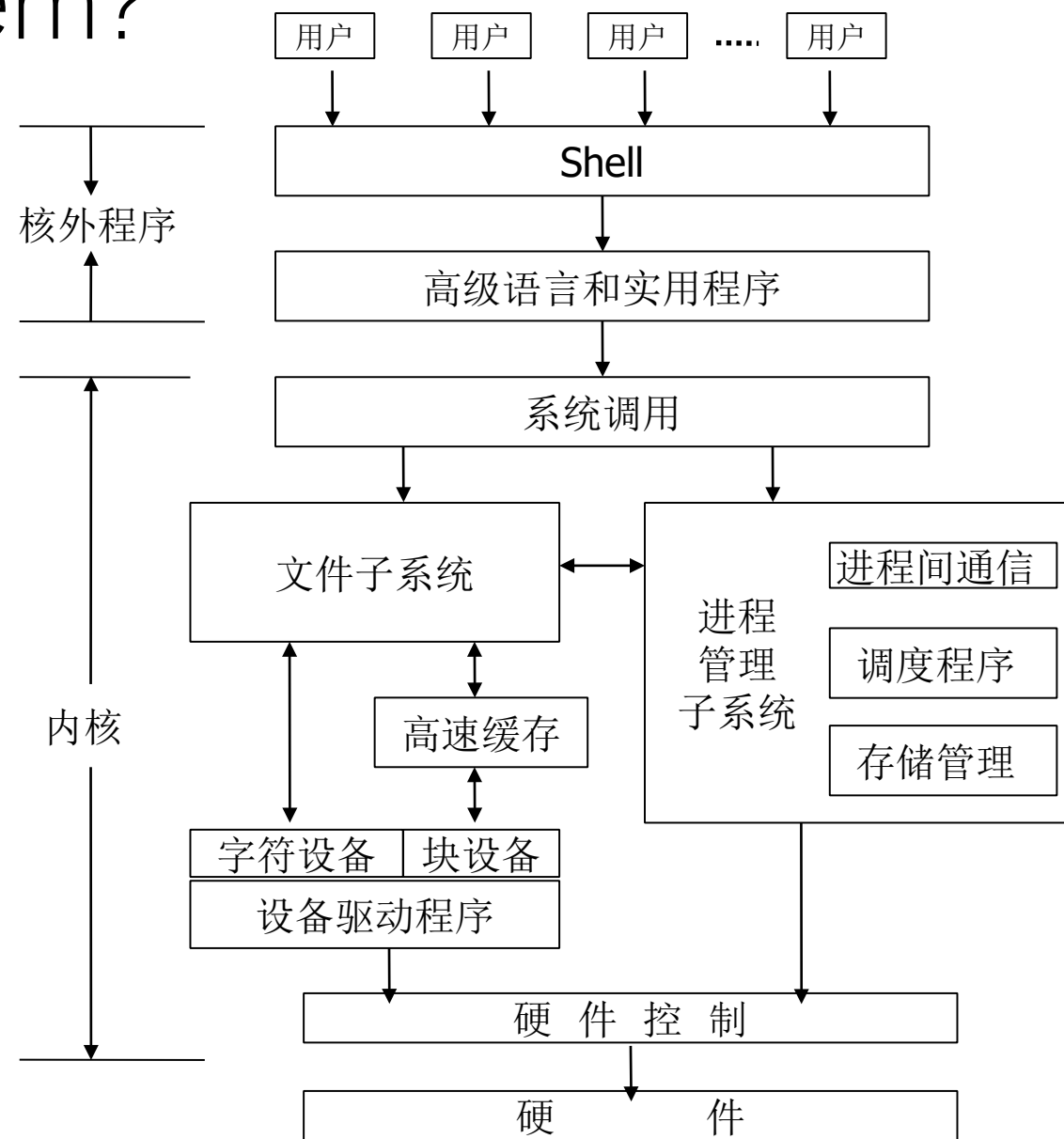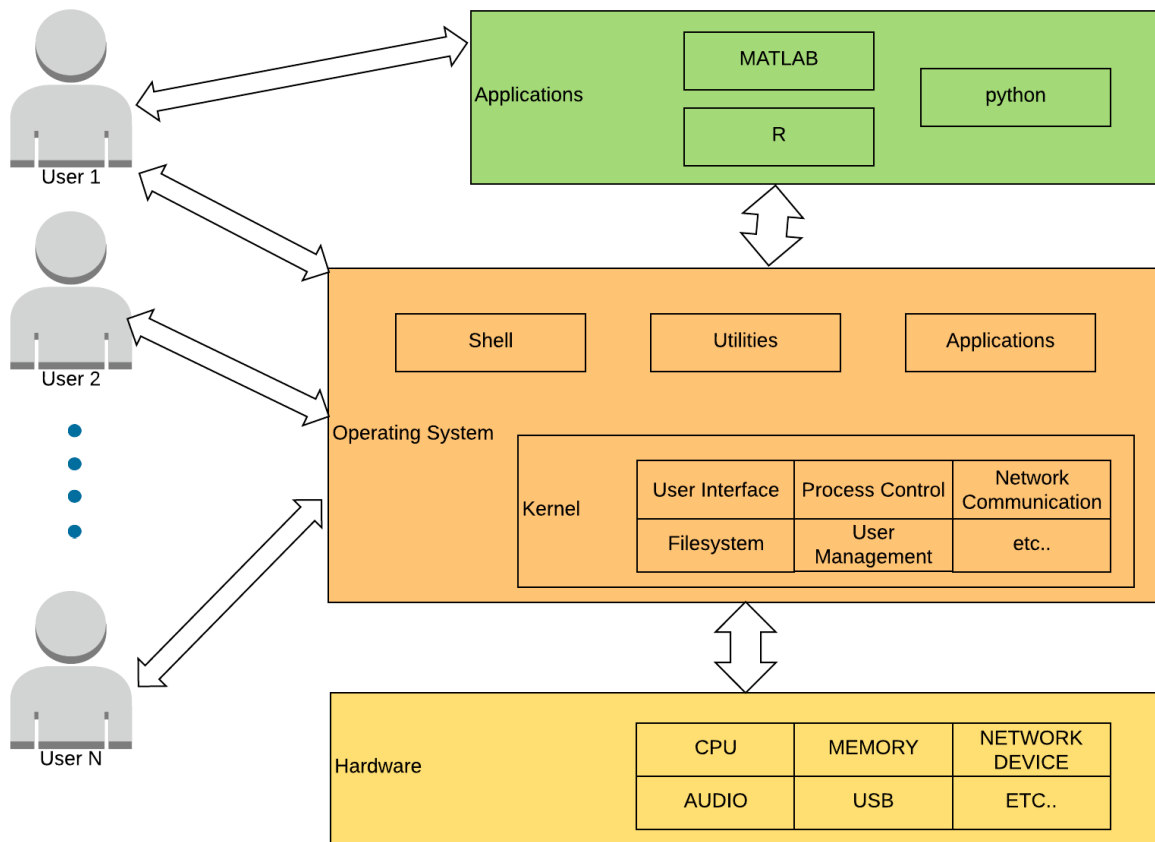- Creating and Running Code

1. Linux Introduction

Linux What, Who, When, Where & Why

What is Linux

- Unix-like computer **operating system** assembled under the model of free and open-source software development and distribution.

- These operating systems share the **Linux kernel**.
  - Typically have the GNU utilities

