CS356: Discussion #1

Development Environment

Marco Paolieri (paolieri@usc.edu)

USC University of Southern California

Contact Information

Marco Paolieri
PhD at the University of Florence, Italy (2015)
Postdoc at USC since 2016

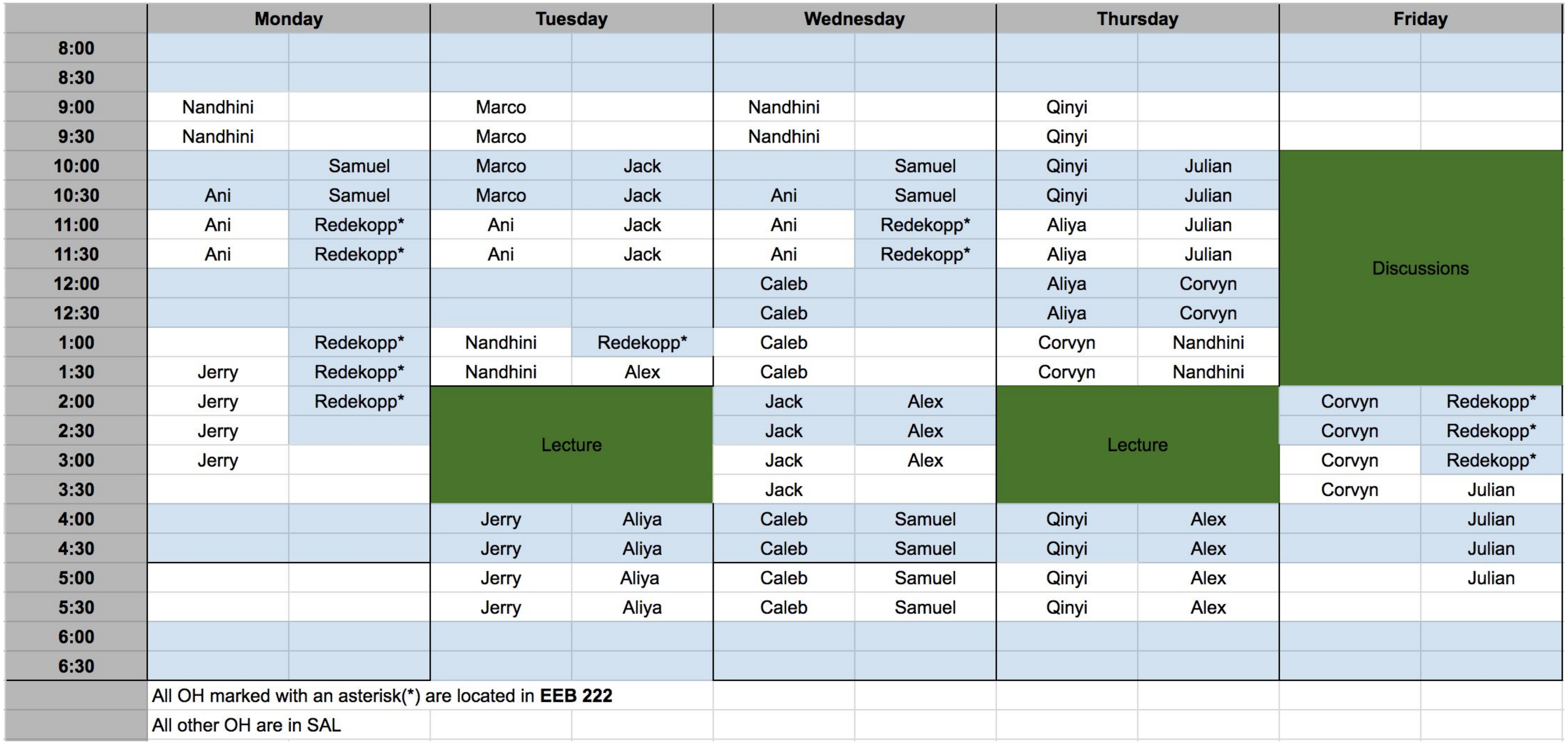
Email: **paolieri@usc.edu**
Office Hours: **Tuesday, 9:00–11:00am** (**SAL 200**)

Discussion sessions:
- Friday, 10:00–11:50am (KDC 235)
- Friday, 12:00–1:50pm (KDC 235)

Course Staff and Office Hours

Instructor: Mark Redekopp

11 TAs/CPs!



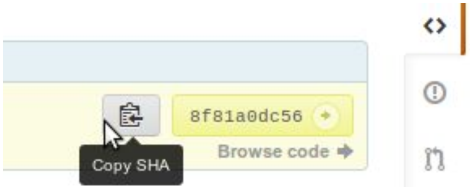
	Monday		Tuesday		Wednesday		Thursday		Friday	
8:00										
8:30										
9:00	Nandhini		Marco		Nandhini		Qinyi			
9:30	Nandhini		Marco		Nandhini		Qinyi			
10:00		Samuel	Marco	Jack		Samuel	Qinyi	Julian	Discussions	
10:30	Ani	Samuel	Marco	Jack	Ani	Samuel	Qinyi	Julian		
11:00	Ani	Redekopp*	Ani	Jack	Ani	Redekopp*	Aliya	Julian		
11:30	Ani	Redekopp*	Ani	Jack	Ani	Redekopp*	Aliya	Julian		
12:00					Caleb		Aliya	Corvyn		
12:30					Caleb		Aliya	Corvyn		
1:00		Redekopp*	Nandhini	Redekopp*	Caleb		Corvyn	Nandhini		
1:30	Jerry	Redekopp*	Nandhini	Alex	Caleb		Corvyn	Nandhini		
2:00	Jerry	Redekopp*	Lecture		Jack	Alex	Lecture		Corvyn	Redekopp*
2:30	Jerry				Jack	Alex			Corvyn	Redekopp*
3:00	Jerry				Jack	Alex			Corvyn	Redekopp*
3:30					Jack				Corvyn	Julian
4:00			Jerry	Aliya	Caleb	Samuel	Qinyi	Alex		Julian
4:30			Jerry	Aliya	Caleb	Samuel	Qinyi	Alex		Julian
5:00			Jerry	Aliya	Caleb	Samuel	Qinyi	Alex		Julian
5:30			Jerry	Aliya	Caleb	Samuel	Qinyi	Alex		
6:00										
6:30										
	All OH marked with an asterisk(*) are located in **EEB 222**									
	All other OH are in SAL									

