Version Control with Git and What is there in Project 1

PALLABI GHOSH (PALLABIG@ANDREW.CMU.EDU)

15-441 COMPUTER NETWORKS

RECITATION 1

What is version control?

- Revisit previous code versions
- Backup projects
- Work with others
- Find where things broke

Version Control Workflow

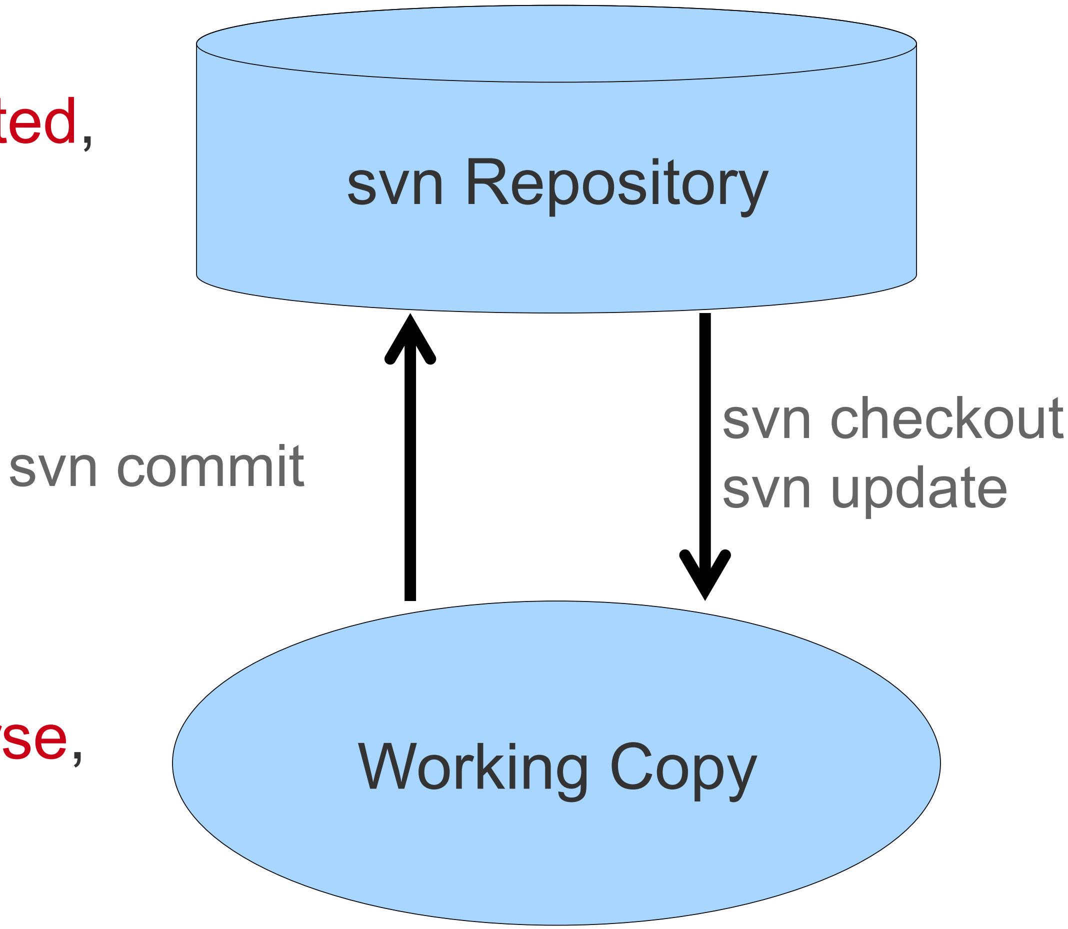
- Check for any remote updates
- Do your work
- Test your work
- Check differences, try to isolate changes
- Check for any remote updates
- Commit your work

Options

- Git
- Subversion (svn)
- Mercurial (hg)
- Bazaar (bzr)
- CVS
- ~~Dropbox~~
- Others…

svn

Usually remotely hosted, shared with a team.



