

# CSCI 3411 – OS

## Lab 2

MORE COMMANDS, SHELLS, UNIX SYSTEM CALLS

# C is the Language of Unix!

- ▶ <http://git.savannah.gnu.org/cgit/coreutils.git/tree/src/>
  - ▶ <http://git.savannah.gnu.org/cgit/coreutils.git/tree/src/ls.c>
  - ▶ <http://git.savannah.gnu.org/cgit/coreutils.git/tree/src/mkdir.c>
  - ▶ <http://git.savannah.gnu.org/cgit/coreutils.git/tree/src/rm.c>

# More Useful Unix Commands

- ▶ Tab – autocomplete.
- ▶ Ctrl + C – force a program to terminate.
- ▶ Ctrl + D – insert end-of-file (EOF) character.
- ▶ & - run program in background.

# More Useful Unix Commands

- ▶ Manual: `man command` (also works with C functions and ASCII)
- ▶ Viewing file contents:
  - ▶ `cat` – prints file contents to terminal
  - ▶ `more` – displays file contents in paged format (prints to terminal)
  - ▶ `less` – displays file contents in paged format (separate “window”)
- ▶ Moving and Copying files:
  - ▶ `mv oldname newname`
  - ▶ `cp oldname newname`

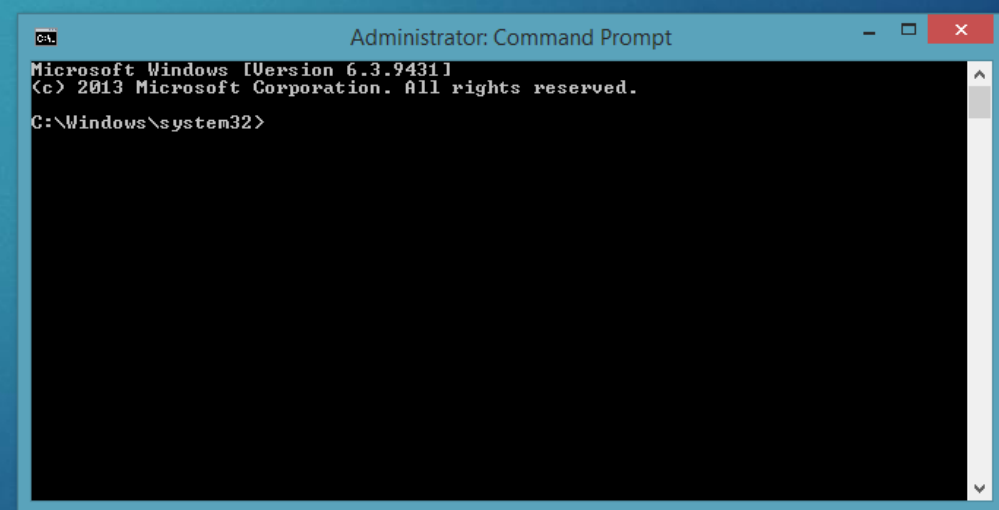
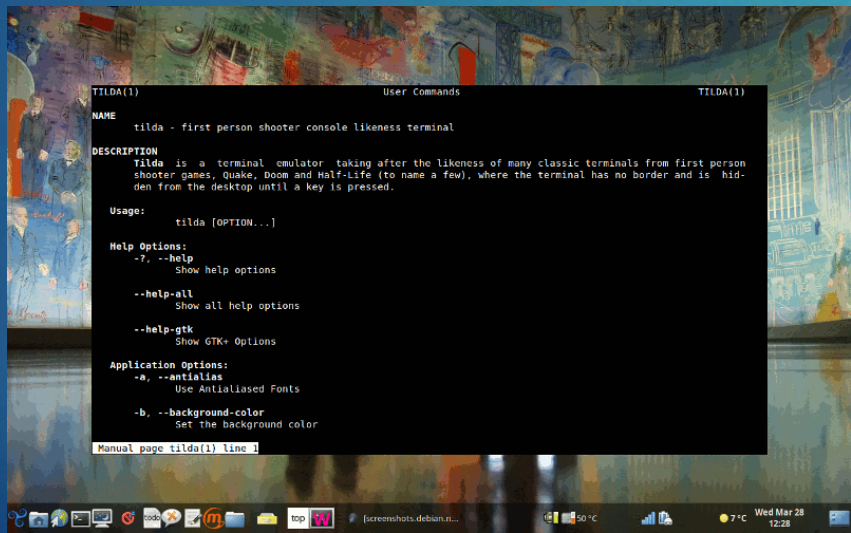
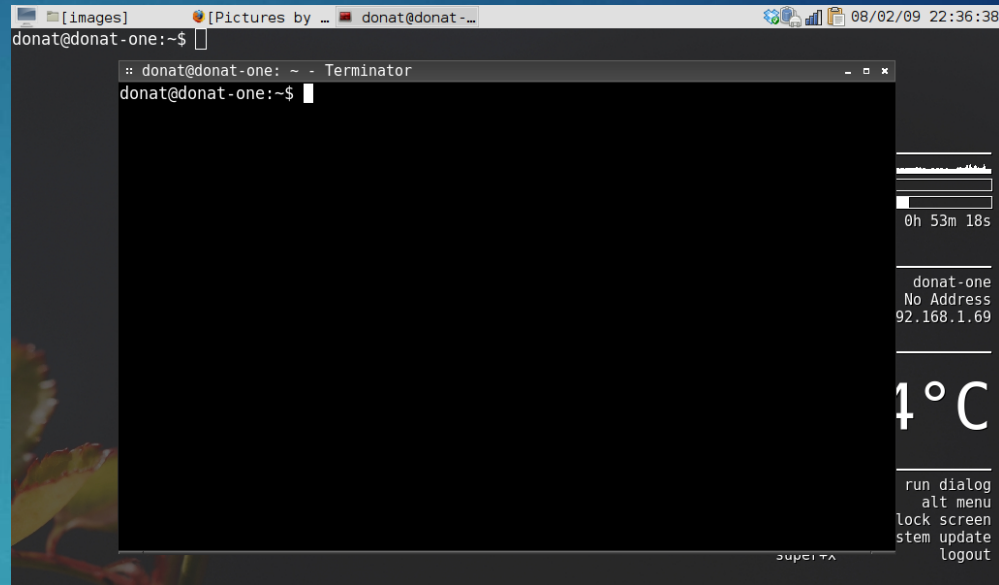
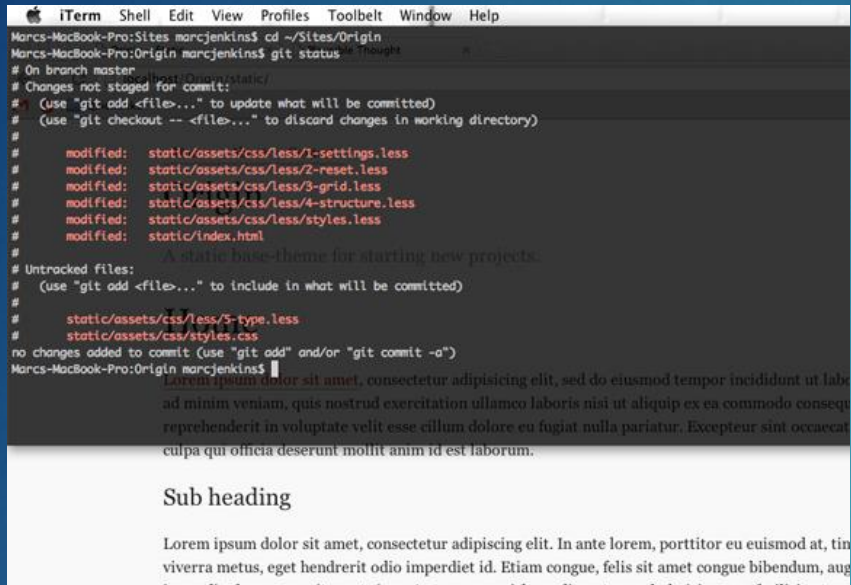
# I/O Redirection, Pipes, and Grep

- ▶ *program < filename*
  - ▶ Use the contents of filename as input into program.
- ▶ *program > filename*
  - ▶ Send the output of program to filename (overwrite)
- ▶ *program >> filename*
  - ▶ Send the output of a program to filename (append)
- ▶ *program1 | program2*
  - ▶ Use the output of program1 as input for program2
- ▶ *grep pattern filename*
  - ▶ *program | grep pattern*
  - ▶ *-c x* (Show X lines of context around matched patterns)



# Terminal vs. Shell?

BASH?