Schedule: Exams and Assignments

- Week 1: Binary Representation **HW0**
- Week 2: Integer Operations
- Week 3: Floating-Point Operations **Data Lab 1**
- Week 4: Assembly **Data Lab 2**
- Week 5: Assembly
- Week 6: Assembly **Bomb Lab**
- Week 7: **Exam I** (Oct. 2) and Security Vulnerabilities
- Week 8: Memory Organization
- Week 9: Caching **Attack Lab**
- Week 10: Virtual Memory
- Week 11: Dynamic Memory Allocation and Linking
- Week 12: Processor Organization and **Exam II** (Nov. 8) **Cache Lab**
- Week 13: Processor Organization
- Week 14: Code Optimization and **Thanksgiving**
- Week 15: Cache Coherency and Review **Allocation Lab**
- Week 16: Study Days and **Final** (Dec. 6)

Submitting Your Homework

- You will receive access to a GitHub repository for this class
  - `https://github.com/usc-csci356-fall2018/hw_`**`yourusername`**
- We will add homework assignments to subdirectories of your repository
  - Directories named **`proj1`** through **`proj6`**
  - Inside, you will find instructions and scripts to test your work
- When you are ready:
  - Create a `README.md` in the assignment directory
  - Suppress all debug information (e.g., `printf`)
  - Make sure that your code **compiles without errors in the VM**
  - Commit your code and **push to GitHub**
  - Save and **submit the commit SHA** on the course website



- We will check and grade your commit (multiple submissions are allowed)

Installing the Virtual Machine

- Download the VM (7 GB):
  `http://bytes.usc.edu/files/cs103/install/StudentVM_Spring2018.ova`

- Install VirtualBox from `www.virtualbox.org` (with extension pack)
- [File] > [Import Appliance] > [Select .ova file] > [Next] > [Import]

- The VM will suggest you to login as **`student`** after booting
- The password is **`developer`**
- Fine-tuning and troubleshooting instructions at:
  `http://bytes.usc.edu/cs103/install/`

At the end of this discussion session, we will help you solve problems with the VM on your laptop.

Otherwise, ask on **Piazza**!

On the Virtual Machine

You can use the VM to complete your assignments.

- If you prefer to use your own system (macOS, Windows, Linux) for development, **test your code on the VM before submitting**.
- On the VM, you will (mostly) use these tools:
  - **Linux shell** (`bash`): to run commands and manage files
  - **GCC** to compile C programs
  - **GIT** to interact with your GitHub repository and submit assignments

Linux Shell

- Most commands also on **macOS** (natively) and **Windows** (Cygwin)
- Unix philosophy (Ken Thompson and others at Bell Labs):
  - Make each program do one thing well
  - Expect the output of every program to become input to another
  - Design and build software to be tried early
  - Use tools in preference to unskilled help to lighten programming tasks

In the VM, start the **Gnome console**. At the bash prompt, you can:

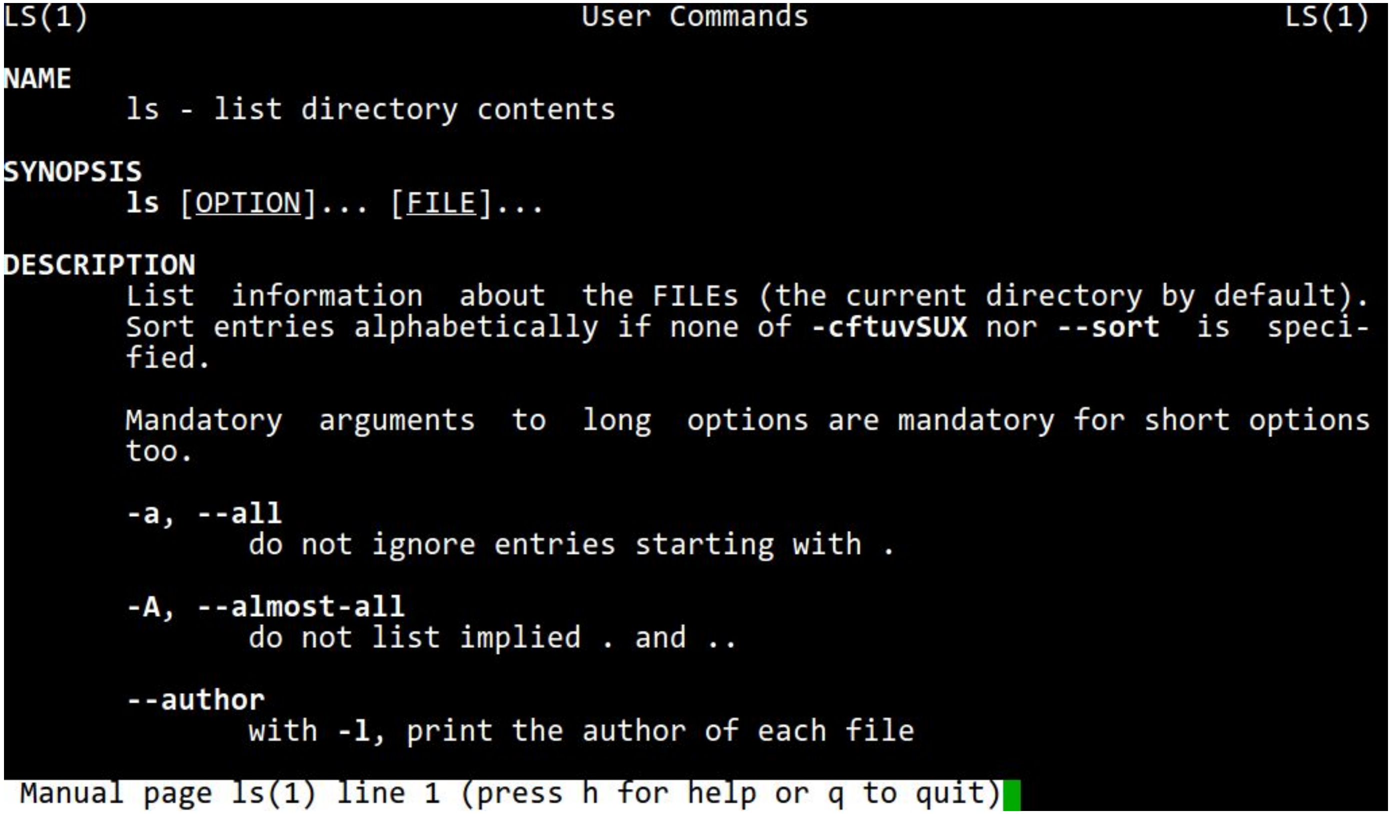
- Give commands (type command, then ENTER)
- Navigate history of commands (UP/DOWN keys)
- Search history of commands (CTRL + r)
- Close the shell (CTRL + d)

```
marco@mycomputer:~$
```