svn Repository

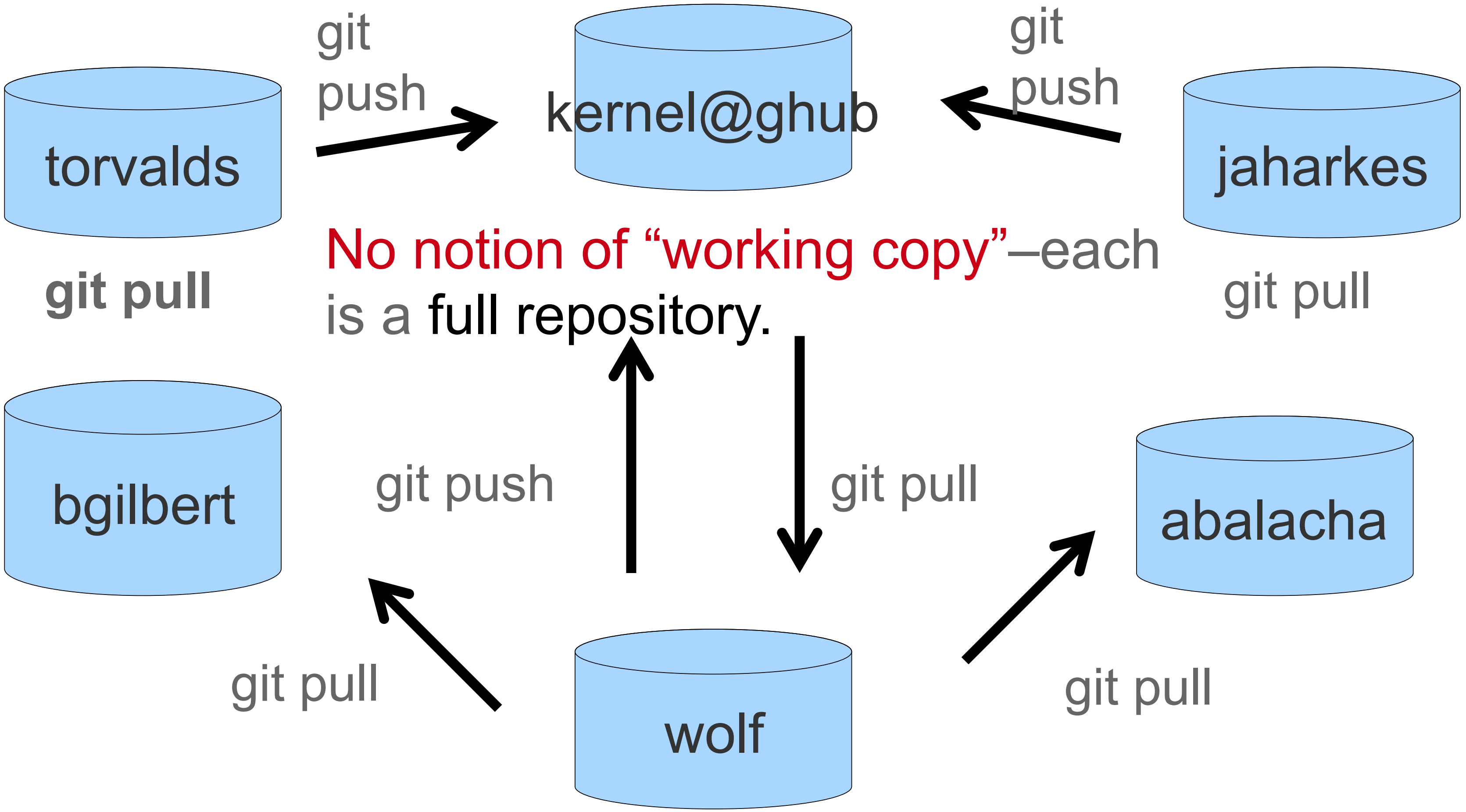
svn commit

svn checkout
svn update

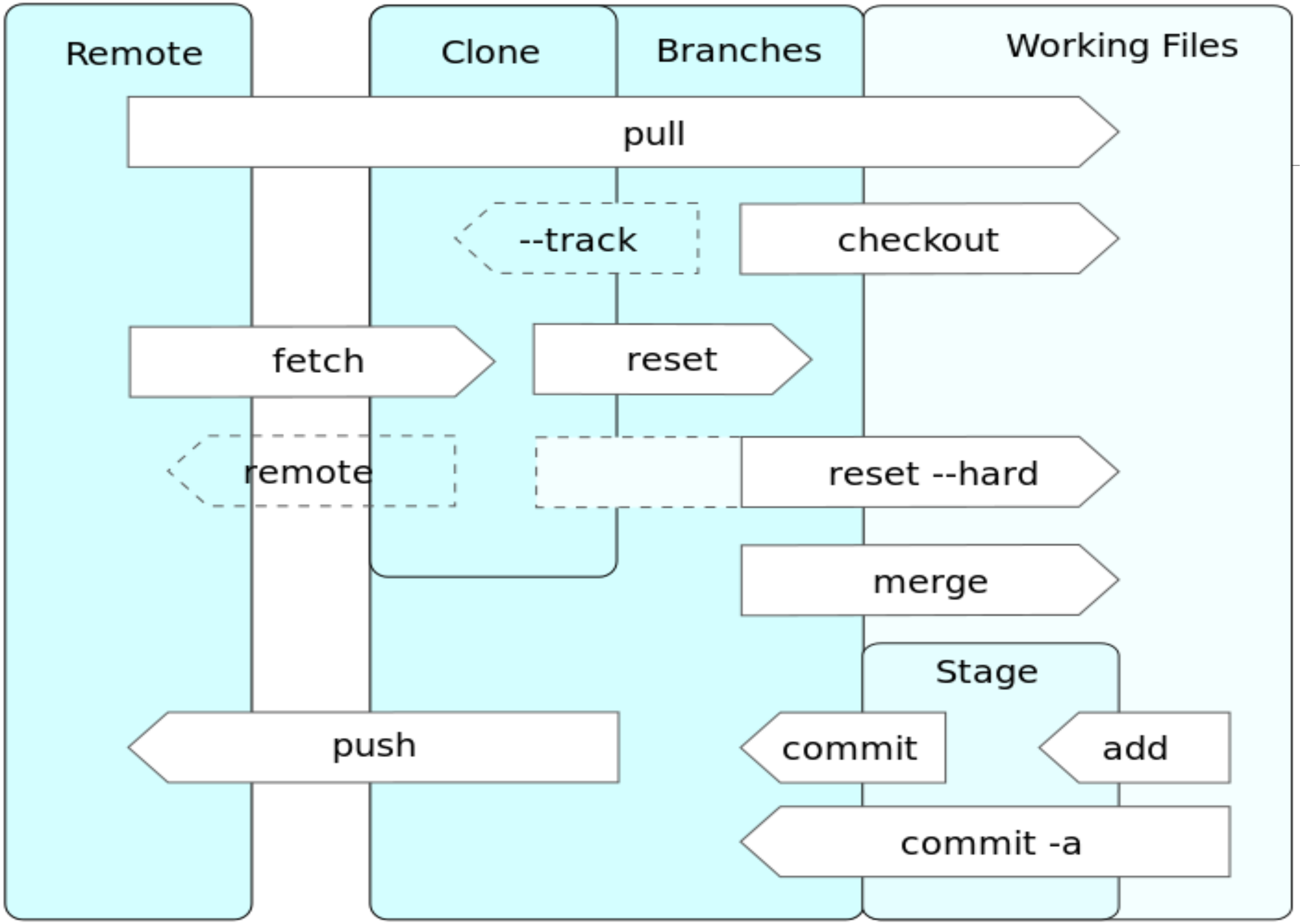
Working Copy

Your private universe, before commit.

git



No notion of “working copy”–each is a full repository.


Remote
Clone
Branches
Working Files
pull
--track
checkout
fetch
reset
remote
reset --hard
merge
Stage
push
commit
add
commit -a

Creating a Repository (repo)

Create locally

```
git init .
```

Create remote

```
git init –-bare
```

Clone local copy

```
git clone git://path/to/repo
```

--bare or not?

