



พื้นฐานลิบุกช์

สำหรับนักศึกษาชั้นปีที่ 4 สาขาวิชาวิศวกรรมคอมพิวเตอร์

กรงฤทธิ์ กิติศรีวงศ์พันธุ์

Email : songrit@npu.ac.th

สาขาวิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนครพนม

ชุดคำสั่ง

- echo, cat, head, tail, grep, sed
- awk, cat, cd, chmod, chown, cd, cut, dd , echo, export, file, find, grep, id, ls, more, mv, ps, rm, sed, tail, top, xargs
- hexdump, nc, nmap, netstat, ifconfig
- gcc, g++, python, readelf, objdump, binwalk, gdb
- service, systemctl
- Shell script

ວັດຖຸປະສົງຄໍ

- 1. ທີ່ອຕົວແປຣພື້ນຈູານໃນຮະບບປົງປັດຕິການ
- 2. ກາຣຄວບຄຸມສຶກຮີໃນກາຣໃຊ້ໄຟລ໌
- 3. ກາຣຍກະດັບສຶກຮີ
- 4. ຄຳສັ່ງພື້ນຈູານ

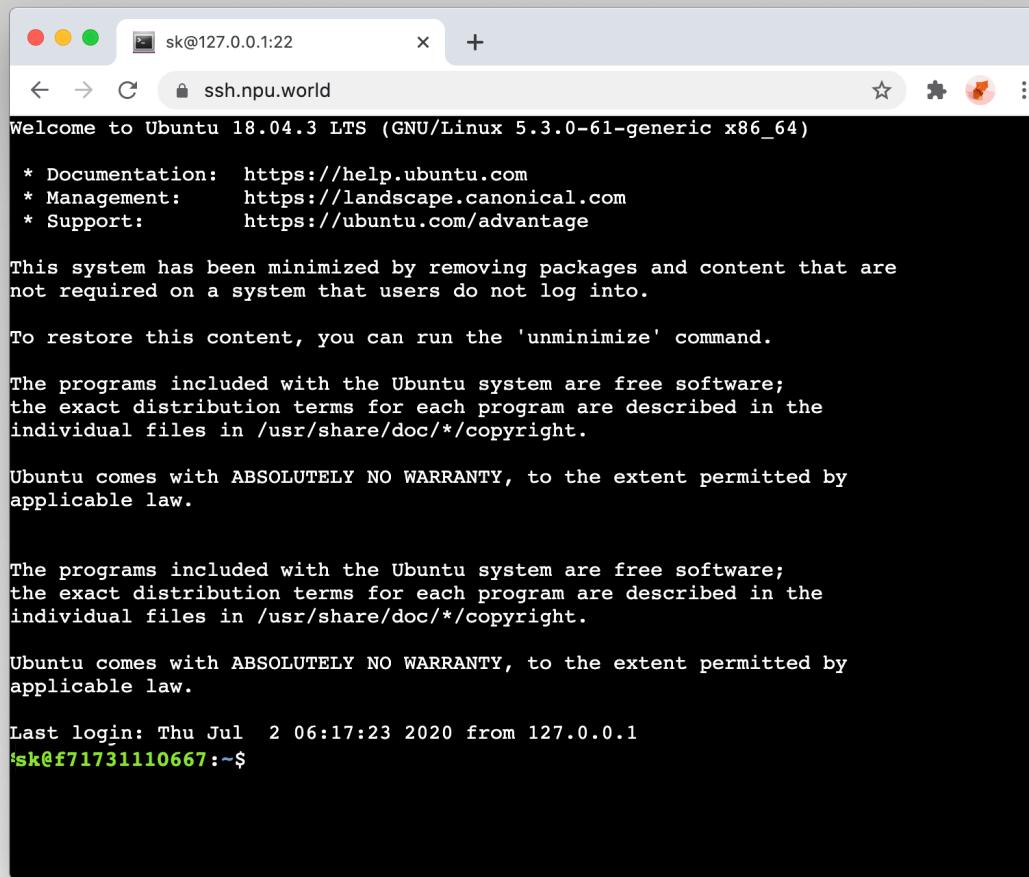
การรันโปรแกรม

- Execution file
- Permission
- PATH

ชื่อตัวแปรพื้นฐานในระบบปฏิบัติการ

- เมื่อบูรณาการ ถึงขั้นตอนสุดท้ายเป็นการเรียกโปรแกรม
- ระบบปฏิบัติการจะเรียก โปรแกรม **เชลล์**
 - ส่วนติดต่อกับผู้ใช้งาน เราเรียกว่า “เชลล์”(shell)
 - **เชลล์** เป็นได้ในรูปแบบ กูย(GUI) หรือแบบ Command line (CLI)
- ระบบปฏิบัติการลินุกซ์มีเชลล์ชื่อ Bash เป็นพื้นฐาน

Bash shell



A screenshot of a terminal window titled "sk@127.0.0.1:22" connected to "ssh.npu.world". The window displays the standard Ubuntu 18.04 LTS welcome message, which includes links for documentation, management, and support, a note about system minimization, instructions for restoring content, copyright information, and a disclaimer about warranty. The message concludes with the last login information and a prompt at the bottom: "sk@f71731110667:~\$".

```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.3.0-61-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Thu Jul  2 06:17:23 2020 from 127.0.0.1
sk@f71731110667:~$
```

สภาพแวดล้อมในเซลล์

- โปรแกรมเซลล์มีการสร้างตัวแปรสำหรับใช้งานในระบบ
- ตัวแปรคือ ชื่อเรียก

Variable = Value
ตัวแปร = ค่า

- ตัวอย่าง
 - A = 5

คำสั่งพื้นฐาน

- แสดงรายชื่อไฟล์ และ ไดเร็กทอรี
- ข้อมูลที่ได้จากการแสดงชื่อไฟล์
- การเปลี่ยนข้อมูลไฟล์