- The prompt shows (by default): user name, host name, current directory

Linux Shell: Asking for Help

```
$ man ls
```



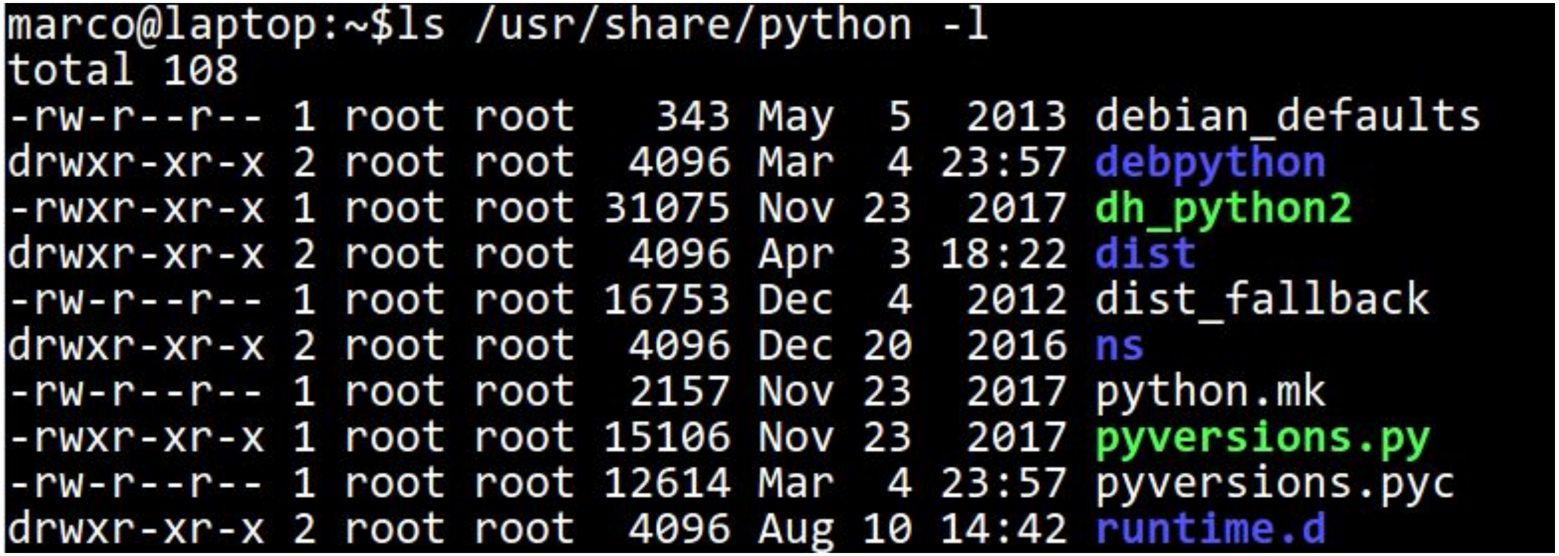
```
LS(1)                            User Commands                           LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

 Manual page ls(1) line 1 (press h for help or q to quit)
```

Search with `/word`, **n** for next match, **N** for previous match, **q** to quit man page

Linux Shell: Working with Files

- Change directory: `$ cd myrepo`
- List files
  - `$ ls` (show files in current directory)
  - `$ ls -a` (do not ignore entries starting with a dot)
  - `$ ls -l` (show permissions, owner, group, size, date)
  - `$ ls -lh` (show size in kB / MB / GB)
  - `$ ls -lht` (sort by time, newest first)
- Copy or move files
  - `$ cp a.txt b.txt` (create copy of a file)
  - `$ cp *.txt dir1` (copy all text files to existing directory)
  - `$ mv *txt dir1 dir2` (move all text files and `dir1` inside `dir2`)
  - `$ cp -r dir2/dir1 dir3` (recursively copy `dir1` inside `dir3`)
- Create directory (`mkdir dir1`) and remove empty directory (`rmdir dir1`)
- Remove files (`rm file1`) and non-empty directories (`rm -r dir2`):

Linux Shell: File Permissions



```
marco@laptop:~$ls /usr/share/python -l
total 108
-rw-r--r-- 1 root root   343 May  5  2013 debian_defaults
drwxr-xr-x 2 root root  4096 Mar  4 23:57 debpython
-rwxr-xr-x 1 root root 31075 Nov 23  2017 dh_python2
drwxr-xr-x 2 root root  4096 Apr  3 18:22 dist
-rw-r--r-- 1 root root 16753 Dec  4  2012 dist_fallback
drwxr-xr-x 2 root root  4096 Dec 20  2016 ns
-rw-r--r-- 1 root root  2157 Nov 23  2017 python.mk
-rwxr-xr-x 1 root root 15106 Nov 23  2017 pyversions.py
-rw-r--r-- 1 root root 12614 Mar  4 23:57 pyversions.pyc
drwxr-xr-x 2 root root  4096 Aug 10 14:42 runtime.d
```