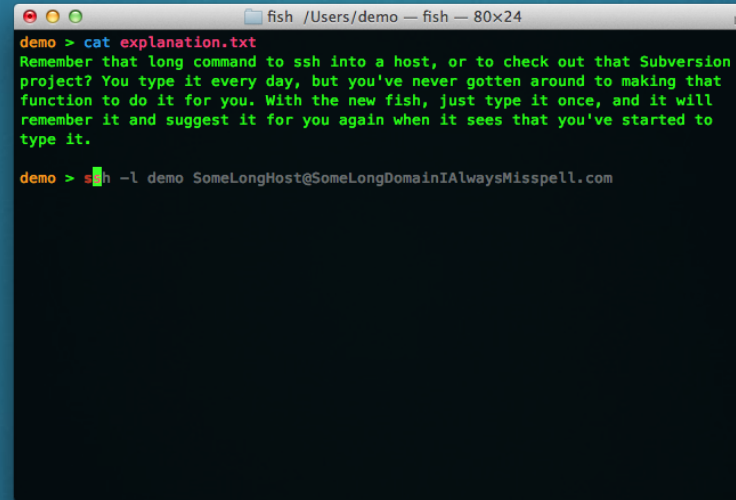
# Bash

- ▶ Command history (`history`)
  - ▶ `!!` – run last command
  - ▶ `!x` – run command `x`
  - ▶ `!name` – run last history item that began with `name`
  - ▶ `Ctrl + R` – reverse history search
- ▶ `alias name="command"`
  - ▶ `alias rm="rm -i"`



# Other Shells

- ▶ Bourne shell (sh)
- ▶ bash
- ▶ csh
- ▶ fish
- ▶ zsh

A screenshot of a terminal window titled "fish /Users/demo — fish — 80x24". The terminal shows a user named "demo" running the command "cat explanation.txt". The output of the command is a paragraph of text: "Remember that long command to ssh into a host, or to check out that Subversion project? You type it every day, but you've never gotten around to making that function to do it for you. With the new fish, just type it once, and it will remember it and suggest it for you again when it sees that you've started to type it." Below this, the user runs "ssh -l demo SomeLongHost@SomeLongDomainIAAlwaysMisspell.com".

```
demo > cat explanation.txt
Remember that long command to ssh into a host, or to check out that Subversion
project? You type it every day, but you've never gotten around to making that
function to do it for you. With the new fish, just type it once, and it will
remember it and suggest it for you again when it sees that you've started to
type it.

demo > ssh -l demo SomeLongHost@SomeLongDomainIAAlwaysMisspell.com
```

fish

A screenshot of a terminal window showing a sequence of commands and their outputs. The user starts in a directory "~/testproject" and runs "cd testproject". Then they run "gco detached-head-state -q" and "touch dirty-working-directory". Next, they run "ssh milly", which shows a login for "agnoster@milly" on "Ubuntu 11.04". After "Connection to milly.agnoster.net closed.", they run "sudo -s", enter a password, and become root. As root, they run "top &", which shows process 34523. Then they run "rm no-such-file", which results in an error. Finally, they run "kill %%", which terminates the top process. The prompt returns to the user.

```
~> cd testproject
~/testproject > gco detached-head-state -q
~/testproject > touch dirty-working-directory
~/testproject > ssh milly
Welcome to Ubuntu 11.04 (GNU/Linux 2.6.18-308.8.2.el5.028stab101.1 x86_64)
Last login: Wed Sep 26 03:42:49 2012 from 71-215-222-90.mpls.qwest.net
agnoster@milly ~>
Connection to milly.agnoster.net closed.
~> sudo -s
Password:
~# root@Arya ~# top &
[1] 34523
[1] + 34523 suspended (tty output) top
~# root@Arya ~# rm no-such-file
rm: no-such-file: No such file or directory
~# root@Arya ~# kill %%
[1] + 34523 terminated top
~# root@Arya ~#
~>
```