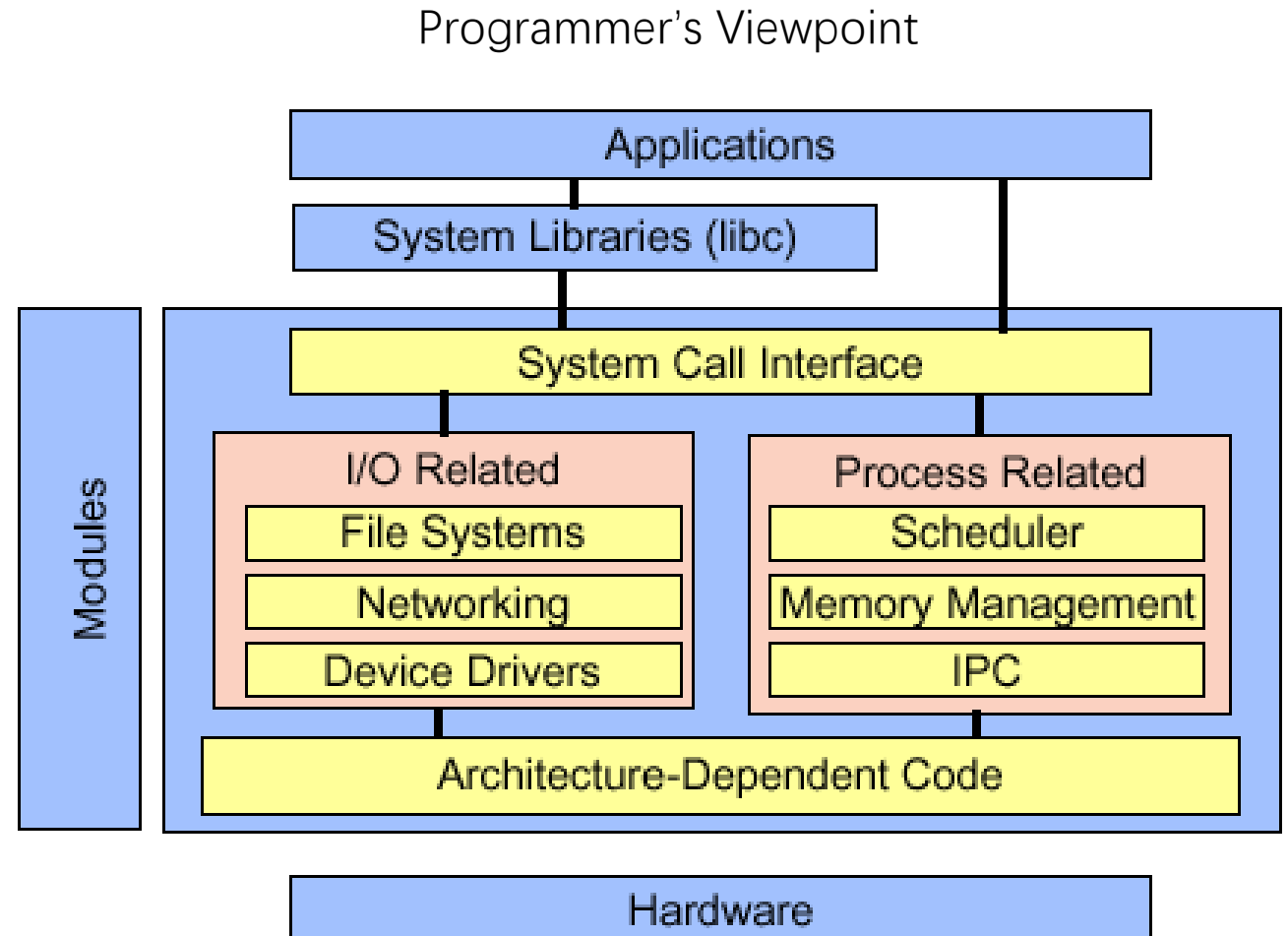- Comes in several "distributions" to serve different purposes.

What is an operating system?

Applications
MATLAB
python
R

Operating System
Shell | Utilities | Applications

Kernel
| User Interface | Process Control | Network Communication |
| Filesystem | User Management | etc.. |

Hardware
| CPU | MEMORY | NETWORK DEVICE |
| AUDIO | USB | ETC.. |

User 1
User 2
User N

用户 用户 用户 ..... 用户

核外程序

Shell

高级语言和实用程序

内核

系统调用

文件子系统

进程管理子系统

进程间通信

调度程序

存储管理

高速缓存

字符设备 | 块设备

设备驱动程序

硬 件 控 制

硬　　件

What is Linux

- Bird's eye view

Programmer's Viewpoint
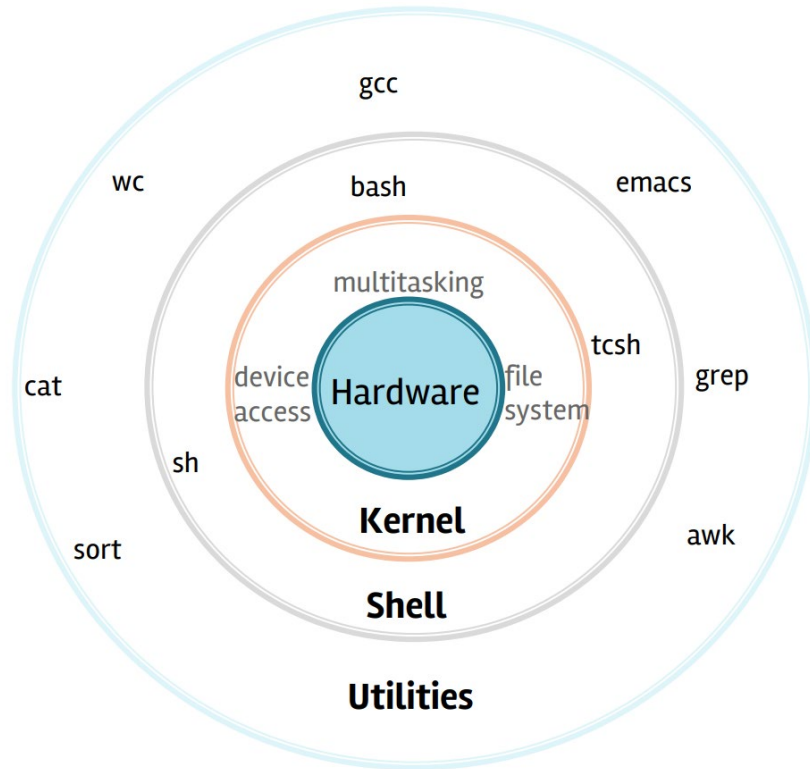
Who is Linux

&

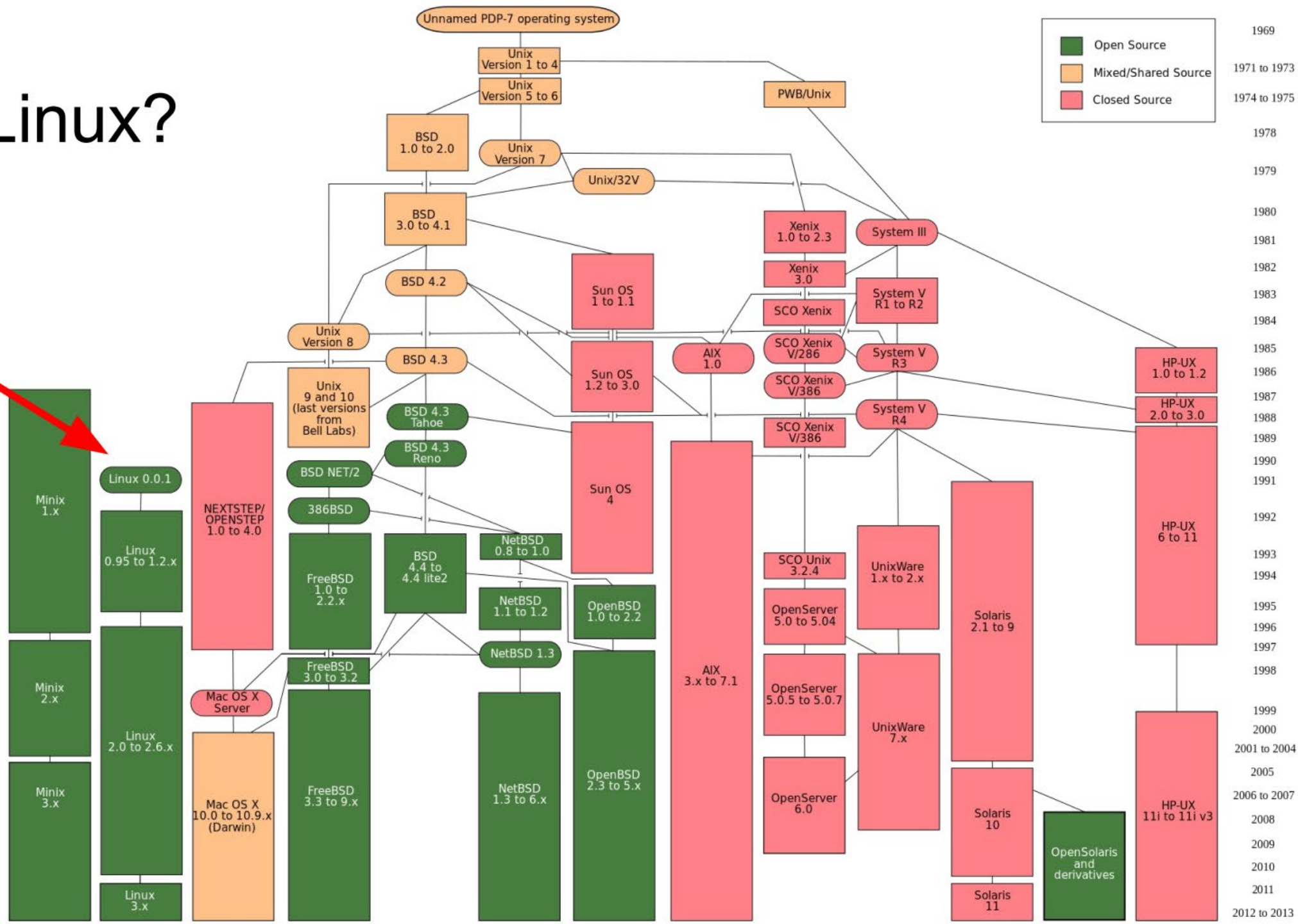

Linux is an **O/S core** originally written by Linus Torvalds. Now almost 10,000 developers including major technology companies like Intel and IBM.