`-rwxrwxrwx`

- Permissions for owner, group, other users (starts with `-` for files, `d` for dirs)
- Change owner/group of all files recursively: `$ chown -R user:group *`
- Set typical directory permissions: `$ chmod 755 dir1`
- Set typical file permissions: `$ chmod 644 file.txt`

Linux Shell: Useful commands

- Filter file by line:
  - `$ grep gold words.txt` (find lines containing gold)
  - `$ grep -v gold words.txt` (find lines **not** containing gold)
- Count lines, words, bytes in a file
  - `$ wc hello.txt` (count lines, words, bytes)
  - `$ wc -l hello.txt` (count just lines)
- Sort lines in a file
  - `$ sort hello.txt` (print sorted lines)
  - `$ sort -u hello.txt` (print sorted lines, removing duplicates)
- Replace line text using regular expressions, print the result
  - `$ sed 's/Hello/Ciao/' hello.txt` (replace first occurrence)
  - `$ sed 's/\t/    /g' hello.txt` (replace all tabs with 4 spaces)
- Create an empty file: `$ touch empty.txt`
- Find files by name: `$ find /home/marco -name "*.txt" -type f`

Linux Shell: Working with Streams

- Print a string to output stream:

  -
    ```
    $ echo "Hello World"
    Hello World
    ```
- Redirect command output to file:
  - `$ echo -n "Hello World" > hello.txt` (no \n, redirect stdout)
  - `$ echo "!!" >> hello.txt` (append, don't create a new file)
  - `$ echo "!!" 1>> hello.txt` (equivalent, stdout is stream #1)
- Concatenate files:
  - `$ cat hello.txt hello.txt` (print "Hello World!!" twice)
  - `$ cat hello.txt hello.txt > twice.txt` (save to file)
- Use a file as program input (stream #0)
  - `$ tr H B < hello.txt` (print "Bello World!!")
  - `$ cat < hello.txt` (print "Hello World!!")
- Redirect error stream (stream #2) to file:
  - `$ cat /etc/sudoers 2> hello.txt` (write error msg to file)

Linux Shell: Working with Pipes

“Expect the output of every program to become input to another.”

- Filter file and count/sort matching lines

  ```
  $ grep gold words.txt   | wc -l
  $ grep gold < words.txt | wc -l
  $ cat words.txt | grep gold | wc -l
  $ cat words.txt | grep gold | sort -u
  $ cat words.txt | grep gold | sort -u > out.txt
  ```

- Examine text files

  ```
  $ less file.txt
  $ cat file1.txt | less
  $ cat file1.txt file2.txt | head -n10
  $ cat file1.txt file2.txt | sort | tail -n10
  # tail -F /var/log/messages
  ```

Linux Shell: System Commands

- System resources
  - Disk space: `$ df -h`
  - Available RAM: `$ free -h`
  - Available CPUs/cores: `$ lscpu`
- Running processes and CPU usage
  - `$ top`
  - `$ htop`
- Connected/mounted disks:
  - `$ lsblk -a`
- Network interfaces and connections:
  - `$ netstat -ta`
  - `$ ip addr show`
  - `$ ip route show`
  - `# ifconfig`

GCC: The GNU Compiler Collection

- Many front-end languages: C, C++, Objective-C, Fortran, Ada
- Many target architectures: x86-64 (`-m64`), i386 (`-m32`), ARM
- `gcc` runs
  - the **compiler** (C to assembly)
  - the **assembler** (assembly to object code)
  - the **linker** (combine many binary objects to obtain final executable)