PATH, PWD, USER

ls cp mv rm tree

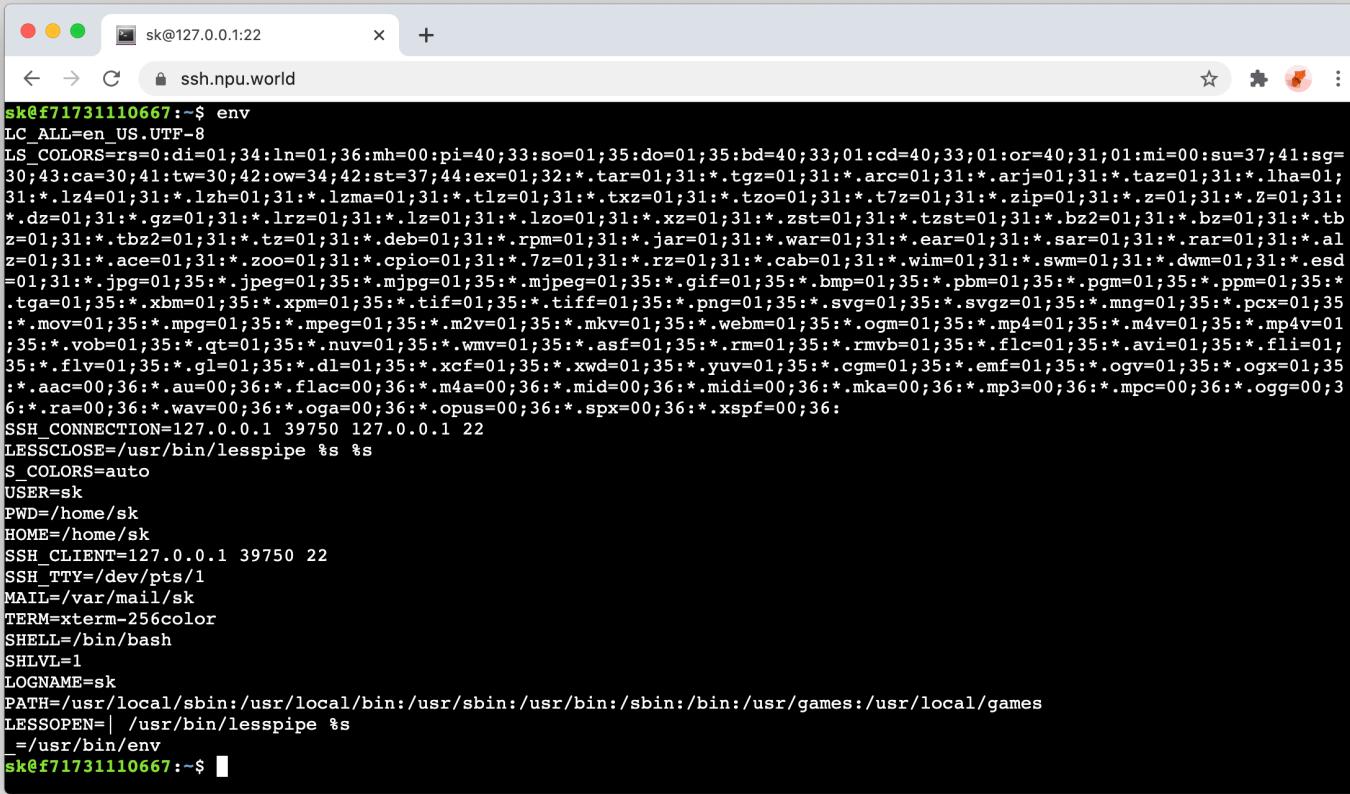
file tar wget find sha256sum

chmod chown

cat grep more tail awk sed string xargs

ตัวแปรในตั้งต้นของเชลล์

- คำสั่งดูตัวแปรตั้งต้น ของเชลล์
- env



```
sk@f71731110667:~$ env
LC_ALL=en_US.UTF-8
LS_COLORS=rs=0:di=0;34:ln=0;36:mb=00:pi=40;33:so=01;35:do=01;35:bd=40;33:01:cd=40;33:01:or=40;31:01:mi=00:su=37;41:sg=
30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;
31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.tz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:
*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tb
z=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.al
z=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd
=01;31:*.jpg=01;35:*.jpeg=01;35:*.mpg=01;35:*.mpeg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*
.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35
:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01
;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;
35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35
:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;3
6:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
SSH_CONNECTION=127.0.0.1 39750 127.0.0.1 22
LESSCLOSE=/usr/bin/lesspipe %s %s
S_COLORS=auto
USER=sk
PWD=/home/sk
HOME=/home/sk
SSH_CLIENT=127.0.0.1 39750 22
SSH_TTY=/dev/pts/1
MAIL=/var/mail/sk
TERM=xterm-256color
SHELL=/bin/bash
SHLVL=1
LOGNAME=sk
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
LESSOPEN=| /usr/bin/lesspipe %s
_= /usr/bin/env
sk@f71731110667:~$
```

ลักษณะของไฟล์

- ไฟล์ทั่วไป (Regular file) (-)
- ไดเรกทอรี่ (Directory) (d)
- ลิงค์ไปยังไฟล์อื่น (link) (l)

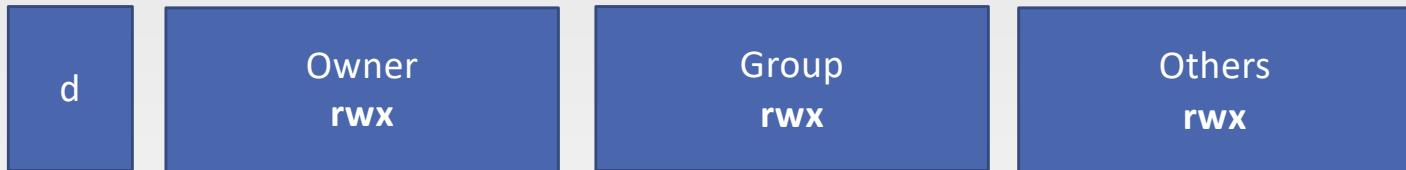
permission

```
cpe@server:~$ ls -l
total 4
-rw-rw-r-- 1 cpe cpe    0 Aug 27 15:51 abc.txt
1rwxrwxrwx 1 cpe cpe 23 Aug 27 15:51 dhclient.conf -> /etc/dhcp/dhclient.conf
drwxrwxr-x 2 cpe cpe 4096 Aug 27 15:52 thedirectory
cpe@server:~$
```

- เรียกว่า permission

គុណសមប័តីផ្សា

- អាង (Read) → r
- ផើយប (Write) → w
- រាប (Execute) → x



```
cpe@server:~$ ls -l
total 4
-rw-rw-r-- 1 cpe cpe
lrwxrwxrwx 1 cpe cpe
drwxrwxr-x 2 cpe cpe
cpe@server:~$
```

ข้อมูลของไฟล์

```
total 19188
-rw-r----- 1 root shadow      934 Jul 21 08:18 shadow
-rw-r----- 1 root shadow     804 Jul 21 08:18 shadow-
-rw-r--r-- 1 root root       103 Feb 14 2019 shells
drwxr-xr-x 2 root root    4096 Feb 14 2019 skel
-rw-r--r-- 1 root root      100 Jun 25 2018 sss.conf
```

- ใช้คำสั่ง ls -l

- Permission
- Owner
- Group
- Size
- Date
- File name

ສຶກຮີໃນໄຟລ

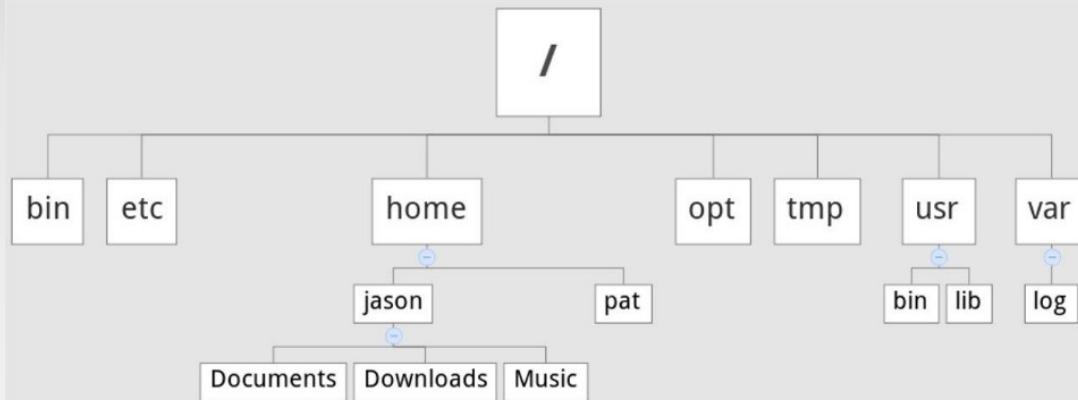
- ແປ່ງຜູ້ໃຊ້ເປັນ 3 ກລຸ່ມ
 - ເຈົ້າຂອງໄຟລ (Owner)
 - ກລຸ່ມເດືອກກັນ (Groups)
 - ດຽວໜ້າ (Others)
- Owner ເປັນຜູ້ສ້າງໄຟລ → ເຈົ້າຂອງໄຟລ
- Groups ຄື່ອກລຸ່ມຂອງ user ທີ່ມີສຶກຮີຕ່ອໄຟລ
- Others ຄື່ອກລຸ່ມອື່ນທີ່ໄມ້ໃຊ້ Owner ແລະ Groups