A set of programs written by Richard Stallman and others. They are the GNU utilities.

When is Linux?

~1991

Legend:
- Open Source (green)
- Mixed/Shared Source (orange)
- Closed Source (pink)

Timeline (right margin): 1969, 1971 to 1973, 1974 to 1975, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001 to 2004, 2005, 2006 to 2007, 2008, 2009, 2010, 2011, 2012 to 2013

Nodes: Unnamed PDP-7 operating system, Unix Version 1 to 4, Unix Version 5 to 6, PWB/Unix, BSD 1.0 to 2.0, Unix Version 7, Unix/32V, BSD 3.0 to 4.1, Xenix 1.0 to 2.3, System III, BSD 4.2, Sun OS 1 to 1.1, Xenix 3.0, System V R1 to R2, SCO Xenix, Unix Version 8, BSD 4.3, Sun OS 1.2 to 3.0, AIX 1.0, SCO Xenix V/286, System V R3, Unix 9 and 10 (last versions from Bell Labs), SCO Xenix V/386, HP-UX 1.0 to 1.2, BSD 4.3 Tahoe, System V R4, HP-UX 2.0 to 3.0, Minix 1.x, Linux 0.0.1, NEXTSTEP/OPENSTEP 1.0 to 4.0, BSD 4.3 Reno, BSD NET/2, Sun OS 4, SCO Xenix V/386, Linux 0.95 to 1.2.x, 386BSD, NetBSD 0.8 to 1.0, Mac OS X Server, FreeBSD 1.0 to 2.2.x, BSD 4.4 to 4.4 lite2, SCO Unix 3.2.4, UnixWare 1.x to 2.x, HP-UX 6 to 11, Minix 2.x, NetBSD 1.1 to 1.2, OpenBSD 1.0 to 2.2, OpenServer 5.0 to 5.04, Solaris 2.1 to 9, Linux 2.0 to 2.6.x, Mac OS X 10.0 to 10.9.x (Darwin), FreeBSD 3.0 to 3.2, NetBSD 1.3, AIX 3.x to 7.1, OpenServer 5.0.5 to 5.0.7, UnixWare 7.x, Minix 3.x, FreeBSD 3.3 to 9.x, NetBSD 1.3 to 6.x, OpenBSD 2.3 to 5.x, OpenServer 6.0, Solaris 10, OpenSolaris and derivatives, HP-UX 11i to 11i v3, Solaris 11, Linux 3.x

A Short History of UNIX

- Multics, AT&T Bell Lab, GE, MIT
- 1969, UNIX, Ken Thompson, Dennis Ritchie
- 1973, Rewrite UNIX with C
- Berkeley UNIX(BSD UNIX)
- 1983, System V
- Commercial products
  - SunOS, Solaris, HP-UX, AIX, SCO UNIX
- Standards
  - SVID, IEEE POSIX, X/Open XPG4.2

A Short History of Linux(1)

- 1984: Richard Stallman starts GNU project
  - GNU's Not Unix
  - http://www.gnu.org

- Purpose: Free UNIX
  - "Free as in Free Speech, not Free Beer"

- First step: re-implementation of UNIX Utilities
  - C compiler, C library
  - emacs
  - bash

- To fund the GNU project, the Free Software Foundation is founded
  - http://www.fsf.org

A Short History of Linux(2)

- 1991: Linus Torvalds writes 1st version of Linux kernel
  - Initially a research project about the 386 protected mode
  - Linus' UNIX -> Linux
  - Combined with the GNU and other tools forms a complete UNIX system
- 1992: First distributions emerge
  - Linux kernel
  - GNU and other tools
  - Installation procedure
- The rest is history...

Where is Linux

- **World Wide Web**
  - 67% of the world's web-servers run Linux (2016)
- **Research/High-Performance Compute**
  - Google, Amazon, NSA, 100% of TOP500 Super-computers.
- **Modern Smartphones and devices**
  - The Android phone
  - Amazon Kindle
  - Smart TVs/Devices

- Desktop computers and laptops
- Servers
- High performance clusters
- Embeded systems
- Home entertainment (smart TVs, IoT devices)
- Cellphones (Android)

Why Linux

- Free and open-source.
- Powerful for research datacenters
- Personal for desktops and phones
- Universal
- Community (and business) driven.

The most common OS used by BU researchers when working on a server or computer cluster

内容

- Linux Introduction
- Connecting
- Linux Interaction
- I/O Redirection
- The Filesystem
- Processes & Job Control
- Editors
- Creating and Running Code

2. Connecting

Let's use Linux

Connection Protocols & Software

Local System

Remote Server

- Connection Protocols and Software
  - Remote Connections: Secure SHell (SSH)
  - Remote Graphics: X-Windowing (X, X-Win)
  - Data Transfer: Secure File Transfer Protocol (SFTP)
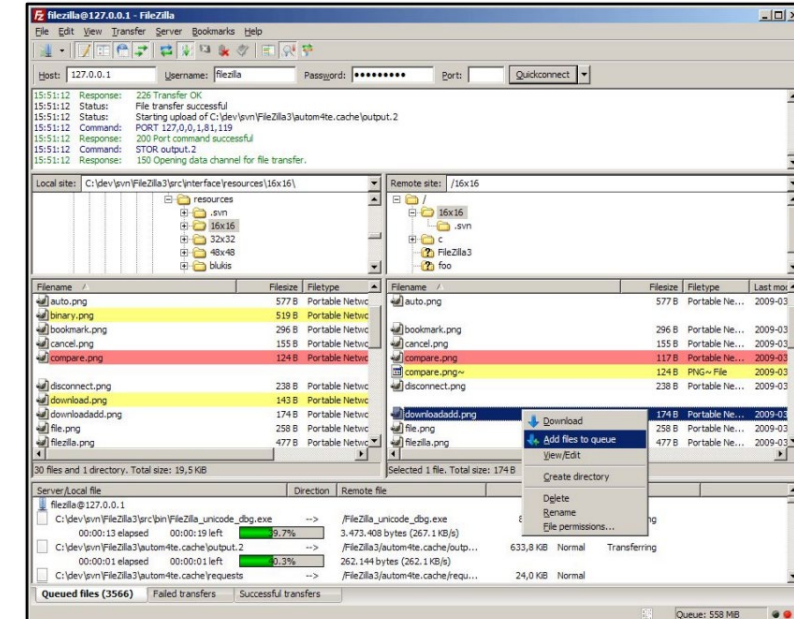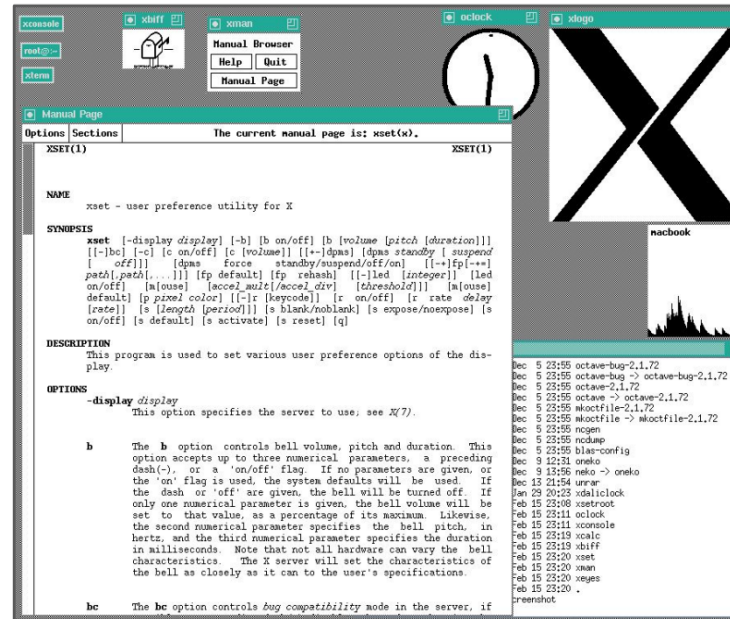
Connecting from Different Platforms