```c
#include <stdio.h>

/* multi-line
   comment  */

// single-line comment (C99)
int main(void) {
    char *str = "world";
    printf("Hello, %s!\n", str);
    return 0;
}
```

Compiling and Running

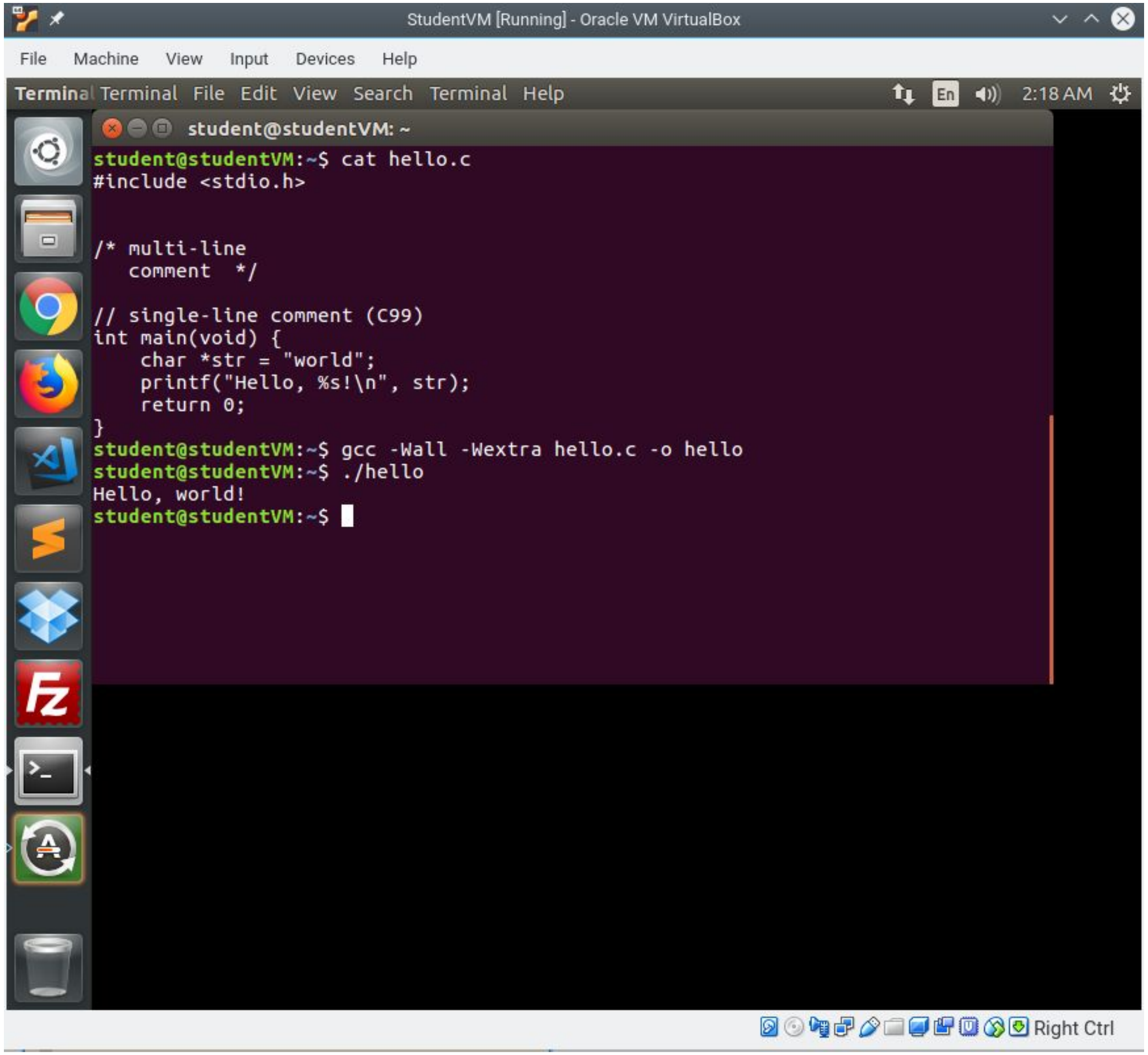
```
$ gcc -Wall -Wextra hello.c -o hello
$ ./hello
Hello, world!
```

Checking the compiler version

```
$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.5) 5.4.0 20160609
```

We will explore GCC in the rest of the class...

GCC: Try now on the VM



StudentVM [Running] - Oracle VM VirtualBox
File
Machine
View
Input
Devices
Help
Terminal Terminal File Edit View Search Terminal Help
En
2:18 AM
student@studentVM: ~
Right Ctrl

