



Women in Computer Science at LSU

Spring 2023

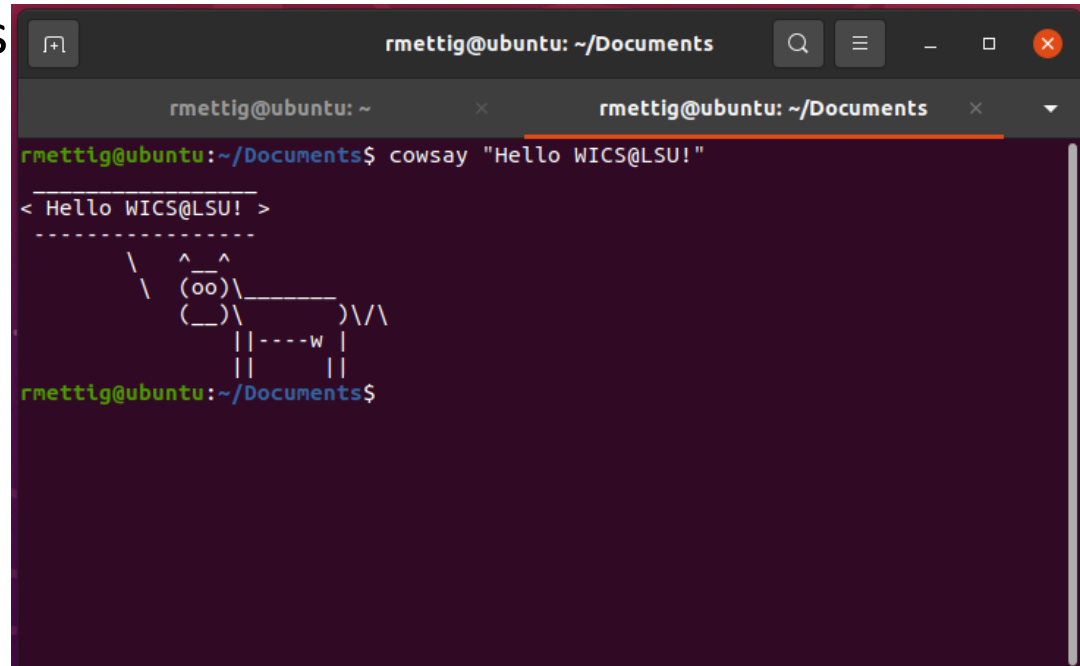
Introduction to Git

A few considerations...

- This is a workshop for all levels...
 - ...but we start from the ground up
- No prior experience required
- There's something for every level
- Follow along to the practical demo
- Ask questions
- Have fun! 😊

In the previous chapter...

- Command Line Interface (CLI) :
Text-based interface that allows you to issue commands to the computer
- Navigate Linux file system
- Create/delete/edit files
- Manage packages



```
rmettig@ubuntu: ~/Documents
rmettig@ubuntu: ~
rmettig@ubuntu: ~/Documents$ cowsay "Hello WICS@LSU!"
< Hello WICS@LSU! >
  ^__^
  (oo)\_______
      (__)\       )\/\
         ||----w |
         ||     ||
rmettig@ubuntu:~/Documents$
```

What is Git

- Version Control System (VCS)
- Multiple different versions of the same project
- Used *everywhere* in the industry



Why Git?

- Backup our work
- Keep track of changes
 - Doesn't apply only to code!
- Freedom to try different approaches and make mistakes
- Contribute to open source projects (good for resume)

Workshop

- Linux-based
 - Kali Linux VMs
- Pre-reqs
 - CLI basics (navigation, file mgmt., etc.)
 - Github account
- Basic commands
 - Staging
 - Committing
 - Roll backs
 - Pushing to remote repository
 - ...more
- Exercises

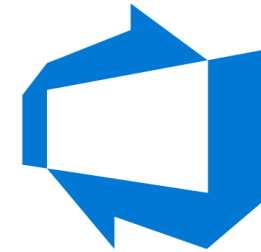
How to Git

Git vs Github

- Git
 - VCS tool to manage your code
 - Works locally
- Github
 - Cloud-based host for Git repositories



There are others!