read / write / Execute

	Owner	Groups	Others
Read	Yes	Yes	No
Write	Yes	No	No
Execute	yes	No	No

โครงสร้างระบบจัดเก็บไฟล์



- จัดสตรรไฟล์เรียงตามลำดับชั้น (Tree)
- ใช้คำสั่ง ls (มาจาก list) แสดงรายชื่อไฟล์
- มี / อยู่ชั้นบนสุด เรียกว่า รูท (root)

/sbin

- เก็บชุดคำสั่งที่รันได้เฉพาะ root
- ผนเมใช้วิธีจำ secure binary (sbin)
- ตัวอย่างคำสั่ง
 - fdisk
 - fsck
 - ifconfig
 - init
 - mkfs
 - shutdown
 - reboot

ไฟล์และคำสั่ง

- โครงสร้างไฟล์
- ชนิดของไฟล์
- คำสั่งเกี่ยวกับไฟล์

បុណ្ណាសារ Windows

- នេះ Windows មี សារអត្ថៃ 3 បុណ្ណា
 - Directory (Folder)
 - Regular file
- ពេលវេលា regular file មិនមែនជាបុណ្ណា
 - ដូចជាអ្នកប្រើប្រាស់ (permission)
 - ធម្មតាសារ (hidden file)
 - ធម្មតាសារ execute
 - ធម្មតាសារ shortcut file
 - ធម្មតាសារ word, excel

ชนิดไฟล์ใน Linux

- ลิบุกซึมองว่า สิ่งที่ OS ติดต่อได้เป็นไฟล์ทั้งหมด
- แต่ลักษณะการติดต่อนั้นขึ้นอยู่กับคุณสมบัติของไฟล์
- เช่น คำสั่ง ls ใช้ได้กับทั้ง
 - ไฟล์
 - ไดเร็กทอรี
 - อุปกรณ์ฮาร์ดแวร์ , หน่วยความจำ
 - Driver
- มีการกำหนดสิทธิ์เข้าใช้ไฟล์
 - เป็น 3 กลุ่ม
 - เจ้าของไฟล์ กลุ่มของผู้ใช้ และ คนอื่น

คำสั่งเกี่ยวกับข้องกับไฟล์

- คำสั่งก่อระบบไฟล์
- คำสั่งจัดการไฟล์
- คำสั่งกำหนดสิทธิ์แก้ไฟล์
- คำสั่งเลือกแสดงผล

คำสั่งท่องระบบไฟล์

- **pwd** (Present Working Directory)
 - → ดูตำแหน่งปัจจุบัน
- **ls** (List) → ดูรายชื่อไฟล์ในตำแหน่งปัจจุบัน
- **cd** (Change directory) → เปลี่ยนตำแหน่งที่อยู่

ສຶກຮີເຮັ່ນຈາກຜູ້ສ່ວັງໄຟລ໌

Ho Ho ..



Owner

ອ່ານ

ເຂົ້ານ

ຮັບໂປຣແກຣມ



Group

ອ່ານ

ເຂົ້ານ

ຮັບໂປຣແກຣມ



Others

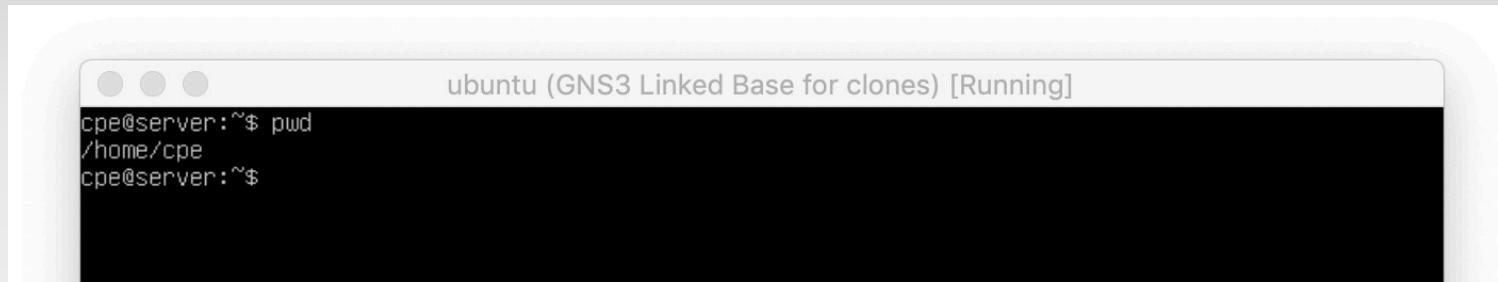
ອ່ານ

ເຂົ້ານ

ຮັບໂປຣແກຣມ

คำสั่ง pwd

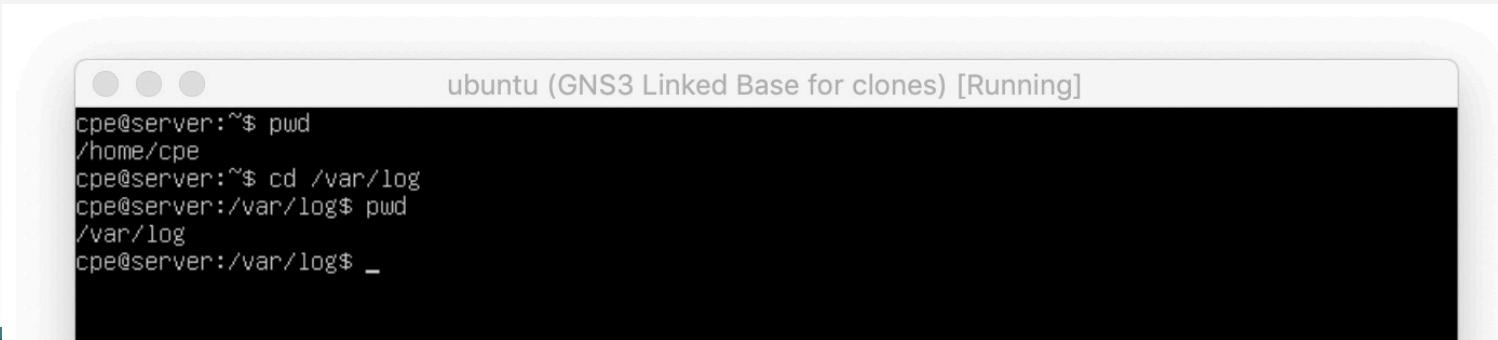
- pwd



```
cpe@server:~$ pwd
/home/cpe
cpe@server:~$
```