	SSH	X-Win	SFTP
Microsoft Windows		MobaXterm https://mobaxterm.mobatek.net	
Apple macOS	Terminal (Built in)	XQuartz https://www.xquartz.org	Cyberduck https://cyberduck.io
Linux	Terminal (Built in)	X11 (Built in)	Various (Built in)

- **Microsoft Windows**: You need software that emulates an "X" terminal and that connects using the "SSH" Secure Shell protocol.
  - SSH/X-Windows: X-Win32: https://www.bu.edu/tech/services/support/desktop/distribution/xwindows/
  - MobaXterm: http://mobaxterm.mobatek.net/
  - SFTP: Filezilla https://filezilla-project.org/

- **Apple macOS**
  - SSH: Terminal ○ Built in to macOS Applications > Utilities > Terminal
  - X-Windows: XQuartz ○ Download: https://www.xquartz.org/ ○ Note: This install requires a logout.
  - SFTP: Your choice ○ Filezilla: https://filezilla-project.org/ (Cross-platform, open-source) ○ Cyberduck: https://cyberduck.io (macOS native, drag-and-drop)

- **Linux**
  - SSH: Terminal ○ Built in to Linux Applications > System > Terminal
  - X-Windows: X11 ○ Built in to Linux ○ Use your package manager.
  - SFTP: Your choice ○ Usually has one Built in. ○ Alternate: Filezilla (https://filezilla-project.org/)

Using **ssh**

- **Secure Shell**

- ssh [opts] <username>@remote.host
  - *username* is the username on the *remote* host.
  - *remote.host* is the url of the server you want to log into.
  - Use -l to specify username (no need for @ anymore).
  - -p <port>: connect to a specific port (may be necessary depending on the server).
  - Can forward graphical *programs* (NOT the entire screen):
    - Enable X11 forwarding with -X.
    - Enable "trusted" X11 forwarding with -Y (actually less secure, only use if needed).

ssh by Example

- On csug (CS Undergraduate) I am sjm324:
  - ssh sjm324@128.253.141.42
  - ssh -l sjm324 128.253.141.42
- Sweet! Hey csug has Matlab, can I use it?
  >>> /usr/local/MATLAB/R2012a/bin/matlab
  Warning: No display specified. You will not be able to
  display graphics on the screen.
  exit()
  # exit() left Matlab
  >>> exit # close the ssh connection
- ssh -X sjm324@128.253.141.42
  >>> /usr/local/MATLAB/R2012a/bin/matlab

Transferring Files

Secure Copy

scp [flags] <from> <to>

    - It's exactly like cp, only you are transferring over the web.

    - Transfer *from* the client *to* the remote host.

    - Transfer *from* the remote host *to* the client.

    - Copy directories just like before using the -r flag.

    - Must specify user on the remote.

    - Remote syntax:

        user@host.name:/path/to/file/or/folder

    - You need the : to start the path.

    - If you don't have permission...you can't get it!

    - More modern systems let you TAB complete across the remote directories :)

scp by Example

- Transfer from remote to local computer:

>>> scp sjm324@blargh.ru:/absolute/path/colorize.sh ~/Desktop/
colorize.sh 100% 3299 3.2KB/s 00:00

- Transfer from remote to local:

>>> scp sjm324@blargh.ru:~/Desktop/colorize.sh /usr/share/
colorize.sh 100% 3299 3.2KB/s 00:00

- Transfer from the client to the remote: just reverse it.

>>> scp /usr/share/colorize.sh sjm324@blargh.ru:~/Desktop/
colorize.sh 100% 3299 3.2KB/s 00:00

- As with regular cp, can give a new name at the same time:

>>> scp /usr/share/colorize.sh sjm324@blargh.ru:~/new_name.sh
colorize.sh 100% 3299 3.2KB/s 00:00

SSH的强大功能

- **查看远程服务器的cpu信息**
  - ssh -l www-online 192.168.110.34 "cat /proc/cpuinfo"

```bash
#!/bin/bash
uptime >> 'uptime.log'
exit 0
```

- **执行远程服务器的sh文件**
  - 首先在远程服务器的/home/www-online/下创建一个uptimelog.sh脚本
  - 使用chmod增加可执行权限
  - 在本地调用远程的uptimelog.sh   `ssh -l www-online 192.168.110.34 "/home/www-online/uptimelog.sh"`
  - 执行完成后,在远程服务器的/home/www-online/中会看到uptime.log文件，显示uptime内容

```
www-online@nmgwww34:~$ tail -f uptime.log
21:07:34 up 288 days,  8:07,  1 user,  load average: 0.05, 0.19, 0.31
```

http://blog.csdn.net/fdipzone/article/details/23000201

SSH的强大功能

```
#!/bin/bash
while :
do
  uptime >> 'uptime.log'
  sleep 1
done
exit 0
```

- **执行远程后台运行sh**
  - 首先把uptimelog.sh修改一下,修改成循环执行的命令。作用是每一秒把uptime写入uptime.log

```
www-online@onlinedev01:~$ ssh -l www-online 192.168.110.34 "/home/www-online/uptimelog.sh &"
www-online@192.168.110.34's password:
```

ssh 调用远程命令后不能自动退出解决方法

可以将标准输出与标准错误输出重定向到/dev/null，这样就不会一直处于等待状态。

```
www-online@onlinedev01:~$ ssh -l www-online 192.168.110.34 "/home/www-online/uptimelog.sh > /dev/null 2>&1 &"
www-online@192.168.110.34's password:
www-online@onlinedev01:~$
```

```
#!/bin/bash
ssh -f -n -l www-online 192.168.110.34 "/home/www-online/uptimelog.sh &" # 后台运行ssh
pid=$(ps aux | grep "ssh -f -n -l www-online 192.168.110.34 /home/www-online/uptimelog.sh" | awk '{print $2}' | sort -n | head -n 1) #获取进程号
echo "ssh command is running, pid:${pid}"
sleep 3 && kill ${pid} && echo "ssh command is complete" # 延迟3秒后执行kill命令，关闭ssh进程，延迟时间可以根据调用的命令不同调整
exit 0
```

```
www-online@onlinedev01:~$ ./ssh_uptimelog.sh
www-online@192.168.110.34's password:
ssh command is running, pid:10141
ssh command is complete
www-online@onlinedev01:~$
```

可以看到，3秒后会自动退出

- 这个ssh进程会一直运行在后台，浪费资源，需要自动清理这些进程。→ 在ssh执行完成后kill掉ssh这个进程来实现。
- 创建一个sh执行ssh的命令,需要用到ssh的 -f 与 -n 参数，需要ssh也以后台方式运行，这样才可以获取到进程号进行kill操作。

内容

- Linux Introduction
- Connecting
- Linux Interaction
- I/O Redirection
- The Filesystem
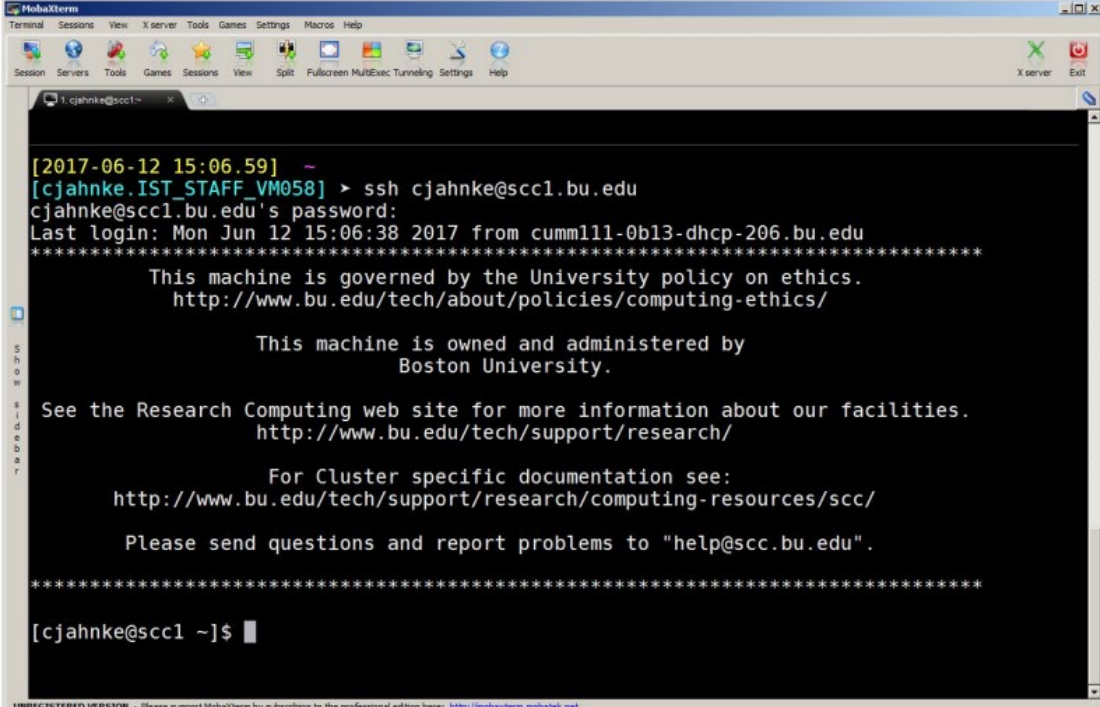- Processes & Job Control
- Editors
- Creating and Running Code

3. Linux Interaction

Shell, Prompt, Commands and System Use

Linux: The Shell

- Program that interprets commands and sends them to the OS
- Provides:
  - Built-in commands
  - Programming control structures
  - Environment variables
  - Linux supports multiple shells.
  - The default on SCC is Bash.