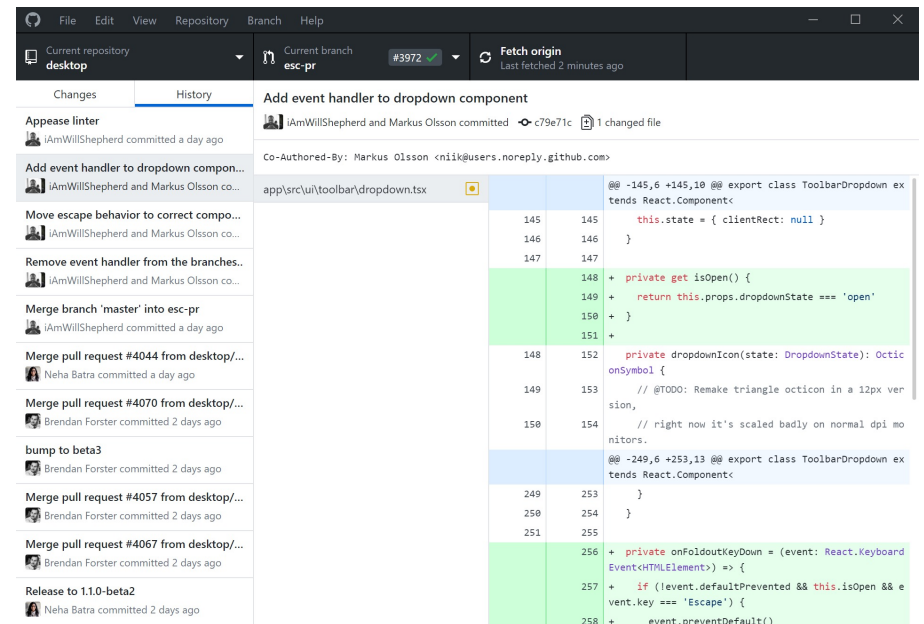
Azure DevOps



GitLab

Desktop vs CLI

- Github has a desktop client
 - It's really easy to use!
- Git concepts are the same across all platforms
- Github also has their own CLI tool
 - gh – we will not be using it
- CLI is less user-friendly
 - ...sometimes it's your only option



The screenshot shows the GitHub Desktop application. The top bar includes a menu (File, Edit, View, Repository, Branch, Help) and a status bar showing the current repository as 'desktop', the current branch as 'esc-pr' with commit #3972, and a 'Fetch origin' button. The left sidebar displays a list of commit history items, including 'Add event handler to dropdown component', 'Move escape behavior to correct compo...', 'Remove event handler from the branches..', 'Merge branch 'master' into esc-pr', 'Merge pull request #4044 from desktop/...', 'Merge pull request #4070 from desktop/...', 'bump to beta3', 'Merge pull request #4057 from desktop/...', 'Merge pull request #4067 from desktop/...', and 'Release to 1.10-beta2'. The main area shows a diff for the file 'app\src\ui\toolbar\dropdown.tsx'. The diff highlights changes between lines 145-147 and 148-151, showing the addition of a 'private get isOpen()' method and a 'private dropdownIcon' property. The code is syntax-highlighted, and the diff uses green and blue backgrounds to indicate additions and deletions.

