Unix tips: good place to place scripts

When you type a command name, bash searches for it in the directories specified in PATH

- Commands are searched in the order specified in PATH.

Example:

$ echo $PATH
/home/me/bin:/usr/local/sbin:/usr/local/bin:
/usr/sbin:/usr/bin:/sbin:/bin:

- Use the PATH variable to add directories to your search path.

Adding a directory

$ PATH=~/bin:"$PATH"
Unix tips: making it permanent

You can make changes permanently by adding expressions to one of these files:

- `/etc/profile`: global, affects all users (root access)
- `~/.bash_profile`: user’s personal file (aka `~/.bashrc`)

After you make changes execute the following for changes to take effect.

```
$ source .bash_profile
```
Unix tips: env

Where is awk?

- In the CSUG machines it is at /bin/awk
- In Mac OS X it is at /usr/bin/awk
- I installed it at /usr/local/bin/awk

What should I use in my hash bang line, if I want portability?
Unix tips: env

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

- `env` tends to be consistently located at `/usr/bin/env`.
- `env` searches for the first program executable in `$PATH`.

Now our script will run on every system, regardless of the location of the program.
Alternatives to man

- help: help for shell built-ins.
- command --help: help for command
- apropos "search term": show appropriate commands
- whatis command: really short description of command
- info command: similar to man, but with hyperlinks
Unix tips: spell checker

aspell -c file
Unix tips: useful tools

- printf format arguments
- basename path
- dirname file

Example:

```bash
$ what="unix"
$ printf "I love %s\n" $what
I love unix
```

Example:

```bash
[bash-3.2: /home/me]$ basename 'pdw'
me
```

Example:

```bash
$ dirname /usr/bin/awk
/usr/bin
```

Instructor: Bruno Abrahao
Unix tips: useful tools

- nl files

**Example:**

```bash
$ cat file
foo
bar
cookie
$ nl file
1 foo
1 bar
1 cookie
```
Unix tips: disk space

- `df -h`

**Example:**

```
$ df -h
Filesystem     Size  Used  Avail  Capacity
/dev/disk1     465Gi  401Gi  64Gi   87%
```

- `du -sm files`

**Example:**

```
$ du -sm /home/abrahao/*
10 Desktop
12360 Documents
21409 Movies
```
Unix tips: format

**Long Command**

```bash
$ find . \( -name \*.log -o -name \*.toc -o -name "*.synctex.gz" \) -exec rm {} \;
```

**Adding breaks**

```bash
$ find . \\
\( \( \) \( \\
  -name \*.log \\
  -o \\
  -name \*.toc -o \\
  -o \\
  -name *.synctex.gz \\
\) \) \\
-exec rm {} \;
```
Split screen horizontally
:sp

Split screen vertically
:vsp

Move between split regions
<ctrl-w w>
`${v}`

- separates variable `v` from other text.

**Example:**

```
$ v="unix"
$ echo $v
unix
$ echo "$v-text"

$ echo "${v}-text"
unix-text
```
Arrays in bash can be declared in a number of different ways.

- `declare -a name`
- `name[subscript]=value`
- `name=(value1 value2 ...)`
- `name=(\[0\]=value1 \[1\]=value2 ...)`

Only one dimensional arrays in Bash! Indexes from 0.
Examples

courses=(cs2042 cs2043 cs2044)
courses=([0]=cs2042 [87]=cs2043 [100]=cs2044)
courses[100]=cs2044

The last statement produces an array with a single element, namely "cs2044", indexed at 100, not 100 elements.
- echo $name  # name[0]
- echo ${name[87]}  # the item indexed at 87
- echo $name[87]  # the item indexed at 0 followed by the string "[87]"
Operations

- `echo ${name[*]}` # all items in a single string
- `echo ${name[@]}` # all items, each in a separate string
presidents=("Barack Obama", "George Bush")

Example:

$ for i in "${pres[*]}"; do echo $i; done
Barack
Obama
George
Bush

Example:

$ for i in "${pres[@]}"; do echo $i; done
Barack Obama
George Bush
- echo `${#name[@]}`  # length of array
- echo `${#name[100]}`  # length of string at 100
- echo `${!name[*]}`  # all indices in a single string
- echo `${!name[@]}`  # all indices, each is a separate string
- unset name  # deletes array
- unset `${name[2]}`  # deletes item indexed at 2
- name=  # deletes item indexed at 0
Let’s see how we can sort an array
Shell Functions

function name {
    commands
    return
}

name () {
    commands
    return
}

- Should be defined before they are called
- return is optional
Shell variables are global

Use the statement `local` to create a local variable that is deleted after the function returns

**Example:**

```
$ foo=0
$ myfunc () { local foo; foo=1; echo $foo; }
$ myfunc
1
$ echo $foo
0
```
Works exactly the same way we pass arguments to scripts

Example:

$ sum () { echo $((1+$2)); }
$ sum 4 5
9
Python!