"Bash" = "Bourne-again Shell"
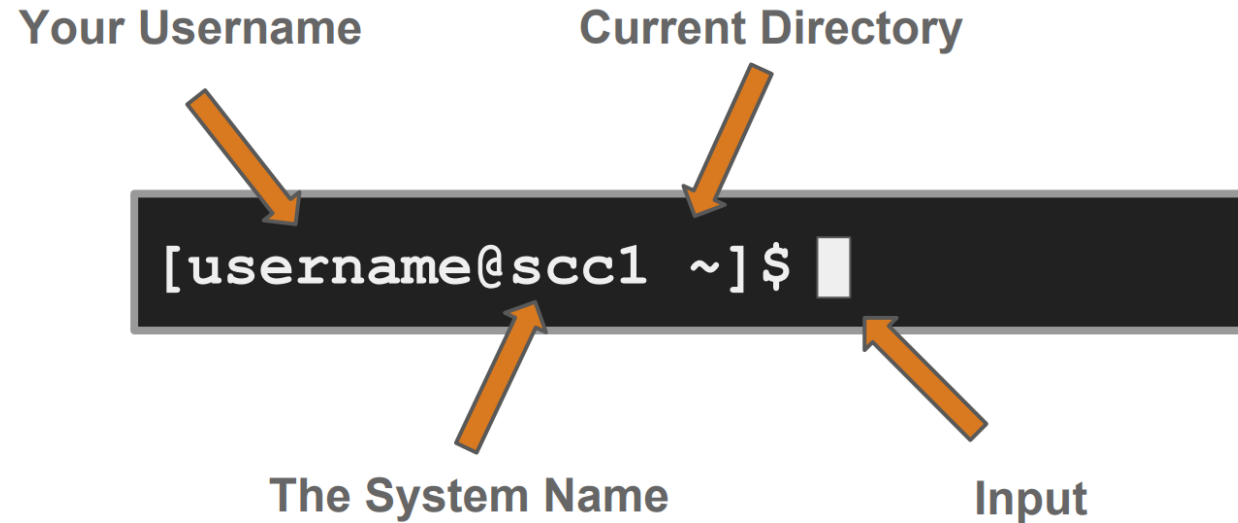(GNU version of ~1977 shell written
by Stephen Bourne)

Linux: The "prompt"

In Linux " ~ " is a shorthand for your home directory.

Linux: Command Basics

```
[username@scc1 ~]$ command --option argument
```

- **Command**: Command/program that does one thing

- **Options**: Change the way a command does that one thing
  - ○ Short form: Single-dash and one letter e.g. ls -a
  - ○ Long form: Double-dash and a word e.g. ls --all

- **Argument**: Provides the input/output that the command interacts with.

For more information about any command, use man or info (e.g. "man ls")

Commands: Hands-On

- After you connect, type

  - **whoami**   # my login
  - **hostname**   # name of this computer
  - **echo "Hello, world"**   # print characters to screen
  - **echo $HOME**   # print environment variable
  - **echo my login is $(whoami)**   # replace $(xx) with program output
  - **date**   # print current time/date
  - **cal**   # print this month's calendar
  - **shazam**   # bad command

Commands: Hands-On Options

- Commands have three parts; **command**, **options** and **arguments/parameters**.

- Example: **cal –j 3 1999**. "cal" is the command, "-j" is an option (or switch), "3" and "1999" are arguments/parameters.

```
[username@scc1 ~]$ cal -j 3 1999
```

- What is the nature of the prompt?

- What was the system's response to the command?

Commands: Selected text processing utilities

- **awk**   Pattern scanning and processing language
- **cat**   Display file(s)
- **cut**   Extract selected fields of each line of a file
- **diff**   Compare two files
- **grep**   Search text for a pattern
- **head**   Display the first part of files
- **less**   Display files on a page-by-page basis
- **sed**   Stream editor (esp. search and replace)
- **sort**   Sort text files
- **split** Split files
- **tail**   Display the last part of a file
- **tr**   Translate/delete characters
- **uniq**   Filter out repeated lines in a file
- **wc**   Line, word and character count

NOT ALL. Just a few of the commands for text processing

Variables and Environment Variables

- Variables are named storage locations.
    USER=augustin
    foo="this is foo's value"

- "Environment variables" are variables *used* and *shared* by the shell
    For example, **$PATH** tells the system where to find commands.

- Environment variables are *shared with programs* that the shell runs.

Bash variables

- To create a new variable, use the assignment operator '='

  [**username@scc1 ~**]$ foo="this is foo's value"

- The foo variable can be printed with echo

  [**username@scc1 ~**]$ echo $foo

  this is foo's value

- To make $foo visible to programs run by the shell (i.e., make it an "environment variable"), use export:

  [**username@scc1 ~**]$ export foo

Environment Variables

- To see all currently defined environment variable, use **printenv**:

```
[username@scc1 ~]$ printenv
HOSTNAME=scc1
TERM=xterm-256color
SHELL=/bin/bash
HISTSIZE=1000
TMPDIR=/scratch
SSH_CLIENT=168.122.9.131 37606 22
SSH_TTY=/dev/pts/191
USER=cjahnke
MAIL=/var/spool/mail/cjahnke
PATH=/usr3/bustaff/cjahnke/apps/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin
PWD=/usr3/bustaff/cjahnke/linux-materials
LANG=C
MODULEPATH=/share/module/bioinformatics:/share/module/chemistry
SGE_ROOT=/usr/local/ogs-ge2011.11.p1/sge_root
HOME=/usr3/bustaff/cjahnke
```

Command History and Command Line Editing

- Try the **history** command

- Choose from the command history using the up ↑ and down ↓ arrows

- To redo your last command, try **!!**

- To go further back in the command history try !, then the number as shown by history (e.g., **!132**). Or, !ls, for example, to match the most recent '**ls**' command.

- What do the left ← and right → arrow do on the command line?

- Try the <**Del**> and <**Backspace**> keys

Help with Commands

- Type
  - ○ date –-help
  - ○ man date
  - ○ info date

- ● BASH built-ins
  - ○ A little different from other commands
  - ○ Just type the command 'help'
  - ○ Or 'man bash'

Yes, you can always Google/百度 it.

On using 'man' with 'less'

- The **man** command outputs to a pager called **less**, which supports many ways of scrolling through text:

○	Space, f	# page forward
○	b	# page backward
○	<	# go to first line of file
○	>	# go to last line of file
○	/	# search forward (n to repeat)
○	?	# search backward (N to repeat)
○	h	# display help
○	q	# quit help

内容

- Linux Introduction
- Connecting
- Linux Interaction
- I/O Redirection
- I/O redirection with pipes
- The Filesystem
- Processes & Job Control
- Editors
- Creating and Running Code

4. I/O Redirection

I/O redirection with pipes

- Many Linux commands print to "standard output", which defaults to the terminal screen. The '|' (pipe) character can be used to divert or "redirect" output to another program or filter.

○ w   # show who's logged on
○ w | less   # pipe into the 'less' pager
○ w | grep 'tuta'   # pipe into grep, print lines containing 'tuta'
○ w | grep –v 'tuta'   # print only lines not containing 'tuta'
○ w | grep 'tuta' | sed s/tuta/scholar/g   # replace all 'tuta' with 'scholar'

More examples of I/O redirection

- Try the following (use up arrow to avoid retyping each line):

  - w | wc   # count lines
  - w | cut –d ' ' –f1 | sort   # sort users
  - w | cut –d ' ' –f1 | sort | uniq   # eliminate duplicates

- We can also redirect output into a file:

  - w | cut –d ' ' –f1 | sort | uniq > users

- Note that 'awk' can be used instead of 'cut':

  - w | awk '{print $1;}' | sort | uniq > users

- Quiz:

  - How might we count the number of distinct users currently logged in? For extra credit, how can we avoid over-counting by 2? (Hint: use 'tail'.)