Useful Terminology

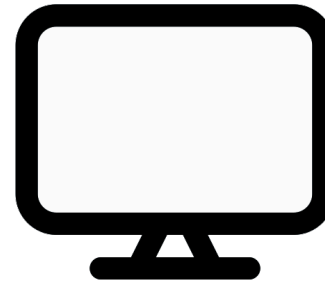
Remote

- Needs network connection
- Only keeps backup of what you push



Local

- Works offline
- On your machine only



Git Terminology

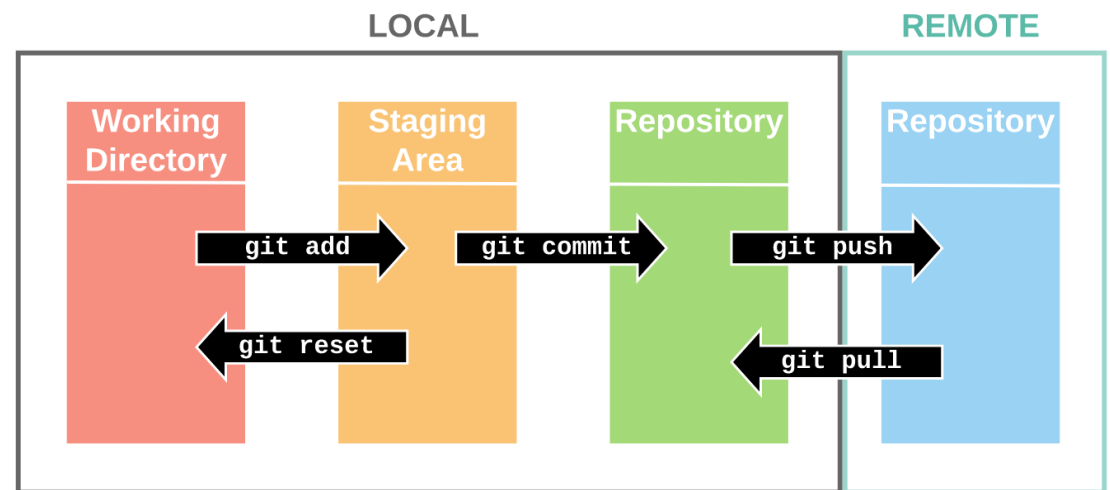
- Repository
 - Contains project files and stores file version history
- Branch
 - A version timeline within a repository
- Commit
 - “Save” the changes done to the project to the working version
- Clone
 - A local copy of a remote repository
- Fork
 - Personal copy of another user’s repository
 - Changes made to your fork don’t affect the other user’s directly

Git Terminology (cont.)

- Fetch
 - Get changes from a remote repository without merging them
- Merge
 - Get changes from one branch and applies it into another
 - Either in the same repository or on another fork (aka Pull Request)
- Pull
 - Fetch and merge changes in one command
- Push
 - Send committed changes to a remote repository
- Pull Request (PR)
 - proposed changes to a repository submitted by a user and accepted or rejected by a repository's collaborators

General Workflow

- Clone/Fork/Create repository
- Create working branch
- Work and make changes
- Commit changes (frequently)
- Push local changes
- Merge changes into main branch



Img src: <https://support.nesi.org.nz/hc/en-gb/articles/360001508515-Git-Reference-Sheet>

Git Commands

Command	Purpose	Command	Purpose
git config	check local repo config	git branch	
git config user.name "username"	set local repo username (--global flag can set for all)	git config user.email "email"	set local repo email (--global flag can set for all)
git init	initialize git repo	git push -u origin <dest-branch>	push local changes to destination branch
git clone <url>	download remote repo	git pull	fetch and merge remote changes to working repo
git add <file>	add files to staging	git merge <branch>	merge changes into your current branch
git status	check staging status	git diff <branch-1> <branch-2>	show changes
git commit -m "<message>"	commit current changes to local working dir	git reset --hard origin/main	rollback working dir to that of last commit erasing changes
git remote	show remote version of repository	git reset --soft origin/main	rollback working dir to that of last commit keeping changes
git checkout <branch>	change branch	git log	list of commits on a branch

More commands

Command	Purpose	Command	Purpose
<code>rm -rf .git</code>	undo \$git init	<code>git checkout -b <new-branch></code>	create a new branch and switch to it
<code>git branch -a</code>	list all branches including remote	<code>git help</code>	git CLI manual!
<code>git --version</code>	check current version (or if it's installed!)	<code>git branch -M main</code>	rename branch as main
<code>git config --list --show-origin</code>	see current config settings	<code>git remote add origin <url></code>	link working repo to remote repo!!
<code>git rm --cached <file></code>	remove file from staging	<code>git restore <file></code>	reverts changes in staged file