- cd /var/log

- pwd



```
cpe@server:~$ pwd
/home/cpe
cpe@server:~$ cd /var/log
cpe@server:/var/log$ pwd
/var/log
cpe@server:/var/log$ _
```

คำสั่ง cd

- ใช้เปลี่ยนตำแหน่งกีออยู่ปัจจุบัน
- cd <ไดเร็คทอรี่ที่ต้องการเปลี่ยน>
- cd อย่างเดียวแปลว่ากลับบ้าน เช่น user : cpe
 - /home/cpe

คำสั่ง ls

- คำสั่ง list ไฟล์
- ls -a แสดงรายการไฟล์ทั้งหมด รวมไฟล์
 - ซ่อน (ไฟล์ที่ขึ้นต้นด้วย . เป็นไฟล์ซ่อนอัตโนมัติ เช่น .bashrc)
 - ไฟล์ซ่อน . อย่างเดียวหมายถึง ไดเร็กทอรีปัจจุบัน
 - ไฟล์ซ่อน .. หมายถึงไดเร็กทอรีบน หนึ่งขั้น
- ls -l แสดงรายการแบบรายละเอียดครบ (long)

```
drwxr-xr-x 4 root root      4096 Feb 14  2019 X11
drwxr-xr-x 4 root root      4096 Feb 14  2019 xdg
-rw-r--r-- 1 root root     477 Mar 16  2018 zsh_command_not_found
cpe@server:/etc$ _
```

คำสั่งเกี่ยวกับข้องกับไฟล์

- คำสั่งท่องระบบไฟล์
- คำสั่งจัดการไฟล์
- คำสั่งกำหนดสิทธิ์แก้ไฟล์
- คำสั่งเลือกแสดงผล

การจัดการไฟล์

- สร้างไฟล์
- ดูเนื้อหาในไฟล์
- ลบไฟล์
- คัดลอกไฟล์
- สร้างลิงค์
- กำหนดผู้มีสิทธิ์ใช้ไฟล์

สร้างไฟล์

- สร้างไฟล์เปล่า
 - **touch abc.txt** → สร้างไฟล์เปล่าชื่อ abc.txt
- สร้างไฟล์ด้วยคำสั่ง echo
 - echo "Hello" > abc.txt (ไฟล์ abc.txt มีเนื้อหา Hello)
 - echo "Hello2" >> abc.txt (เพิ่ม Hello2 ต่อท้าย ในไฟล์ abc.txt)
- สร้างไฟล์คำสั่ง dd โดยกำหนดขนาดเอง
 - สร้างไฟล์ 1GB
 - **dd if=/dev/zero of=file.txt count=1024 bs=1048576**
 - if → input file, of → output file
 - count จำนวนรอบ
 - bs → block size หน่วย ไบต์

สร้างไฟล์ (ชนิดไดเร็คทอรี่)

- คำสั่งสร้างไดเร็คทอรี
 - mkdir abc → สร้างชื่อ abc
 - -p ใช้สร้าง subdirectory ที่ยังไม่เคยมี
 - mkdir -p ~/abc/1/2/3
 - ไม่เคยมีไดเร็คทอรี 1/2/3
 - ช่วยให้สร้างไฟล์ในคำสั่งเดียว

คำสั่งดูไฟล์

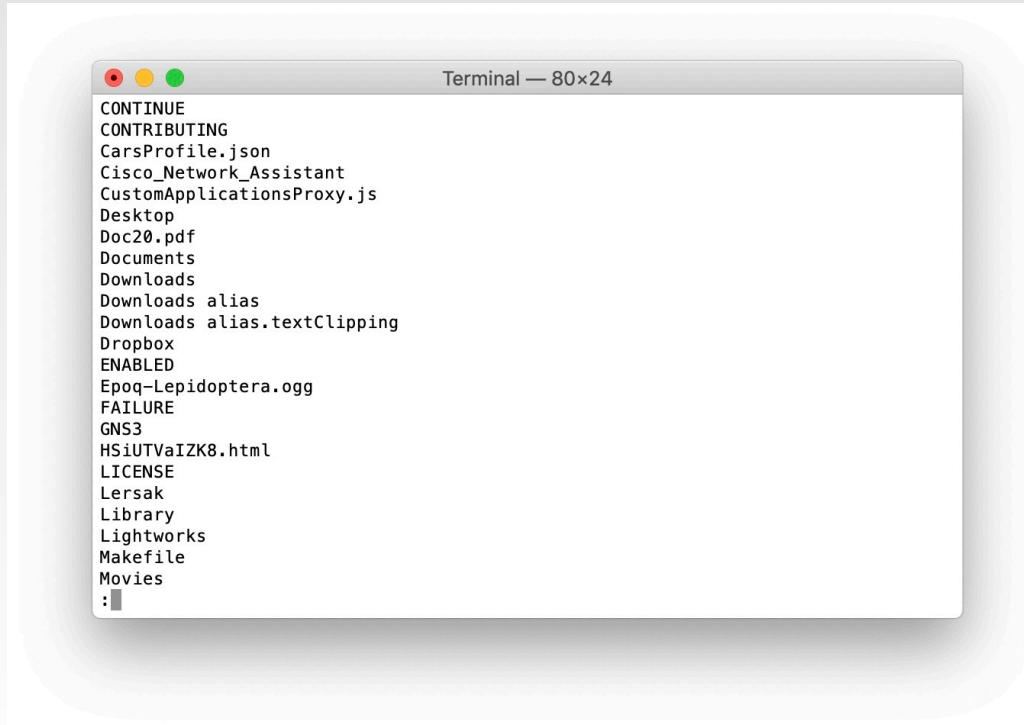
- cat
- more
- head
- tail

คำสั่ง cat

- อ่าน text file
- cat abc.txt → อ่านข้อความในไฟล์ abc.txt
- ใช้ร่วมกับเครื่องหมาย > เพื่อเปลี่ยนทิศทางเช่น
 - cat abc.txt > abc-2.txt ← อ่านไฟล์ abc.txt และเขียนไฟล์ใหม่เป็น abc-2.txt
 - cat c.txt >> abc.txt ← อ่านไฟล์ c.txt และเขียนต่อท้าย abc.txt

คำสั่ง more

- ดูไฟล์หนึ่งหน้าจอ
- เคาะ space bar เลื่อนหน้าจอ



A screenshot of a Mac OS X Terminal window titled "Terminal — 80x24". The window contains a list of file names and directory names, demonstrating the use of the "more" command. The list includes: CONTINUE, CONTRIBUTING, CarsProfile.json, Cisco_Network_Assistant, CustomApplicationsProxy.js, Desktop, Doc20.pdf, Documents, Downloads, Downloads alias, Downloads alias.textClipping, Dropbox, ENABLED, Epoq-Lepidoptera.ogg, FAILURE, GNS3, HSiUTVaIZK8.html, LICENSE, Lersak, Library, Lightworks, Makefile, Movies, and a colon followed by a blank line. The window has the standard OS X title bar with red, yellow, and green buttons.