zsh



Next Homework

# Fork

- ▶ `#include <unistd.h>`
  - ▶ Returns 0 for the child.
  - ▶ Returns the child's process ID for the parent.
  - ▶ Returns -1 if failed.
- 
- ▶ Child receives copy of data at the point the fork occurred.
  - ▶ Child begins execution at the same point.

# Exec

- ▶ `#include <unistd.h>`
- ▶ A family of functions (`execl`, `execv`, `execvp`, etc.) to execute a program, thus replacing the current process with the new one.
- ▶ `execv(filepath, argument List)`
  - ▶ Formally: `execv(const char *path, char *const argv[])`
  - ▶ Example – `execEcho.c`
- ▶ `execvp(filepath, argument List)`
  - ▶ Formally: `execvp(const char *path, char *const argv[])`
  - ▶ Differs from `execv` because it will search for the specified file.
  - ▶ Example – `execLs.c`

# Wait

- ▶ `#include <sys/wait.h>`
- ▶ `pid_t wait(int *status)`
  - ▶ Causes the parent to wait until their child finishes execution.
  - ▶ Returns the process ID (`pid_t`) of the terminated child process.
  - ▶ Exit code will be placed in *status* (can be NULL)
- ▶ `pid_t waitpid(pid_t pid, int *status, int options)`
  - ▶ Parent waits until the child specified by *pid* finishes.
- ▶ Example



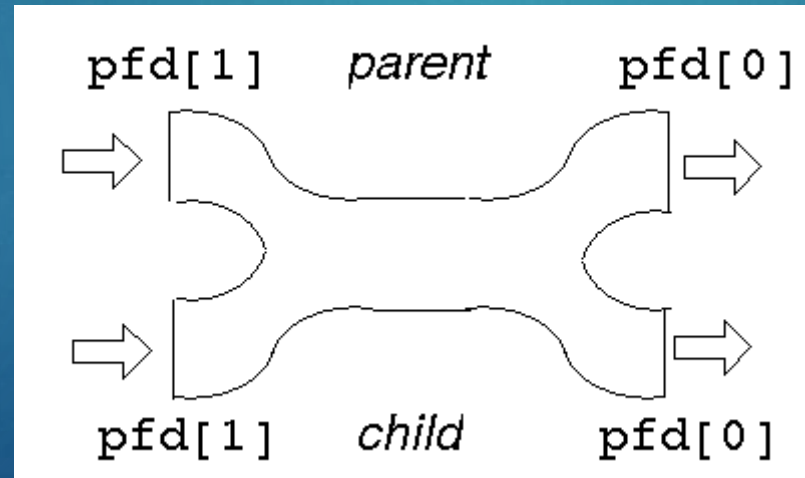
# Pipe

- ▶ `#include <unistd.h>`
- ▶ Creates two file descriptors for the two ends of a pipe (read-side and write-side).
  - ▶ One direction.
- ▶ Parent can write to one side and a child can read from the other (or vice versa).
- ▶ 

```
int myPipe[2];  
pipe(myPipe)    //That's it!
```
- ▶ `myPipe[0] = read-side, myPipe[1] = write-side`
  - ▶ *read(file descriptor, buffer, bytes to read)*
  - ▶ *write(file descriptor, data, bytes to write)*

# Pipe

- ▶ Example: pipeToSelf.c, pipeToChild.c
- ▶ Note for two pipes:
  - ▶ Parent does not need child's write side or the parent's read side.
  - ▶ Child does not need parent's write side or the child's read side.



# Ideas for Shell Prompts

- ▶ time.h:
  - ▶ Obtain the current time.
- ▶ unistd.h:
  - ▶ Get current directory.
  - ▶ Get user ID (or username).
  - ▶ Get hostname (computer name).
- ▶ Success / failure of last command (or child process).

Questions?