- No-bare
  - Creates a repository in your working directory
  - Don’t need to create multiple copies of your repo
  - Won’t help if you nuke the directory/disk
  - This is probably what you need if you’ll work in AFS
- --bare
  - Creates a “server copy” for hosting the project
  - Workflow more similar to svn (but still better)
  - Everyone pushes to shared bare repo (like svn)
  - You don’t work in this copy; must clone elsewhere
  - You want this to develop on your PC

Aside: network protocols

- Use different protocols to pull/push to repositories.
- If on the same computer:
  - git://path/to/repo
- If hosted on AFS
  - ssh+git://path/to/repo
- No ssh keys for AFS, sorry

Aside: Configure git

```
git config --global user.name “Pallabi Ghosh”
```

```
git config --global user.email “pallabig@Andrew.cmu.edu”
```

Clone

Pull a copy of the repo to develop on

```
git clone git://path/to/repo
```

```
git clone ssh+git://
unix.andrew.cmu.edu/afs/andrew/
course/15/441-641/ANDREWID/
ANDREWID-15-441-project-1.git
```

status

- Which files changed?
- Which files aren’t being watched?
- Which files are stashed for commit?

```
git status
```

Pull

- Get latest updates from remote copy

```
git pull
```

- If this fails, you probably need to commit any unsaved changes

Commit

- Merge your changes into the repository

```
git add foo.c …
git commit
```

Push

- Don’t push broken code!!

```
git push
```

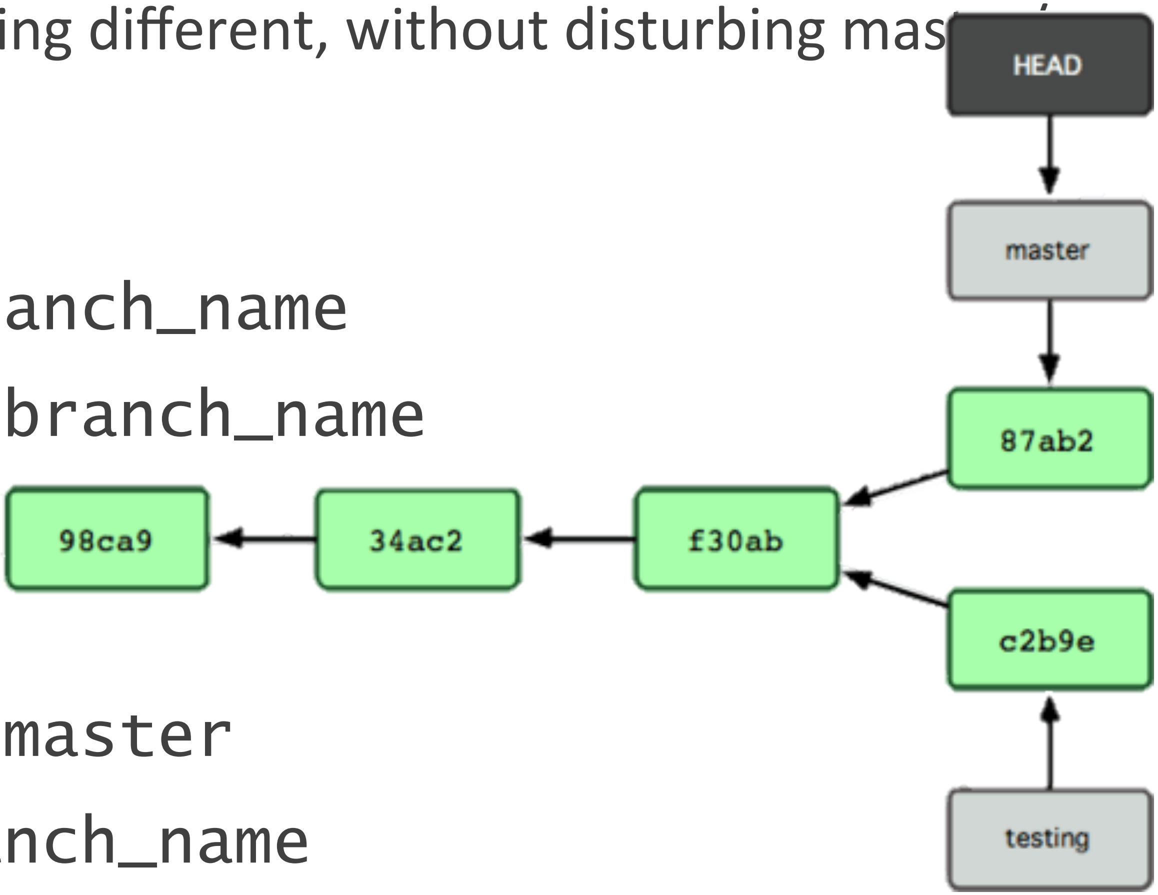
- If this fails, you probably need to pull first