Before we start – setup personal access token

- Security measure
 - Avoid using your password
 - Limit access scope
 - Can be expired and needs to be reissued periodically
- Most VCS hosts will make you use it
 - Necessary for APIs – DO NOT STORE THESE PUBLICLY!!!!!! EVER!!!
 - Their docs have walkthroughs
 - But it can be confusing!
- More info: <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>

More considerations

- Access token will be up to 7 days
- Please do all your work on your Desktop folder!
 - We will delete all the local repositories after
- Please make sure you push all the changes by the end of the workshop
 - If you are not sure if they went through, please let me know before we leave and I will help you

Access Token Setup

Please follow along!

Exercises pt. 1 – Simple Initialization

1. Navigate to home Desktop folder, create new dir and initialize it as a git repo
2. Create remote repository “myfirstrepo”
 1. Add remote repo as origin
3. Check current staging status
4. Create file named README.md
 1. Add some text to it if you’d like, such as “WICS workshops and Vim Rulez!” 😊
5. Add that file to your staging area and check status
6. Commit that file with a message saying “first commit”
7. Check staging area again – it should be clear
8. Check your commit log – do you see your commit on there?
9. Push changes to remote repository
 1. Check the repo on Github to see if your changes went through!

Exercises pt.2 – Personal webpage with Github Pages

1. Via the Github UI, create a new public repository titled `<your_username>.github.io`
2. Via the CLI, navigate to Desktop and from there clone the remote repository we just created
 1. Navigate into it
3. From the CLI, create a new file titled `index.html` and write “Hello WICS! This is my first web page” and save it
4. Stage and commit the changes
 1. Commit message should be “first commit”
5. Push the changes to the remote repository
6. Navigate to `https://<your_username>.github.io`
- Homework: go through the docs and fiddle around with your personal webpage!
 - Link in “References” slide

Break

There are more exercises after!

Exercises pt. 3 – Staging area

- The goal with this exercise is for you to get some practice with the staging area – **run git status in between every command to see what happened!**
 1. In the directory/repo from exercise 1, create a new file “file1” and make changes to it.
 2. Add it to the staging area.
 3. Repeat steps 1 and 2 – create “file2”.
 4. Remove one of the files from the staging area – don’t delete it!
 5. Repeat steps 1 and 2 – you should have 3 files now.
 6. Make a change to “file1”.
 7. Revert that change.
 8. Commit with message “committing file 1 and 3”
 1. What are the contents of file1 now?
 9. Make edits to file 1 and 2.
 10. Commit them with message “edited file 1 and added file 2”

Exercises pt. 4 – Branching/merging

- Continue in the same directory from exercises 1 and 3! On top of checking git status, make sure to check git log between commits and merges!
- 1. Create a new branch called “mybranch” and change into it
- 2. Check the contents of “file1”
- 3. While in mybranch, edit the contents of file1 to “mybranch”
- 4. Commit change “changed contents of file1” and check git log
- 5. Checkout branch “main” and check the contents of file1
 1. Is it different from when you were in “mybranch”?
- 6. Merge mybranch into main
- 7. Create a new fiile “myfile” in branch “main” and list contents of your current directory (ls) , then stage and commit it
- 8. Checkout mybranch and create a new file “otherfile” and list the contents of your directory, then stage and commit it
- 9. Change back into branch main and merge “mybranch”, then list the contents of your directory
- 10. Push changes to your remote repository!

Next steps:

- Get comfortable with the commands we've discussed
 - Do the exercises!
- Search for project on Github you're interested in
 - Clone/fork it!
- Practice creating, breaking, and fixing your own repos!
- Get familiar with Github as a platform
- Sign up for Github Student!
 - <https://education.github.com/students>

Further reading

- “Linux Basics for Hackers”
 - NoStarchPress link: <https://nostarch.com/linuxbasicsforhackers>
- Github documentation