คำสั่ง head

- อ่านไฟล์นับจากส่วนหัว
- อ่านไฟล์ auth.log 10 บรรทัดนับจากต้นไฟล์
 - head /var/log/auth.log
 - head -10 /var/log/auth.log
 - head -5 /var/log/auth.log → อ่านไฟล์ 5 บรรทัด

คำสั่ง tail

- อ่านไฟล์เหมือน head แต่บันจากท้ายไฟล์
 - tail -5 /var/log/auth.log อ่าน 5 บรรทัด บันจากท้าย
- อ่านไฟล์และค่อยดูการอัพเดตไฟล์
 - tail -f /var/log/syslog
 - กด ctl+c ออกจากโปรแกรม

```
cpe@server:/etc$ tail -f /var/log/kern.log
Aug 27 10:24:39 server kernel: [ 75.499136] audit: type=1400 audit(1566901479.824:25): apparmor="STATUS" operation="profile_replace" profile="unconfined" name="/snap/core.snapd/snapd" pid=1208 comm="apparmor_parser"
Aug 27 11:32:32 server kernel: [ 4148.518972] e1000: enp0s3 NIC Link is Down
Aug 27 11:32:36 server kernel: [ 4152.551222] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
Aug 27 11:32:49 server kernel: [ 4164.646896] e1000: enp0s3 NIC Link is Down
Aug 27 11:32:51 server kernel: [ 4166.665534] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
Aug 27 12:24:12 server kernel: [ 5575.884176] e1000: enp0s3 NIC Link is Down
Aug 27 12:24:14 server kernel: [ 5577.900621] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
'C
cpe@server:/etc$ tail -f /var/log/kern.log
Aug 27 10:24:39 server kernel: [ 75.499136] audit: type=1400 audit(1566901479.824:26): apparmor="STATUS" operation="profile_replace" profile="unconfined" name="/snap/core/7396/usr/lib/snapd/snap-confine//mount-namespace-capture-helper" pid=1205 comm="apparmor_parser"
Aug 27 10:24:39 server kernel: [ 75.522257] audit: type=1400 audit(1566901479.848:27): apparmor="STATUS" operation="profile_replace" profile="unconfined" name="snap-update-ns.core" pid=1207 comm="apparmor_parser"
Aug 27 10:24:39 server kernel: [ 75.584493] audit: type=1400 audit(1566901479.912:28): apparmor="STATUS" operation="profile_replace" profile="unconfined" name="snap.core.hook.configure" pid=1208 comm="apparmor_parser"
Aug 27 11:32:32 server kernel: [ 4148.518972] e1000: enp0s3 NIC Link is Down
Aug 27 11:32:36 server kernel: [ 4152.551222] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
Aug 27 11:32:49 server kernel: [ 4164.646896] e1000: enp0s3 NIC Link is Down
Aug 27 11:32:51 server kernel: [ 4166.665534] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
Aug 27 12:24:12 server kernel: [ 5575.884176] e1000: enp0s3 NIC Link is Down
Aug 27 12:24:14 server kernel: [ 5577.900621] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
```

คำสั่งเกี่ยวกับข้องกับไฟล์

- คำสั่งท่องระบบไฟล์
- คำสั่งจัดการไฟล์
- คำสั่งกำหนดสิทธิ์แก้ไฟล์
- คำสั่งเลือกแสดงผล

คำสั่งเกี่ยวกับข้องกับไฟล์

- คำสั่งท่องระบบไฟล์
- คำสั่งจัดการไฟล์
- **คำสั่งกำหนดสิทธิ์แก่ไฟล์**
- คำสั่งเลือกแสดงผล

คำสั่งกำหนดสิทธิ์ไฟล์

- ใช้กำหนดคุณสมบัติ ข้อมูลควบคุมไฟล์
- **chmod** (change mode)
- **chown** (change owner)
- **chattr** (change attribute)

คำสั่ง chmod

- คำสั่ง chmod กำหนดสิทธิ์ตามผู้ใช้
- Permission ด้วยไบนาเรี่ย
- ให้สิทธิ์ แทนด้วย 1 , ไม่ให้สิทธิ์เป็น 0

d	rwx	rwx	rwx
—	---	---	---
Owner	Group	Others	

คำสั่ง chmod

- ให้เจ้าของไฟล์มีสิทธิ์ทุกอย่างแต่ผู้เดียว

0	111	000	000
—	----	----	----
Owner	Group	Others	

- $111_2 \ 000_2 \ 000_2 = 7 \ 0 \ 0$
- chmod 700 abc.txt**

คำสั่ง chmod

- ใช้กำหนด / เปลี่ยนแปลงลักษณะไฟล์ให้สามารถ
 - Read / Write / Execution
- **chmod 444 abc.txt**
 - กำหนดให้ไฟล์ abc.txt อ่านได้ทุกคน
- **chmod 400 abc.txt**
 - เอาสิทธิ์การอ่านออกจากไฟล์ ไม่ให้ใครอ่านได้ (นอกจากเจ้าของ)
- **chmod 666 abc.txt → ให้สิทธิ์เขียนไฟล์กับทุกคน**
- **chmod 755 → ให้สิทธิ์อ่านไฟล์ และให้สิทธิ์รันกับทุกคน**

คำสั่งเกี่ยวกับข้องกับไฟล์

- คำสั่งท่องระบบไฟล์
- คำสั่งจัดการไฟล์
- คำสั่งกำหนดสิทธิ์แก้ไฟล์
- คำสั่งเลือกแสดงผล

คำสั่งเลือกและลบ

- grep
- sed
- cut
- awk

คำสั่ง grep

- เลือกแสดงผล
- grep "test" abc.txt
 - เลือกแสดงผลเฉพาะบรรทัดที่มีข้อความ test
- grep -v "test" abc.txt
 - เลือกแสดงผลเฉพาะบรรทัดที่ไม่มีข้อความ test

คำสั่ง sed

- คำสั่งแทนที่ข้อความ
- 'ส่วนหน้า/ข้อความคันหา/ข้อความทดแทน/ส่วนหลัง'
- sed 's/ข้อความคันหา/ข้อความทดแทน/g'
 - **s** ใช้แทนการคันหาที่รวม ช่องว่าง (space)
 - ข้อความคันหาแบบตรงทุกคำ ลำดับถูก
 - ข้อความที่จะนำมาแทนที่
 - **g** ให้คันจนจบไฟล์ (ไม่หยุดที่การพบข้อความแรก)

คำสั่ง sed

- คำสั่งแทนที่ข้อความ
- sed "s/test number one>Hello world/g" abc.txt
 - แทนที่ข้อความ Hello world ด้วย test number one
 - และแสดงออกหน้าจอ
 - sed "s/test number one>Hello world/g" abc.txt > abc2.txt
 - เขียนลงไฟล์ abc2.txt

ວັນນີ້

- Introduction to Unix and other's family
- ຮະບບປະກົດຕາກາລິນຸກຊື່
 - ໄພລົດແລະຄໍາສັ່ງ
 - **Shell script**

Shell script

- Shell อ่านว่า เชล์
- เป็น User space สำหรับผู้ใช้งานระบบลินุกซ์
- Shell เป็นได้ทั้ง GUI หรือ CLI
- ในที่นี้หมายถึง CLI (Command line)
- โปรแกรม shell มีหลากหลาย
 - **sh (Bourne shell)**
 - **Bash (Bourne-Again shell)**
 - Csh (C shell)
 - ksh (Korn shell)