Branch & Merge

- Work on something different, without disturbing mas trunk

```
git branch branch_name
git checkout branch_name
do stuff…

git checkout master
git merge branch_name
```



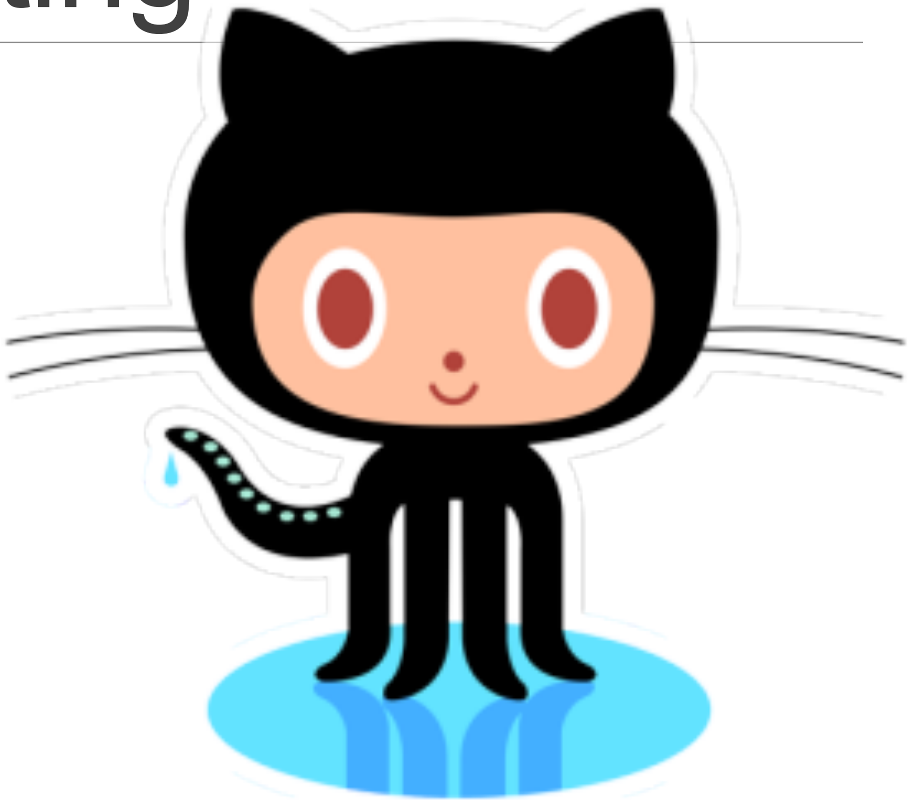
Tag

- Mark a revision as “final” or “ready”

```
git tag tag_name
git push --tags
```

Remote Hosting

- github.com
- bitbucket.org
- svnhub.com
- AFS
- Google code
- Sourceforge



Aside: AFS Permissions

- To make a bare repo in AFS that someone else can pull/push from:

1. Make a new directory in your home dir
2. fs sa . ANDREWID rlidwk
3. git init --bare

Good practices

- Small commits
- Useful messages
- Commit frequently
- Develop in branches
- Tag releasable versions

Small commits

- Only change one thing per commit
- When something breaks, easier to trace

Helpful commit messages

- Say what you changed
- Keep the first line short
- Make commits easy to find
- www.commitlogsfromlastnight.com

Commit Frequently

- Make changes, commit them
- When something breaks, go to the commit that broke it
- Only push when ready for others to get the changes
  - Don’t make your teammates hate you

Git questions?