<https://nostarch.com/linuxbasicsforhackers>

References

- Git documentation - <https://git-scm.com/>
- Github documentation - <https://docs.github.com/en>
- Git cheat sheet - <https://training.github.com/downloads/github-git-cheat-sheet.pdf>
- Github Pages - <https://pages.github.com/>
- Github Pages documentation- <https://docs.github.com/en/pages>
- Github Student - <https://education.github.com/students>

For next time:

- Be comfortable committing and push/pulling changes
- Workshop: part 3 – Working on shared projects
 - Working on a shared repo
 - Set up our own personal webpage



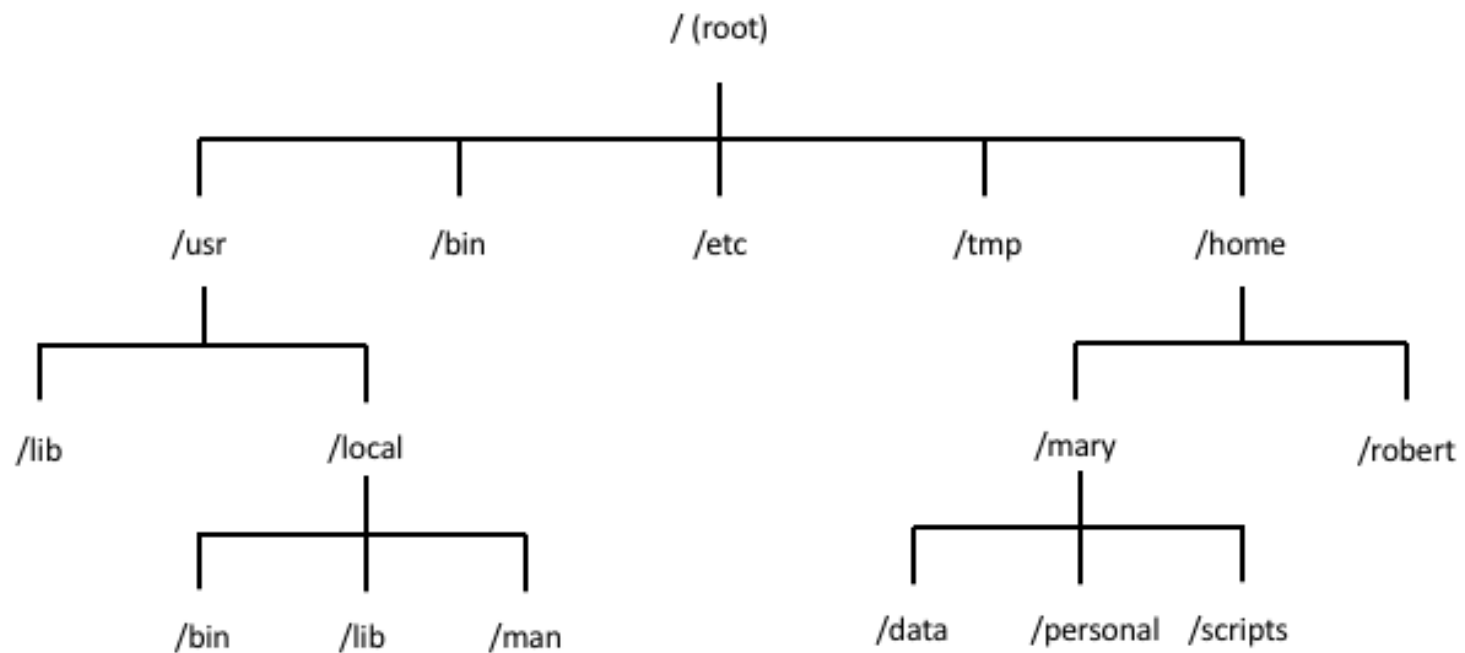
PS: How to find a personal project

- What kind of project are you looking to do?
 - Web dev, tools development, build an app, game hacking...?
- What field are you leaning towards?
 - Applied sciences, software performance, full-stack, DevOps, cyber, ...?
 - It doesn't matter what you pick – all will require SWE fundamentals
- Why do you want to work on the project?
 - Yes, we know you want to build your resume
 - Learn new tool (e.g.: Docker, Metasploit,..), learn new prog lang(e.g.: python, javascript, c++, ...), learn a new skill (e.g.: design, SW orchestration, pipeline development, using APIs, ...)
- Think about your day-to-day
 - What can be automated?
 - What is **ONE** skill you really want to learn?
- Take some time exploring your options << really important!