เขียนโปรแกรมด้วยเชลล์

- เรียกว่า script

- เป็นชุดคำสั่ง CLI รวมในไฟล์เดียว
- Shell มีการทำงานแบบเดียวกับ ภาษาโปรแกรม
 - มีโครงสร้างภาษา มีการกำหนดตัวแปร
 - มีชุดคำสั่งควบคุม : if-else, loop

- คำสั่งใช้กำหนด โปรแกรมเชลล์

```
#! /bin/bash
```

```
#! /bin/sh
```

- รันโปรแกรม

- กำหนดสิทธิ์ executable: \$ chmod +x script
- รันโปรแกรม : \$./script

Bash shell

- **Input**
 - prompting user
 - command line arguments
- **Decision:**
 - if-then-else
 - case
- **Repetition**
 - do-while, repeat-until
 - for
 - select
- **Functions**
- **Traps**

รับค่าอินพุตด้วย read

- คำสั่ง `read` รับค่าอินพุต

Syntax:

```
read varname [more vars]
```

- or

```
read -p "prompt" varname [more vars]
```

- words entered by user are assigned to `varname` and “`more vars`”
- last variable gets rest of input line

รับค่าอินพุต

- รับค่าอินพุต \$1 , \$2 , \$3 \$9
- โดยใช้ช่องว่างเป็นตัวแบ่ง
- ./sum.sh 3.3 4
- ใช้ตัวแปร sum.sh
 - #! /bin/bash
 - a=\$1
 - b=\$2
 - echo "\$a+\$b"|bc
- \$ chmod +x sum.sh
- \$./sum.sh 3.3 4

Special shell variables

Parameter	Meaning
\$0	ชื่อไฟล์ที่รับ
\$1-\$9	กำหนดรับตัวแปรได้ตั้งแต่ \$1 ถึง \$9
\$#	จำนวนพารามิเตอร์
\$*	อ่านทุกค่าอันพูกเป็น string , "\$*" ไฟล์เดียว
\$?	คืนค่าสถานะผลการรับไฟล์ (ปกติคืนค่าเป็น 0)
\$\$	ค่า Process id (PID) ของไฟล์ที่รับข้อมูลนั้น

ตัวอย่างการรับค่า argument

```
% set tim bill ann fred  
      $1  $2    $3  $4  
% echo $*  
tim bill ann fred  
% echo $#  
4  
% echo $1  
tim  
% echo $3 $4  
ann fred
```

คำสั่ง set ใช้
กำหนดตัวค่าตัว
แปรใน CLI ได้

bash control structures

- if-then-else
- case
- loops
 - for
 - while
 - until
 - select

The simple if statement

```
if [ condition ]; then  
    statements  
fi
```

- executes the statements only if condition is true

The if-then-else statement

```
if [ condition ]; then
    statements-1
else
    statements-2
fi
```

- executes statements-1 if condition is true
- executes statements-2 if condition is false

The if..statement

```
if [ condition ]; then
    statements
elif [ condition ]; then
    statement
else
    statements
fi
```

CSCI 330 - The Unix System

- The word **elif** stands for “else if”
- It is part of the if statement and cannot be used by itself

Relational Operators

Meaning	Numeric	String
Greater than	-gt	
Greater than or equal	-ge	
Less than	-lt	
Less than or equal	-le	
Equal	-eg	= or ==
Not equal	-ne	!=
str1 is less than str2		str1 < str2
str1 is greater str2		str1 > str2
String length is greater than zero		-n str
String length is zero		-z str

Compound logical expressions

!

not

&&

and

||

or



and, or

must be enclosed within

[[

]]

```
[[ $a -gt $b ]] && [[ $b -lt $c ]]
```

Example: Using the ! Operator

```
#!/bin/bash

read -p "Enter years of work: " Years
if [ ! "$Years" -lt 25 ] ; then
    echo "You can retire now."
else
    echo "You need 25+ years to retire"
fi
```

Example: Using the && Operator

```
#!/bin/bash

Bonus=500

read -p "Enter Status: " Status
read -p "Enter Shift: " Shift

if [[ "$Status" = "H" && "$Shift" = 3 ]]
then
    echo "shift $Shift gets \$Bonus bonus"
else
    echo "only hourly workers in"
    echo "shift 3 get a bonus"
fi
```

Example: Using the || Operator

```
#!/bin/bash

read -p "Enter calls handled:" CHandle
read -p "Enter calls closed: " CClose
if [[ "$CHandle" -gt 150 ]] || [[ "$CClose" -gt 50 ]]
then
    echo "You are entitled to a bonus"
else
    echo "You get a bonus if the calls"
    echo "handled exceeds 150 or"
    echo "calls closed exceeds 50"
fi
```

File Testing

Meaning

-d file	True if 'file' is a directory
-f file	True if 'file' is an ord. file
-r file	True if 'file' is readable
-w file	True if 'file' is writable
-x file	True if 'file' is executable
-s file	True if length of 'file' is nonzero

Example: File Testing

```
#!/bin/bash
echo "Enter a filename: "
read filename
if [ ! -r "$filename" ]
then
    echo "File is not read-able"
    exit 1
fi
```

Example: File Testing

```
#!/bin/bash

if [ $# -lt 1 ]; then
    echo "Usage: filetest filename"
    exit 1
fi
if [[ ! -f "$1" || ! -r "$1" || ! -w "$1" ]]
then
    echo "File $1 is not accessible"
    exit 1
fi
```

Example: if... Statement

```
# DOUBLE SQUARE BRACKETS
read -p "Do you want to continue?" reply
if [[ $reply = "y" ]]; then
    echo "You entered " $reply
fi

# SINGLE SQUARE BRACKETS
read -p "Do you want to continue?" reply
if [ $reply = "y" ]; then
    echo "You entered " $reply
fi

# "TEST" COMMAND
read -p "Do you want to continue?" reply
if test $reply = "y"; then
    echo "You entered " $reply
fi
```

Example: if..elif.. Statement

```
#!/bin/bash

read -p "Enter Income Amount: " Income
read -p "Enter Expenses Amount: " Expense

let Net=$Income-$Expense

if [ "$Net" -eq "0" ]; then
    echo "Income and Expenses are equal - breakeven."
elif [ "$Net" -gt "0" ]; then
    echo "Profit of: " $Net
else
    echo "Loss of: " $Net
fi
```

The case Statement

- use the case statement for a decision that is based on multiple choices

Syntax:

```
case word in
    pattern1) command-list1
    ;;
    pattern2) command-list2
    ;;
    patternN) command-listN
    ;;
esac
```

case pattern

- checked against word for match
- may also contain:
 - *
 - ?
 - [...]
- multiple patterns can be listed via: |

Example 1: The case Statement

```
#!/bin/bash

echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter q to quit"

read -p "Enter your choice: " reply

case $reply in
    Y|YES) echo "Displaying all (really...) files"
            ls -a ;;
    N|NO)   echo "Display all non-hidden files..."
            ls ;;
    Q)      exit 0 ;;
    *)      echo "Invalid choice!"; exit 1 ;;
esac
```

Example 2: The case Statement

```
#!/bin/bash
ChildRate=3
AdultRate=10
SeniorRate=7
read -p "Enter your age: " age
case $age in
    [1-9] | [1][0-2])    # child, if age 12 and younger
        echo "your rate is" '$'"$ChildRate.00" ;;
    # adult, if age is between 13 and 59 inclusive
    [1][3-9] | [2-5][0-9])
        echo "your rate is" '$'"$AdultRate.00" ;;
    [6-9][0-9])          # senior, if age is 60+
        echo "your rate is" '$'"$SeniorRate.00" ;;
esac
```