Who took 15-213?

And made an HTTP proxy?

Project 1: HTTP déjà vu

- Blast from the past 15-213
- This time a real HTTP server with:
  - SSL
  - select() IO for concurrent connections
  - HTTP 1.1
  - CGI
- Big project, **start early**!

Checkpoint 1 – September 5

- Create a git repo named 15-441-project-1
- Code a select()-based echo server handling multiple clients at once (building on the supplied echo server)

Read the handout carefully – lots of great references

And once again – start early ☺

What do you want to build?

A webserver that can handle multiple concurrent connections!

What's the problem?

Blocking!

What's the solution?

Threading or select()

Threading approach

- Did in 15-213??
- Main server blocks on accept()
- Accept incoming connection
- Fork() child process for each connection
- Pain!
  - Need to manage a pool of threads
  - And what if tasks have to communicate?

World of select()

- Event driven programming!
- Single process that multiplexes all requests.
- Caveat
  - Programming is not so transparent!
  - Server no longer acts like it has only one client!

How to use select()?

- Give select a set of sockets/file descriptors.
- select() blocks till something happens.
  - Data coming in on some socket.
  - Able to write to a socket.
  - Exception at the socket.
- Once woken up, check for the event and service it the way the server would do.

select()

```
#include <sys/select.h>


int select (int  nfds, fd_set*  readfds,
                                       fd_set*  writefds, fd_set*
exceptfds,                   struct timeval *timeout);
```

fd_set Datastructure

- Remember, file descriptor is just an integer!
- Datastructure is basically a bit array!
- Helper macros:

```
FD_ZERO(fd_set* fdset);  /* initializes fdset to have 0s for all fds
  */
FD_SET(int fd, fd_set* fdset);  /* sets the bit for fd in fdset */
FD_CLR(int fd, fd_set* fdset);  /* clears the bit for fd in fdset */
FD_ISSET(int fd, fd_set* fdset); /* returns non-0 if fd is set else 0
  */
```

select() Parameters

- The FDs between 0 to nfds-1 are checked.
- Check for reading in readfds.
- Check for writing in writefds.
- Check for exception in exceptfds.
- These fd_sets can be NULL.
- timeout
  - NULL – blocking
  - else how long to wait for the required condition before returning to the caller.

Return value, Error states

- Success – number of ready descriptors.
  - readfds, writefds and exceptfds are modified

- Time expired – returns 0 (errno set to EINTR)

- Failure – returns -1
  - EBADF, EINTR, EINVAL , ENOMEM

Pseudo-code of Usage

- nfds = 0
- Initialize readfds, writefds, exceptfds using FD_ZERO
- Add the listener socket to readfds using FD_SET and update nfds
- For each active connection
- If connection has available read buffer, add fd to readfds (FD_SET)
- If connection has available write buffer, add to writefds (FD_SET)
- Add to exceptfds (FD_SET) – not really needed for this project.
- Update nfds to ensure that the fd falls in the range
- select_return = select(nfds, readfds, writefds, exceptfds, NULL)
- If select_return > 0
- Handle exceptions if any fd in exceptfds is set to 1 (FD_ISSET)
- Read data from connections for which fd in readfds is set to 1 (FD_ISSET)
- Write data from connections for which fd in writefds is set to 1 (FD_ISSET)
- If listener socket is set to read, accept and handle new connection.
- Else handle error states

Checkpoint 1 Docs

- Makefile - make sure nothing is hard coded specific to your user; should build a file which runs the echo server (name it lisod)
- All of your source code - all .c and .h files
- readme.txt - file containing a brief description of your current implementation of server
- tests.txt - file containing a brief description of your testing methods for server
- vulnerabilities.txt - identify at least one vulnerability in your current implementation

Peek into the future

- Checkpoint 2 – September 19
  - Implement HTTP 1.1 parser and persistent connections

- Checkpoint 3 – October 3
  - Implement HTTPS handshaking and persistent connections via TLS
  - Implement CGI server-side.

All questions?