Appendix

CLI basics reference

Linux File System Overview



Navigation Commands

Command	Purpose	Command	Purpose
pwd	path of working dir	cd ..	change back one level
ls	list contents of dir	cd ../..	change back two levels
ls -l	list contents long format	cd ../../..	change back three levels
ls -a	list all contents in dir	...	so on and so forth
ls -t	list and sort by time	cd /	change to root dir
ls <target dir>	list contents of target dir	cd	change to home dir
cd <target dir>	change directory to target dir	cd ~	change to home dir

File/dir creation and editing

Command	Purpose	Command	Purpose
man <command>	manual!! Check this when in doubt!!	mv <old name> <new name>	renames a new file or dir
touch <file name>	creates an empty file	rm <file name>	deletes a file or empty dir
mkdir <name>	creates empty dir	rm -r <dir name>	recursively deletes all files in non-empty dir
mv <file> <target>	moves a file to the specified target dir	rm -rf <dir name>	same as above, but forces and overrides any warnings
cp <file> <target file>	copies a file into the specified target file	head -10 <file name>	shows first 10 lines in file
cat	see file contents	tail -10 <file name>	shows last 10 lines in file
mkdir	creates new dir	nl <file name>	shows the contents with numbered lines

Adding content to file

- Append or overwrite with echo

A terminal window with a dark purple background and light green text. The window title is 'rmettig@ubuntu: -'. The terminal shows a sequence of commands and their outputs: 'echo "hello " > hello.txt' followed by 'cat hello.txt' which outputs 'hello'; then 'echo "world" >> hello.txt' followed by 'cat hello.txt' which outputs 'hello' and 'world' on separate lines; finally 'echo "goodbye" > hello.txt' followed by 'cat hello.txt' which outputs 'goodbye'. The prompt 'rmettig@ubuntu:~\$' is shown at the end.

```
rmettig@ubuntu:~$ echo "hello " > hello.txt
rmettig@ubuntu:~$ cat hello.txt
hello
rmettig@ubuntu:~$ echo "world" >> hello.txt
rmettig@ubuntu:~$ cat hello.txt
hello
world
rmettig@ubuntu:~$ echo "goodbye" > hello.txt
rmettig@ubuntu:~$ cat hello.txt
goodbye
rmettig@ubuntu:~$
```

- Text editor

Vim basics

Command	Purpose	Command	Purpose
vim <file>	open file in Vim	:q	quit
[I]	insert mode (edit file)	:w	write to file (save)
[ESC]	return to normal mode	:wq	write then quit
[UP, DOWN, RIGHT, LEFT]	navigate the editor	:x	write changes and close file
:q!	discard changes and quit	:w <new name>	save current file as new file

Vim cheat sheet:

<https://www.cs.cmu.edu/~15131/f17/topics/vim/vim-cheatsheet.pdf>

Package management

Command	Purpose	Command	Purpose
sudo <command>	run command with admin privileges	sudo apt-get upgrade	updates the installed packages in your system
apt-get install <package>	install the package		
apt-get remove <package>	uninstall the package		
apt-get purge <software>	uninstall and remove configuration files		
apt-get update	updates list of packages available for download		

Other useful commands

Command	Purpose	Command	Purpose
history	shows CLI history	ssh	connect to remote server over ssh
! <history number>	runs the command in history	chmod	change file permissions
find <start> -type <f d> -name <name>	scans the FS for a target file or directory	cowsay <text>	prints cow made of ASCII art with input text
grep <string>	will only show results containing string	clear	clear the contents in the terminal window
less <file>	display file contents in a fixed amount of lines	[UP, DOWN]	scroll through recent command history
sed	text replacement	which	checks whether a given command is installed in \$PATH
wc	counts output lines		