内容

- Linux Introduction
- Connecting
- Linux Interaction
- I/O Redirection
- The Filesystem
- Processes & Job Control
- Editors
- Creating and Running Code

5. The Filesystem

The Linux File System

- The structure resembles an upside-down tree
- Directories (a.k.a. folders) are collections of files and other directories.
- Every directory has a parent except for the root directory.
- Many directories have subdirectories.

File structure in GNU / Linux

© skill2die4@yahoo.com

/ "ROOT"

/ BIN
"ESSENTIAL BINARIES"

CAT
CHGRP
CHMOD
CHOWN
CP
DATA
DD
DF
DMESG
ECHO
FALSE
HOSTNAME
KILL
LN
LOGIN
LS
MKDIR
MKNOD
MORE
MOUNT
MV
PS
PWD
RM
RMDIF
SED
SH
STTY
SU
SYNCH
TRUE
UMOUNT
UNAME

/ BOOT
"STATIC FILES OF
BOOT LOADER "

KERNEL
SYSTEM.MAP
VMLINUZ
INITRD
GRUB
MODULE.INFO
BOOT

/ ETC
"HOST SPECIFIC
SYSTEM CONFIG"

CSH.LOGIN
EXPORTS
FSTAB
FTPUSERS
GATEWAYS
GETTYDEFS
GROUP
HOST.CONF
HOSTS
HOSTS.ALLOW
HOSTS.DENY
HOSTS.EQUIV
HOSTS.LPD
INETD.CONF
INITTAB
ISSUE
LS.SO.CONF
MOTD
MTAB
MTOOLS
NETWORKS
PASSWD
PRINTCAP
PROFILE
PROTOCOLS
RESOLV.CONF
RPC
SECURETTY
SERVICES
SHELLS
SYSLOG.CONF

/ OPT
" CONFIG FILE
FOR ADD ON
APPLICATION
SOFTWARE "

/ USR
" SHAREABLE AND
READ-ONLY DATA "

/ LOCAL
"LOCAL
SOFTWARE"
/BIN
/GAMES
/INCLUDE
/LIB
/MAN
/SBIN
/SHARE
/SRC
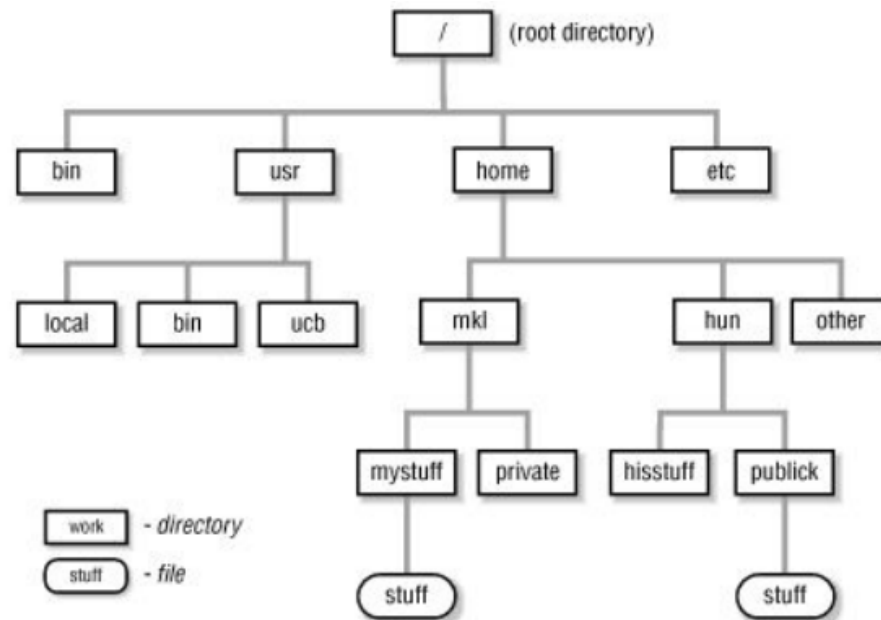
/ SHARE
" STATIC DATA
SHAREABLE
AMONG ALL
ARCHITECTURES "

/ MAN
"MANUAL PAGES"
/MAN1 "user programs"
/MAN2 "system calls"
/MAN3 "lib functions"
/MAN4 "special file"
/MAN5 "file formats"
/MAN6 "games"
/MAN7 "misc."
/MAN8 "system admin."

/ BIN
"MOST USER COMMANDS"

/ INCLUDE
"STANDARD INCLUDE
FILES FOR 'C' PROG."

/ LIB
" OBJ , BIN , LIB
FILES FOR PROG.
AND PACKAGES "

/ SBIN
"NON ESSENTIAL
BINARIES"

/ VAR
"VARIABLE DATA FILES"

/ CACHE
"APPLICATION
CACHE DATA"

/ LIB
" VARIABLE STATE
INFORMATION
REMAINS AFTER
REBOOT "

/ YP
" DATA FOR
NIS SERVICES "

/ LOCK
"LOCK FILES FOR
SHARED RESOURCES"

/ OPT
" VARIABLE DATA OF
PACKAGES INSTALLED"

/ RUN
"INFO OF SYSTEM
SINCE IT WAS BOOTED"

/ TMP
"AVAILABLE FOR PROG."

/ SPOOL
"DATA AWAITING
PROCESSING "
/LPD
/MQUEUE
/NEWS
/RWHO
/UUCP

/ LOG
"LOG FILES
AND DIR"
LASTLOG
MESSAGES
WTMP

/ SBIN
"SYSTEM BINARIES"
FASTBOOT
FASTHALT
FDISK
FSCK
GETTY
HALT
IFCONFIG
INIT
MKFS
MKSWAP
REBOOT
ROUTE
SWAPON
SWAPOFF
UPDATE

/ TMP
"TEMPERORY FILES
DELETED ON BOOTUP"

/ DEV
"LOCATION OF SPECIAL
OR DEVICE FILES
[CONTAINS MAKEDEV]"

/ HOME
" USER HOME
DIRECTORIES"

/ LIB
" LIBRARY AND
KERNEL MODULES"

/ MNT
" MOUNT FILES
FOR TEMPERORY
FILESYSTEMS "

/ OPT
" ADD-ON APPLICATION
SOFTWARE "

/ ROOT
"HOME DIR. FOR
ROOT USER"

Essential navigation commands
○ **pwd** print current directory
○ **ls** list files
○ **cd** change directory

We use pathnames to refer to files and directories in the Linux file system.
There are two types of pathnames:
○ **Absolute** – The full path to a directory or file; begins with /
○ **Relative** – A partial path that is relative to the current working directory; does not begin with /