```
student@studentVM:~$ cat hello.c
#include <stdio.h>


/* multi-line
   comment  */

// single-line comment (C99)
int main(void) {
    char *str = "world";
    printf("Hello, %s!\n", str);
    return 0;
}
student@studentVM:~$ gcc -Wall -Wextra hello.c -o hello
student@studentVM:~$ ./hello
Hello, world!
student@studentVM:~$ 
```

Git: Distributed Version Control

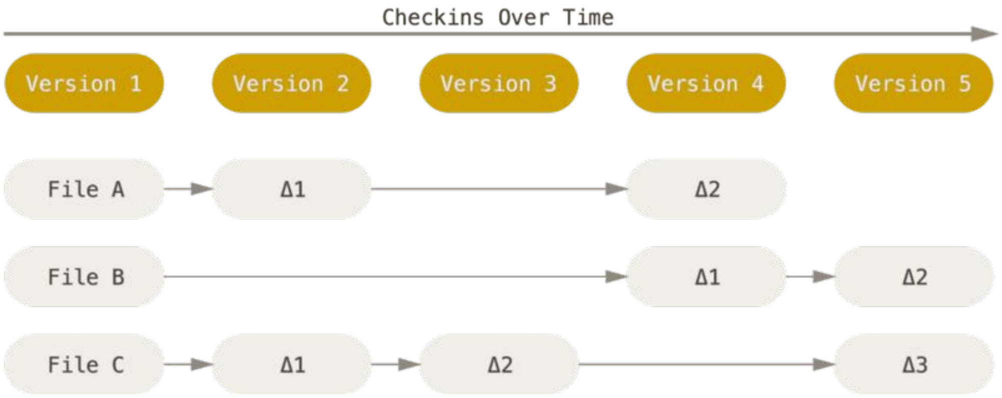
- The most used version control system (others: `cvs`, `svn`, hg, `bzr`)
- Key features:
  - **distributed** (every user has a local copy of the history)
  - **snapshot-based** (not deltas)
  - **simple branching**
- Great references:
  - Pro Git (git-scm.com/book)
  - Visual Guide (marklodato.github.io/visual-git-guide)

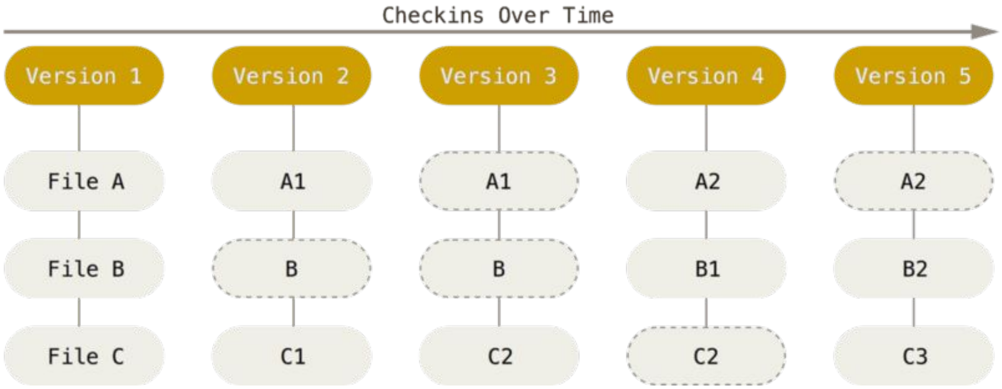
Git and GitHub Setup

- On GitHub
  - In [Settings] > [Profile]: Your public email should be your USC one
  - In [Settings] > [Email]: Your primary email should be your USC one (`username@usc.edu` used in the form)
- Generate a new SSH key
  - `$ ssh-keygen -t rsa -b 4096 -C "username@usc.edu"`
  - Add the contents of **`id_rsa.pub`** to GitHub [Settings] > [SSH Keys]: `help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/`
- Setup Git
  - `$ git config --global user.name "Linus Torvalds"`