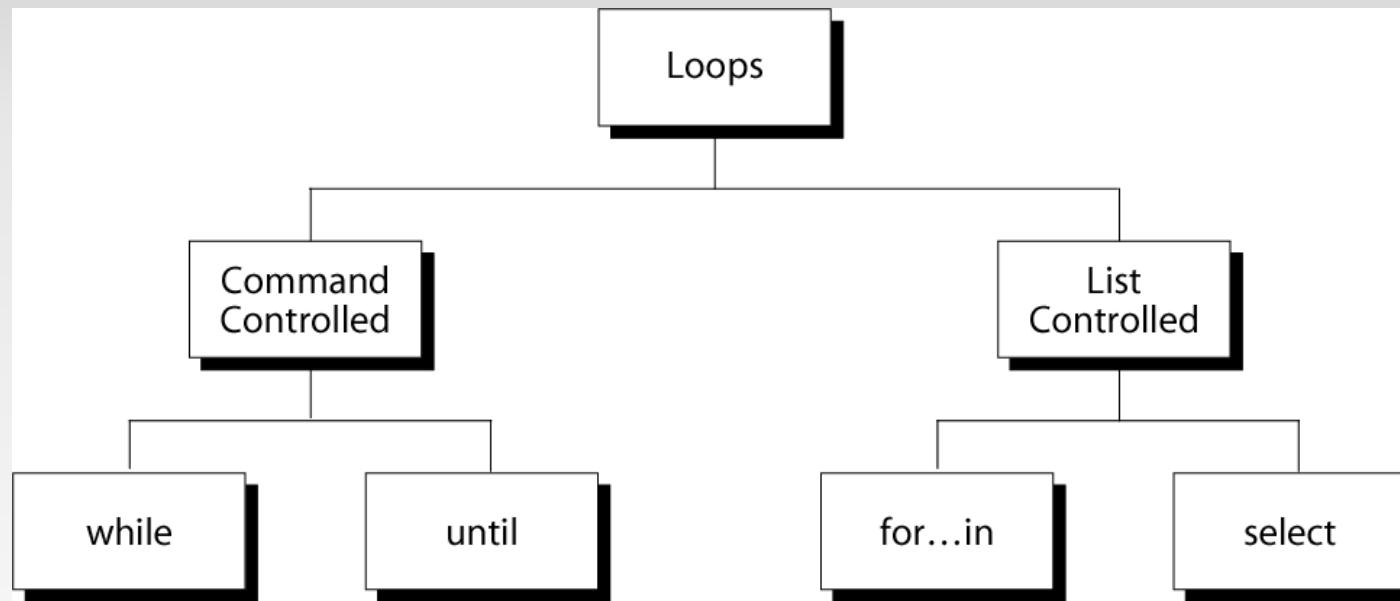
Bash programming: so far

- Data structure
 - Variables
 - Numeric variables
 - Arrays
- User input
- Control structures
 - if-then-else
 - case

Bash programming: still to come

- Control structures
 - Repetition
 - do-while, repeat-until
 - for
 - select
- Functions
- Trapping signals

Repetition Constructs



The while Loop

- Purpose:

To execute commands in “command-list”
as long as “expression” evaluates to true

Syntax:

while [expression]

do

command-list

done

Example: Using the while Loop

```
#!/bin/bash  
  
COUNTER=0  
  
while [ $COUNTER -lt 10 ]  
do  
    echo The counter is $COUNTER  
    let COUNTER=$COUNTER+1  
  
done
```

Example: Using the while Loop

```
#!/bin/bash

Cont="Y"

while [ $Cont = "Y" ] ; do

    ps -A

    read -p "want to continue? (Y/N)" reply

    Cont=`echo $reply | tr [:lower:] [:upper:]`

done

echo "done"
```

Example: Using the while Loop

```
#!/bin/bash

# copies files from home- into the webserver- directory
# A new directory is created every hour

PICSDIR=/home/carol/pics
WEBDIR=/var/www/carol/webcam

while true; do

    DATE=`date +%Y%m%d`
    HOUR=`date +%H`
    mkdir $WEBDIR/"$DATE"
    while [ $HOUR -ne "00" ]; do
        DESTDIR=$WEBDIR/"$DATE"/"$HOUR"
        mkdir "$DESTDIR"
        mv $PICSDIR/*.jpg "$DESTDIR"/
        sleep 3600
        HOUR=`date +%H`
    done
done
```

The until Loop

- Purpose:

To execute commands in “command-list”
as long as “expression” evaluates to false

Syntax:

until [expression]

do

command-list

done

Example: Using the until Loop

```
#!/bin/bash

COUNTER=20

until [ $COUNTER -lt 10 ]

do

    echo $COUNTER

    let COUNTER-=1

done
```

Example: Using the until Loop

```
#!/bin/bash

Stop="N"

until [ $Stop = "Y" ]; do

    ps -A

    read -p "want to stop? (Y/N)" reply

    Stop=`echo $reply | tr [:lower:] [:upper:]`

done

echo "done"
```

The for Loop

- Purpose:

To execute commands as many times as
the number of words in the “argument-list”

Syntax:

for variable in argument-list

do

commands

done

Example 1: The for Loop

```
#!/bin/bash

for i in 7 9 2 3 4 5
do
    echo $i
done
```

Example 2: Using the for Loop

```
#!/bin/bash
# compute the average weekly temperature

for num in 1 2 3 4 5 6 7
do
    read -p "Enter temp for day $num: "
    Temp
    let TempTotal=$TempTotal+$Temp
done

let AvgTemp=$TempTotal/7
echo "Average temperature: " $AvgTemp
```

looping over arguments

- simplest form will iterate over all command line arguments:

```
#! /bin/bash
for parm
do
    echo $parm
done
```

Select command

- Constructs simple menu from word list
- Allows user to enter a number instead of a word
- User enters sequence number corresponding to the word

Syntax:

```
select WORD in LIST  
do  
    RESPECTIVE-COMMANDS  
done
```

- Loops until end of input, i.e. ^d (or ^c)

Select example

```
#! /bin/bash
select var in alpha beta gamma
do
    echo $var
done
```

- Prints:

```
1) alpha
2) beta
3) gamma
#? 2
beta
#? 4
#? 1
alpha
```

Select detail

- PS3 is select sub-prompt
- \$REPLY is user input (the number)

```
#!/bin/bash
PS3="select entry or ^D: "
select var in alpha beta
do
    echo "$REPLY = $var"
done
```

Output:
select ...
1) alpha
2) beta
? 2
2 = beta
? 1
1 = alpha

Select example

```
#!/bin/bash
echo "script to make files private"
echo "Select file to protect:"

select FILENAME in *
do
    echo "You picked $FILENAME
    ($REPLY)"
    chmod go-rwx "$FILENAME"
    echo "it is now private"
done
```

break and continue

- Interrupt for, while or until loop
- The break statement
 - transfer control to the statement AFTER the done statement
 - terminate execution of the loop
- The continue statement
 - transfer control to the statement TO the done statement
 - skip the test statements for the current iteration
 - continues execution of the loop

The break command

```
while [ condition ]
```

```
do
```

```
    cmd-1
```

```
    break
```

```
    cmd-n
```

```
done
```