Special characters interpreted by the shell for filename expansion:
○ **~** your home directory (e.g., /usr1/tutorial/tuta1)
○ **.** current directory
○ **..** parent directory
○ ***** wildcard matching any filename
○ **?** wildcard matching any character
○ **TAB** try to complete (partially typed) filename
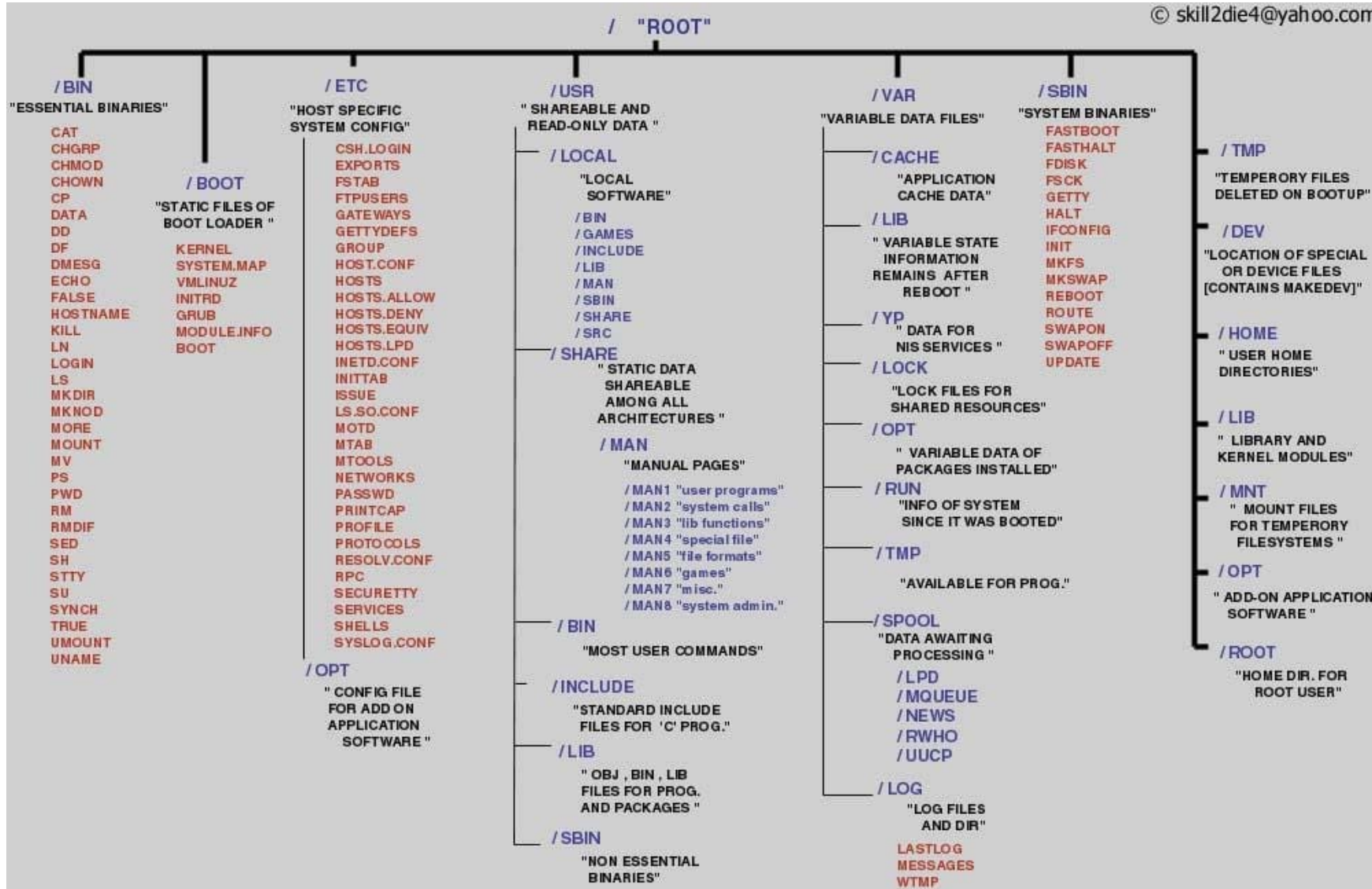
Examples:
○ **pwd** Print working (current) directory
○ **cd ..** Change directory to the "parent" directory
○ **cd /usr/local** Change directory to /usr/local/lib

○ **cd /** Change directory to the "root"
○ **ls -d pro*** Listing of only the directories starting with "pro"
○ **cd ~** Change to home directory (could just type 'cd')

Creating a new File

- The easiest way to create an empty file is using
- touch
- touch [flags] <file>
    - Adjusts the timestamp of the specified file.
    - With no flags uses the current date and time.
    - If the file does not exist, touch creates it.

- File extensions (.txt, .c, .py, etc) often don't matter in Unix.
- Using touch to create a file results in a blank plain-text file (so you don't necessarily have to hadd .txt to it).

Creating a new Directory

- No magic here...
- Make directory
- mkdir [flags] <directory1> <directory2> <...>
    - Can use relative or absolute paths.
        - a.k.a. you are not restricted to making directories in the current directory only.
- Need to specify at least one directory name.
- Can specify multiple, separated by spaces.
- The -p flag is commonly used in scripts: do not fail if directory already exists.
- By default, the mkdir command fails if you give it a directory that already exists.

File Deletion

- Warning: once you delete a file (from the command line) there is no easy way to recover the file.

- Remove File

- rm [flags] <filename>
  - Removes the file <filename>.
  - Remove multiple files with wildcards (more on this later).
  - Remove every file in the current directory: rm *
  - Remove every .jpg file in the current directory: rm *.jpg
  - Prompt before deletion: rm -i <filename>

Deleting Directories

- By default, rm cannot remove directories. Instead we use...

- Remove directory

- rmdir [flags] <directory>
  - Removes an empty directory.
  - Throws an error if the directory is not empty.
  - You are encouraged to use this command: failing on non-empty can and will save you!

- To delete a directory and all its subdirectories, we pass rm the flag -r (for recursive), e.g. rm -r /home/sven/oldstuff

Copy That!

- Copy

- cp [flags] <file> <destination>
  - Copies from one location to another.
  - To copy multiple files, use wildcards (such as *).
  - To copy a complete directory: cp -r <src> <dest>

Move it!

- Unlike the cp command, the move command automatically recurses for directories.
- Move
- mv [flags] <source> <destination>
  - Moves a file or directory from one place to another.
  - Also used for renaming, just move from <oldname> to <newname>.
  - E.g. mv badFolderName correctName

The ls Command

- Useful options for the "`ls`" command:

○	`ls -a`	List all files, including hidden files beginning with a "`.`"
○	`ls -ld *`	List details about a directory and not its contents
○	`ls -F`	Put an indicator character at the end of each name
○	`ls -l`	Simple long listing
○	`ls -lR`	Recursive long listing
○	`ls -lh`	Give human readable file sizes
○	`ls -lS`	Sort files by file size
○	`ls -lt`	Sort files by modification time (very useful!)

Some Useful File Commands

- `cp [file1] [file2]`   copy file
- `mkdir [name]`   make directory
- `rmdir [name]`   remove (empty) directory
- `mv [file] [destination]` move/rename file
- `rm [file]`   remove (-r for recursive)
- `file [file]`   identify file type
- `less [file]`   page through file
- `head -n N [file]`   display first N lines
- `tail -n N [file]`   display last N lines
- `ln –s [file] [new]`   create symbolic link
- `cat [file] [file2...]` display file(s)
- `tac [file] [file2...]` display file in reverse order
- `touch [file]`   update modification time
- `od [file]`   display file contents, esp. binary

Manipulating files and directories

- Examples:

```
○ cd # The same as cd ~
○ mkdir test
○ cd test
○ echo 'Hello everyone' > myfile.txt
○ echo 'Goodbye all' >> myfile.txt
○ less myfile.txt
○ mkdir subdir1/subdir2 # Fails. Why?
○ mkdir -p subdir1/subdir2 # Succeeds
○ mv myfile.txt subdir1/subdir2
○ cd ..
○ rmdir test # Fails. Why?
○ rm -rf test # Succeeds
```

Symbolic links

- Sometimes it is helpful to be able to access a file from multiple locations within the hierarchy.

- On a Windows system, we might create a "shortcut."

- On a Linux system, we can create a symbolic link:

○ `mkdir foo` # make foo directory
○ `touch foo/bar` # create empty file
○ `ln -s foo/bar .` # create link in current dir.

Finding a needle in a haystack

- The **find** command has a rather unfriendly syntax, but can be exceedingly helpful for locating files in heavily nested directories.

- Examples:

  - ○ **find ~ -name bu -type d** # search for "bu" directories in **~**
  - ○ **find . -name my-file.txt** # search for my-file.txt in .
  - ○ **find ~ -name '*.txt'** # search for "***.txt**" in **~**

- Quiz:

  - ○ Can you use **find** to locate a file called "needle" in your haystack directory?
  - ○ Extra credit: what are the contents of the "needle" file?

Recap

ls	list directory contents
cd	change directory
pwd	print working directory
rm	remove file
rmdir	remove directory
cp	copy file
mv	move file

Users and Groups

- Like most OS's, Unix allows multiple people to use the same machine at once. The question: who has access to what?
  - Access to files depends on the users' account.
  - All accounts are presided over by the Superuser, or root account.
  - Each user has absolute control over any files they own, which can only be superseded by root.
  - Files can also be owned by a group, allowing more users to have access.

File Ownership

- You can discern who owns a file many ways, the most immediate being ls -l

- Permissions with **ls**
  - >>> ls -l Makefile
  - -rw-rw-r--. 1 sven users 4.9K Jan 31 04:42 Makefile
                 sven # the user
                      users # the group

- The third column is the *user*, and the fourth column is the *group*.

What is this RWX Nonsense?

- R = read, W = write, X = execute.

- rwxrwxrwx
  - User permissions.
  - Group permissions.
  - Other permissions (a.k.a. neither the owner, nor a member of the group).

- Directory permissions begin with a d instead of a -.

An example

- What would the permissions `-rwxr-----` mean?
    - It is a file.
    - User can read and write to the file, as well as execute it.
    - Group members are allowed to read the file, but cannot write to or execute.
    - Other cannot do *anything* with it.