  - `$ git config --global user.email username@usc.edu`
- Test setup:
  - `$ git clone git@github.com:you/your-repo.git`

Git: Snapshots, not differences

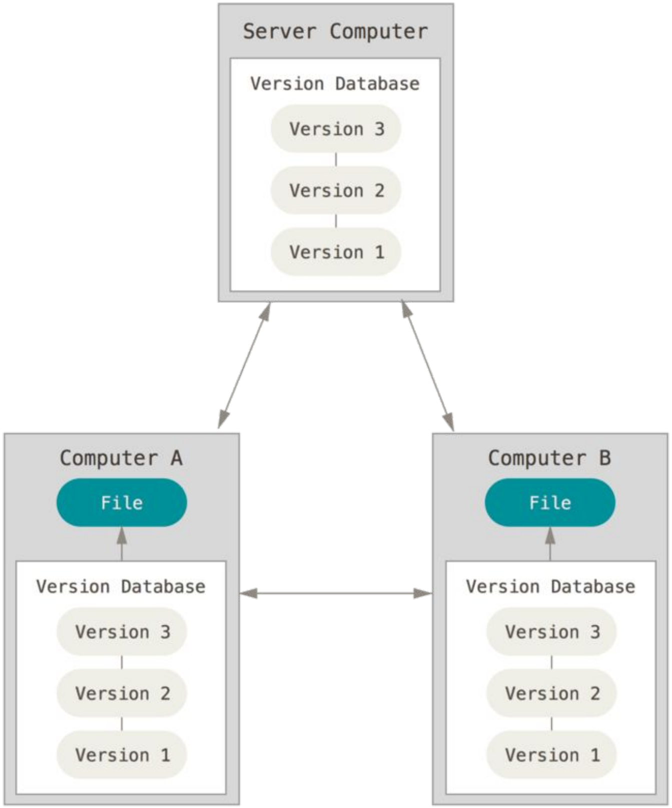
Git commits store a snapshot of all files, not deltas wrt previous versions





Git: Distributed

- Users operate on **local copies** of the history (fast, works offline)
- Most operations **only add data** (can always recover previous versions)
- Files are identified by SHA-1 hashes

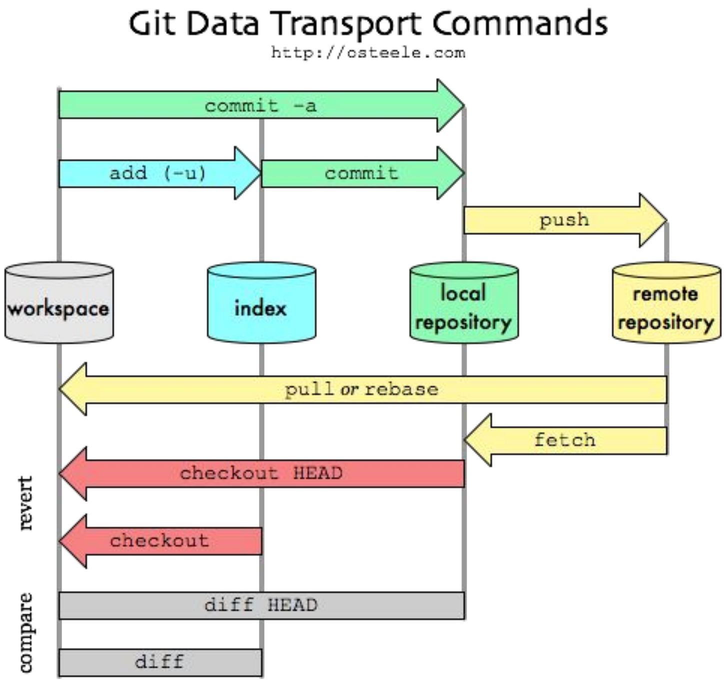


The user decides when to synchronize:

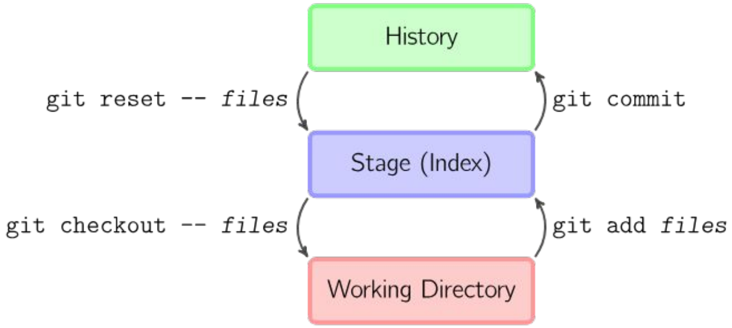
- **Pull** to obtain remote changes
- **Push** to make local changes known to other users
- Usually: **pull**, **merge**, **push**

Git: Working Directory, Index, History

- **Git repository** (index, history, settings) lives inside `.git`
- **Working directory**: the files in the directory tree, your workspace
  - Used for real work (writing C files)
  - Ignore patterns from `.gitignore`
- **Index** (staging area): files ready to be saved in a commit
  - `git add file.txt` adds to index
  - Current version is added: later changes must be added again (check with `$ git status`)
- **History:** a graph of commits, each with author, data, and a snapshot of all files (check with `$ git log --graph`)