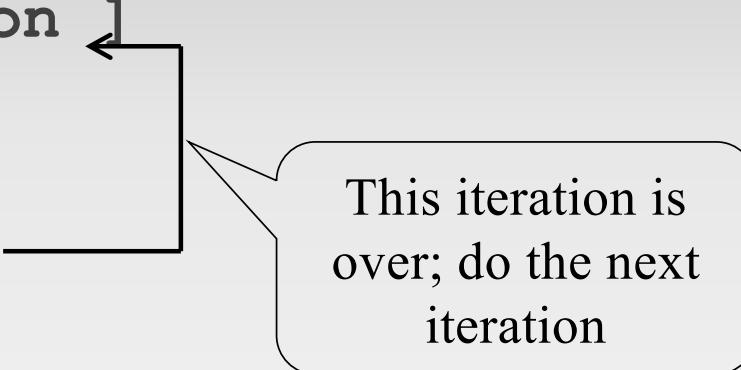
```
echo "done"
```



This iteration is over
and there are no more
iterations

The continue command

```
while [ condition ]  
do  
    cmd-1  
    continue  
    cmd-n  
  
done  
echo "done"
```



This iteration is over; do the next iteration

Example:

```
for index in 1 2 3 4 5 6 7 8 9 10
do
    if [ $index -le 3 ]; then
        echo "continue"
        continue
    fi
    echo $index
    if [ $index -ge 8 ]; then
        echo "break"
        break
    fi
done
```

Bash shell programming

- Sequence
- Decision:
 - if-then-else
 - case
- Repetition
 - do-while, repeat-until
 - for
 - select

DONE !

- Functions
- Traps

still to come

Shell Functions

- A shell function is similar to a shell script
 - stores a series of commands for execution later
 - shell stores functions in memory
 - shell executes a shell function in the same shell that called it
- Where to define
 - In .profile
 - In your script
 - Or on the command line
- Remove a function
 - Use unset built-in

Shell Functions

- must be defined before they can be referenced
- usually placed at the beginning of the script

Syntax:

```
function-name () {  
    statements  
}
```

Example: function

```
#!/bin/bash

funky () {
    # This is a simple function
    echo "This is a funky function."
    echo "Now exiting funky function."
}

# declaration must precede call:

funky
```

Example: function

```
#!/bin/bash

fun () { # A somewhat more complex function.

    JUST_A_SECOND=1

    let i=0

    REPEATS=30

    echo "And now the fun really begins."

    while [ $i -lt $REPEATS ]

        do

            echo "-----FUNCTIONS are fun----->"

            sleep $JUST_A_SECOND

            let i+=1

        done

    }

fun
```

Function parameters

- Need not be declared
- Arguments provided via function call are accessible inside function as \$1, \$2, \$3, ...

\$# reflects number of parameters

\$0 still contains name of script
(not name of function)

Example: function with parameter

```
#! /bin/sh
testfile() {
    if [ $# -gt 0 ]; then
        if [[ -f $1 && -r $1 ]]; then
            echo $1 is a readable file
        else
            echo $1 is not a readable file
        fi
    fi
}

testfile .
testfile funtest
```

Example: function with parameters

```
#! /bin/bash
checkfile() {
    for file
    do
        if [ -f "$file" ]; then
            echo "$file is a file"
        else
            if [ -d "$file" ]; then
                echo "$file is a directory"
            fi
        fi
    done
}
checkfile . funtest
```

Local Variables in Functions

- Variables defined within functions are global,
i.e. their values are known throughout the entire shell program
- keyword “local” inside a function definition makes referenced variables “local” to that function

Example: function

```
#! /bin/bash

global="pretty good variable"

foo () {
    local inside="not so good variable"
    echo $global
    echo $inside
    global="better variable"
}

echo $global
foo
echo $global
echo $inside
```

Handling signals

- Unix allows you to send a signal to any process
 - -1 = hangup `kill -HUP 1234`
 - -2 = interrupt with ^C `kill -2 1235`
 - no argument = terminate `kill 1235`
 - -9 = kill `kill -9 1236`
 - -9 cannot be blocked
 - list your processes with
 `ps -u userid`

Signals on Linux

```
% kill -l
      1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL
      5) SIGTRAP     6) SIGABRT     7) SIGBUS       8) SIGFPE
      9) SIGKILL    10) SIGUSR1    11) SIGSEGV    12) SIGUSR2
     13) SIGPIPE    14) SIGALRM    15) SIGTERM    16) SIGSTKFLT
     17) SIGCHLD   18) SIGCONT   19) SIGSTOP   20) SIGTSTP
     21) SIGTTIN   22) SIGTTOU   23) SIGURG    24) SIGXCPU
     25) SIGXFSZ   26) SIGVTALRM 27) SIGPROF    28) SIGWINCH
     29) SIGIO     30) SIGPWR     31) SIGSYS    34) SIGRTMIN
     35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4
     39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
     43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
     47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
     51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
     55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
     59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
     63) SIGRTMAX-1  64) SIGRTMAX
```

- ^C is 2 - SIGINT

Handling signals

- Default action for most signals is to end process
 - term: signal handler
- Bash allows to install custom signal handler

Syntax:

```
trap 'handler commands' signals
```

Example:

```
trap 'echo do not hangup' 1 2
```

Example: trap hangup

```
#! /bin/bash
# kill -1 won't kill this process
# kill -2 will

trap 'echo dont hang up' 1

while true
do
    echo "try to hang up"
    sleep 1
done
```

Example: trap multiple signals

```
#! /bin/sh

# plain kill or kill -9 will kill
# this

trap 'echo 1' 1
trap 'echo 2' 2

while true; do
    echo -n .
    sleep 1
done
```

Example: removing temp files

```
#! /bin/bash
trap 'cleanup; exit' 2

cleanup () {
    /bin/rm -f /tmp/tempfile.$$.?
}

for i in 1 2 3 4 5 6 7 8
do
    echo "$i.iteration"
    touch /tmp/tempfile.$$.${i}
    sleep 1
done
cleanup
```

Restoring default handlers

- `trap` without a command list will remove a signal handler
- Use this to run a signal handler once only

```
#! /bin/sh
trap 'justonce' 2
justonce() {
    echo "not yet"
    trap 2          # now reset it
}

while true; do
    echo -n "."
    sleep 1
done
```

Debug Shell Programs

- Debugging is troubleshooting errors that may occur during the execution of a program/script
- The following two commands can help you debug a bash shell script:
 - echo
 - use explicit output statements to trace execution
 - set

Debugging using “set”

- The “set” command is a shell built-in command
- has options to allow flow of execution
 - v option prints each line as it is read
 - x option displays the command and its arguments
 - n checks for syntax errors
- options can turned on or off
 - To turn on the option: set -xv
 - To turn off the options: set +xv
- Options can also be set via she-bang line

```
# ! /bin/bash -xv
```

Summary: Bash shell programming

- Sequence
- Decision:
 - if-then-else
 - case
- Repetition
 - do-while, repeat-until
 - for
 - select
- Functions
- Traps

DONE !

สรุป

- พื้นฐานระบบปฏิบัติ Multi-user , Multi—task
- อินุกช์ (Linux) คล้าย Unix
- คำสั่งพื้นฐาน
- Shell script