Changing Permissions

- Change Mode
- chmod <mode> <file>
  - Changes file / directory permissions to <mode>.
  - The format of <mode> is a combination of three fields:
  - Who is affected: a combination of u, g, o, or a (all).
  - Use a + to add permissions, and a – to remove.
  - Specify type of permission: any combination of r, w, x.
  - Or you can specify mode in octal: user, then group, then other.
    - e.g. 777 means user=7, group=7, other=7 permissions.
    - chmod 644 examplefile.txt

Changing Ownership

- Changing the group
- Change Group
- chgrp group <file>
  - Changes the group ownership of <file> to group.

- As the super user, you can change who owns a file:
- Change Ownership
- chown user:group <file>
  - Changes the ownership of <file>.
  - The group is optional.
  - The -R flag is useful for recursively modifying everything in a directory.

File Ownership, Alternate

- If you are like me, you often forget which column is which in
- ls -l...
- Status of a file or filesystem
- stat [opts] <filename>
  - Gives you a wealth of information, generally more than you will every actually need.
  - Uid is the user, Gid is the group.
    - BSD/OSX: use stat -x for standard display of this command.
  - Can be useful if you want to mimic file permissions you don't know.
    - Human readable: --format=%A, e.g. -rw-rw-r--
      - BSD/OSX: -f %Sp is used instead.
    - Octal: --format=%a (great for chmod), e.g. 664
      - BSD/OSX: -f %A is used instead.

```
[linux1@wrao ~]$ stat test.f
  File: `test.f'
  Size: 0          Blocks: 0          IO Block: 4096   regular empty file
Device: 5e01h/24065d       Inode: 257544      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 504/ linux1)  Gid: ( 504/ linux1)
Access: 2017-09-24 21:42:42.695917037 -0400
Modify: 2017-09-24 21:42:42.695917037 -0400
Change: 2017-09-24 21:42:42.695917037 -0400
```

内容

- Linux Introduction
- Connecting
- Linux Interaction
- I/O Redirection
- The Filesystem
- Processes & Job Control
- Editors
- Creating and Running Code

6. Processes & Job Control

Processes and Job Control

- As we interact with Linux, we create numbered instances of running programs called "processes." You can use the 'ps' command to see a listing of your processes (and others!). To see a long listing, for example, of all processes on the system try:

```
[username@scc1 ~]$ ps -ef
```

- To see all the processes owned by you and other members of the class, try:

```
[username@scc1 ~]$ ps -ef | grep tuta
```

- Use "**top**" to see active processes.

```
Tasks: 408 total,    1 running, 407 sleeping,    0 stopped,    0 zombie
Cpu(s):  0.3%us,  0.1%sy,  0.0%ni, 99.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   99022756k total, 69709936k used, 29312820k free,   525544k buffers
Swap:   8388604k total,        0k used,  8388604k free, 65896792k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7019 root      20   0  329m 137m 4852 S  4.0  0.1 217:01.56 sge_qmaster
38246 isw       20   0 88724 2764 1656 S  0.7  0.0   0:01.28 sshd
41113 cjahnke   20   0 13672 1512  948 R  0.7  0.0   0:00.03 top
 2324 root      20   0     0    0    0 S  0.3  0.0   0:21.82 kondemand/2
 7107 nobody    20   0 89572  10m 2400 S  0.3  0.0   2:18.05 gmond
27409 theavey   20   0 26652 1380  880 S  0.3  0.0   0:34.84 tmux
    1 root      20   0 25680 1604 1280 S  0.0  0.0   0:05.74 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.07 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.89 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:01.72 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 stopper/0
```

(refreshes every 2 seconds)

Foreground/background

- Thus far, we have run commands at the prompt and waited for them to complete. We call this running in the "**foreground**."

- Use the "**&**" operator, to run programs in the "**background**",
  - Prompt returns immediately without waiting for the command to complete:

```
[username@scc1 ~]$ mycommand &
[1] 54356                        ← process id
[username@scc1 ~]$
```

- Process Control Practice
  - Let's look at the "countdown" script, in your scripts folder for practice

```
[username@scc1 ~]$ cd ~/scripts
[username@scc1 ~]$ cat countdown
```

  - Make the script executable with chmod

```
[username@scc1 ~]$ chmod +x countdown
```

  - First, run it for a few seconds, then kill with Control-C.

```
[username@scc1 ~]$ ./countdown 100
100
99
98
^C                    ← Ctrl-C = (^C)
```

Foreground/background

- Now, let's try running it in the background with &:

```
[username@scc1 ~]$ ./countdown 60 &
[1] 54355
[username@scc1 ~]$
60
59
```

- The program's output is distracting, so redirect it to a file:

```
[username@scc1 ~]$ countdown 60 > c.txt &
[1] 54356
[username@scc1 ~]$
```

Process control

- Type '**ps**' to see your countdown process.

- Also, try running 'jobs' to see any jobs running in the background from this bash shell.

- To kill the job, use the 'kill' command, either with the five-digit process id:
  - **kill 54356**

- Or, you can use the job number (use 'jobs' to see list) with '%':
  - **kill %1**

Backgrounding a running job with C-z and 'bg'

- Sometimes you start a program, then decide to run it in the background.

```
[username@scc1 scripts]$ ./countdown 200 > c.out
^Z                                          ← Ctrl-Z = (^Z)
[1]+  Stopped                 ./countdown 200 > c.out

[username@scc1 scripts]$ bg
[1]+ ./countdown 200 > c.out &

[username@scc1 scripts]$ jobs
[1]+  Running                 ./countdown 200 > c.out &

[username@scc1 scripts]$
```

内容

- Linux Introduction
- Connecting
- Linux Interaction
- I/O Redirection
- The Filesystem
- Processes & Job Control
- Editors
- Creating and Running Code

7. Editors

File Editors

- gedit
  - Notepad-like editor with some programming features (e.g., syntax highlighting). Requires X-Windows.
- nano
  - Lightweight editor. Non-Xwindows.
- emacs
  - Swiss-army knife, has modes for all major languages, and can be customized. Formerly steep learning curve has been reduced with introduction of menu and tool bars. Can be used under Xwindows or not.
- vim
  - A better version of 'vi' (an early full-screen editor). Very fast, efficient. Steep learning curve. Popular among systems programmers. Terminal or X-Windows.

What is VIM?

- VIM is a powerful "lightweight" text editor.
- VIM actually stands for "Vi IMporoved", where vi is the predecessor.
- VIM can be installed on pretty much every OS these days.
- Allows you to edit things *quickly*, after the initial learning curve.

The 3 Main Modes of VIM

- Normal Mode:
  - Launching pad to issue commands or go into other modes.
  - Allows you to view the text, but not edit it directly (only through commands).
  - You can jump to normal mode by pressing ESCAPE.
- Visual Mode:
  - Used to highlight text and perform block operations.
  - Enter visual mode *from normal mode* by pressing v on your keyboard.
  - Visual Line: shift+v
  - Visual Block: ctl+v
  - Explanation: try them out, move your cursor around...you'll see it.
- Insert Mode:
  - Used to type text into the buffer (file).
  - Like any regular text-editor you've seen before.
  - Enter *from normal mode* with the i key.

Moving Around VIM

- Most of the time (these days at least), you can scroll with your mouse / trackpad.
- You can also use your arrow keys.
- By design, VIM shortcuts exist to avoid moving your hands at all. Use
  - h to go left.
  - j to go down.
  - k to go up.
  - l to go right.
- With that in mind, the true VIM folk usually map left caps-lock to be ESCAPE.

Useful Commands

`:help`	help menu, e.g. specify `:help v`
`:u`	undo
`:q`	exit
`:q!`	exit without saving
`:e [filename]`	open a different file
`:syntax [on/off]`	enable / disable syntax highlighting
`:set number`	turn line numbering on
`:set spell`	turn spell checking on
`:sp`	split screen horizontally
`:vsp`	split screen vertically
`<ctrl+w> <w>`	rotate between split regions
`:w`	save file
`:wq`	save file and exit
`<shift>+<z><z>`	hold shift and hit `z` twice: alias for `:wq`

内容

- Linux Introduction
- Connecting
- Linux Interaction
- I/O Redirection
- The Filesystem
- Processes & Job Control
- Editors
- Creating and Running Code

8. Creating and Running Code

"Hello, world" in C

- **cd** to "**~/c**", and read hello.c into your editor of choice.
- Modify the text on the printf line between "**[** " and "**]**" and save the file.
- Produce an executable file called "hello" by compiling the program with gcc:

```
[username@scc1 ~]$ gcc -o hello hello.c
```

- Run the program at the command line:

```
[username@scc1 ~]$ ./hello
```

- Optional: modify countdown script to run hello program

Obtaining the Supplementary Course Material

- See Linux tutorials:
    - http://www.tutorialspoint.com/unix/
    - Edx Linux intro [Google "edx linux"]
    - http://www.cse.sc.edu/~okeefe/tutorials/unixtut/