Git: Commit


History

git reset -- *files*

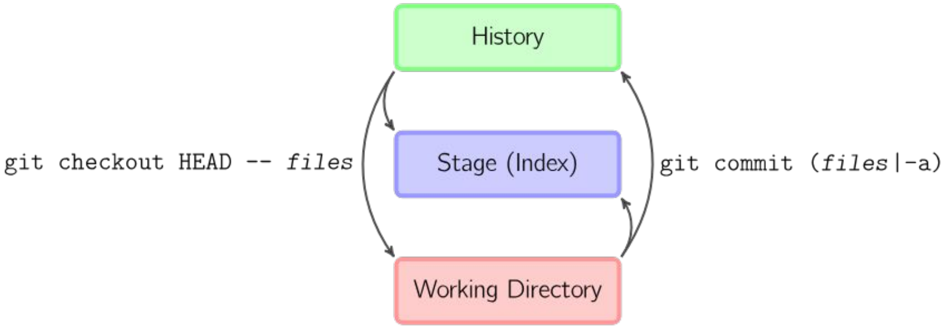
git commit

Stage (Index)

git checkout -- *files*

git add *files*

Working Directory


History

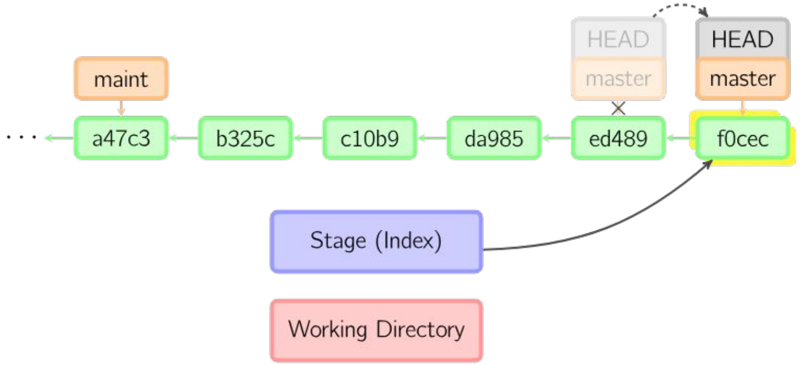
git checkout HEAD -- *files*

Stage (Index)

git commit (*files*|-a)

Working Directory

git commit


HEAD

master

maint

HEAD

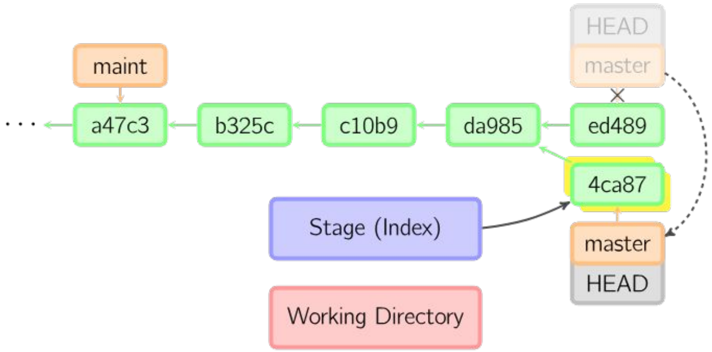
master

··· a47c3 b325c c10b9 da985 ed489 f0cec

Stage (Index)

Working Directory

git commit --amend


HEAD

master

maint

··· a47c3 b325c c10b9 da985 ed489

4ca87

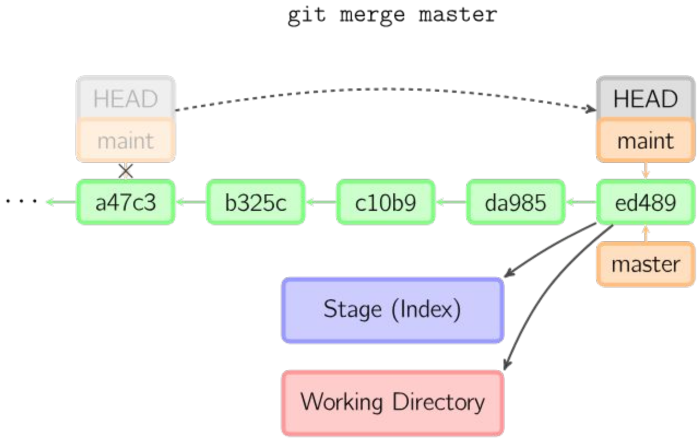
Stage (Index)

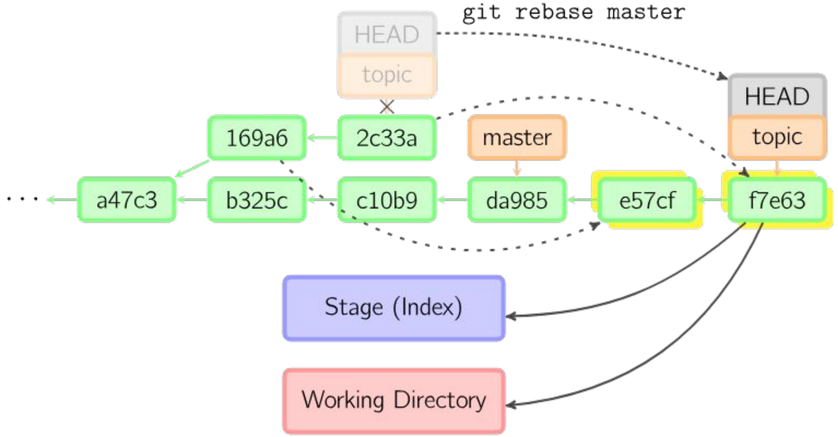
master

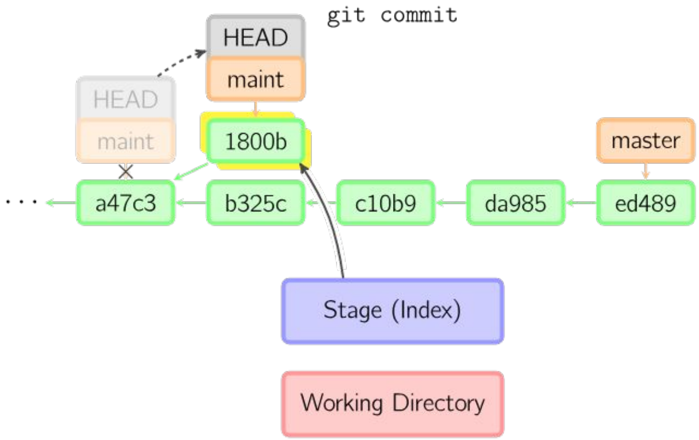
HEAD

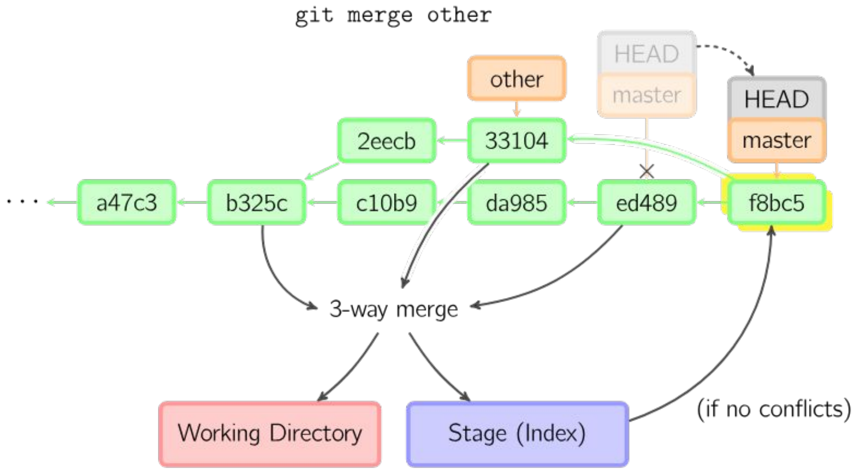
Working Directory

Git: Diverging Histories


git merge master

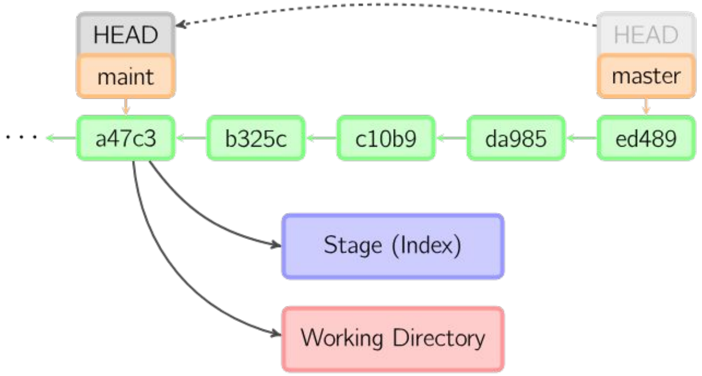

git rebase master


git commit

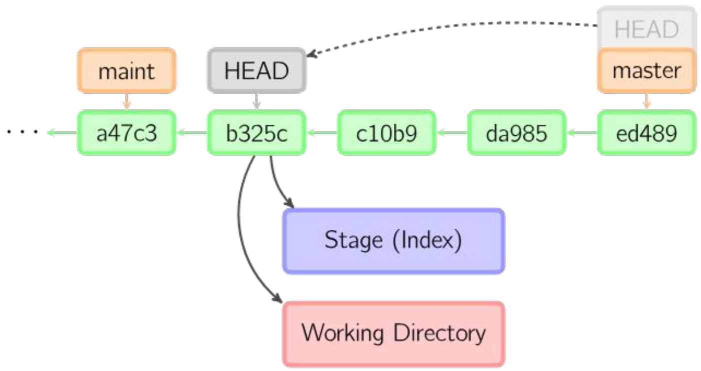

git merge other

Git: Moving around History

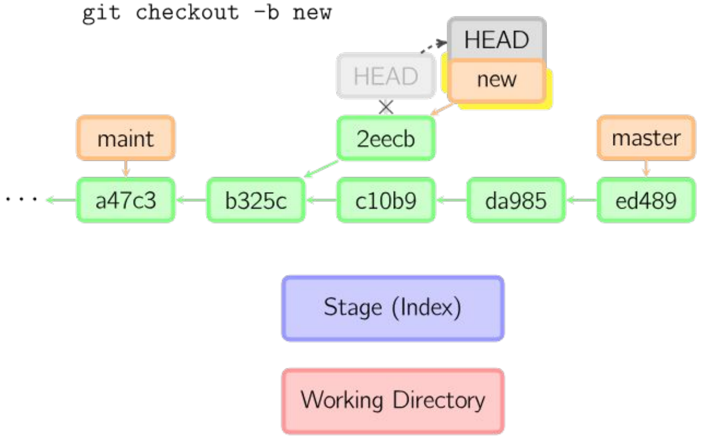
`git checkout maint`



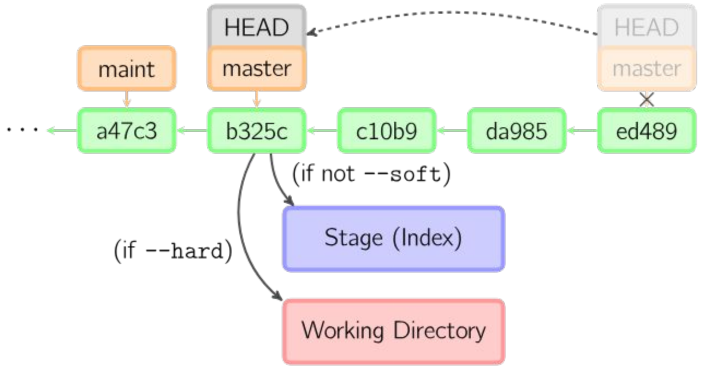
`git checkout master~3`



`git checkout -b new`



`git reset HEAD~3`


HEAD
HEAD
maint
master
master
. . .
a47c3
b325c
c10b9
da985
ed489
(if not --soft)
Stage (Index)
(if --hard)
Working Directory