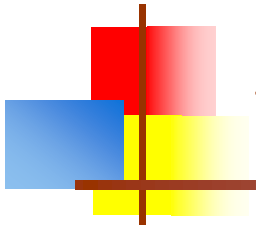
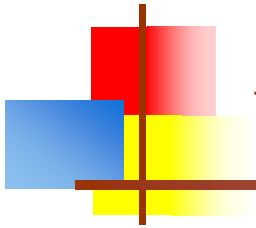
What is Version Control?

- System to manage changes to files
  - Who — vors
  - When — April 17
  - What — Files and differential
  - (Sometimes) Why — To fix Pester tests
- Why use it?
  - Revert to or review prior changes
  - Maintain multiple versions
  - Compare differences
  - Share and collaborate
  - Modern solutions might assume you have it!
    - Infrastructure-as-code
    - Continuous Integration and Delivery
  - Google around for many more reasons!



Version control systems

- Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all "real" projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a "central" repository; users "check out" files, work on them, and "check them in"
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

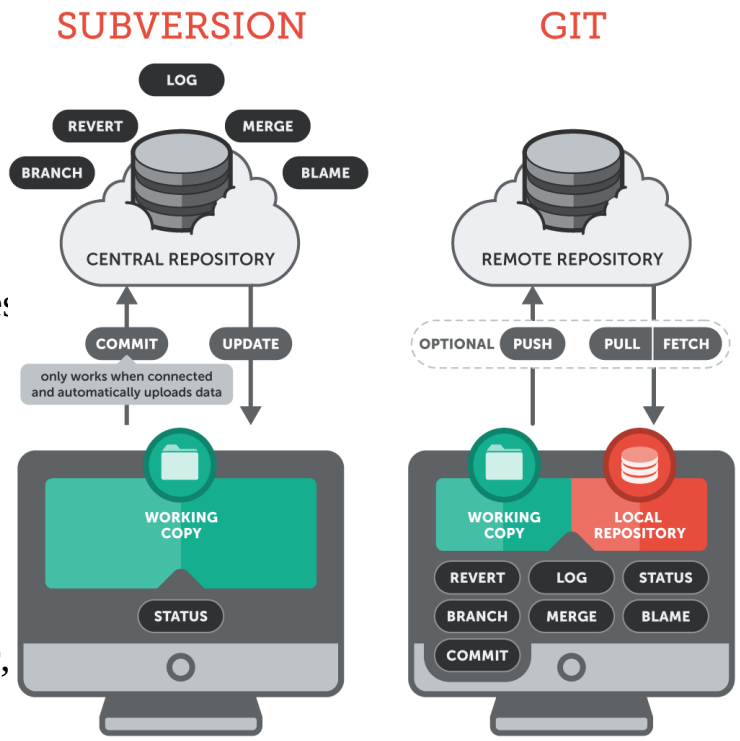


Why version control?

- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- For getting an internship or job:
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work

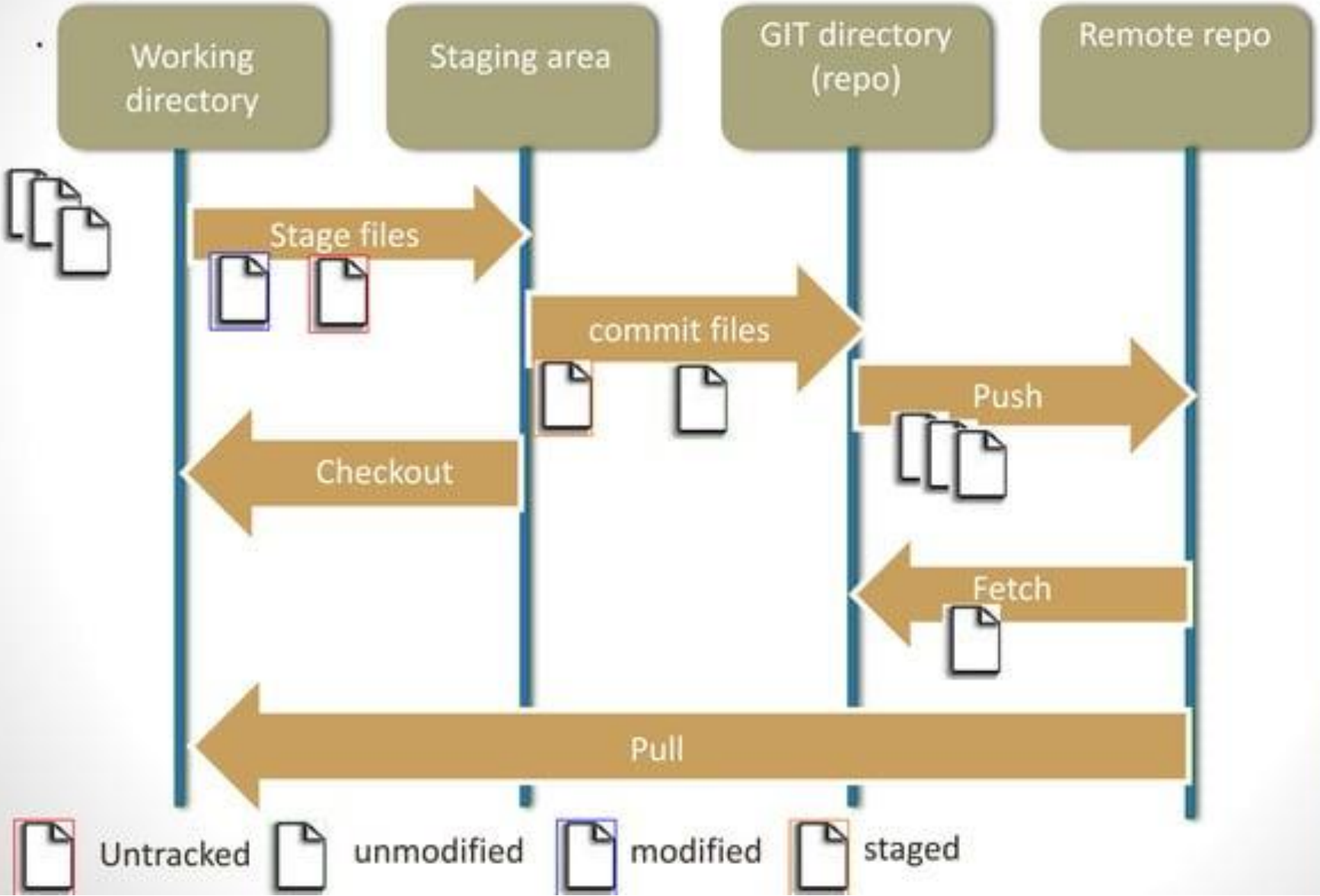
Version Control Systems

- Centralized
  - Work directly against a central server
  - Subversion, CVS, Perforce, etc.
- Distributed
  - Work locally, optionally push to remote repositories
  - Git, Mercurial (Hg), etc.
- Hosted solutions
  - GitHub, BitBucket, Visual Studio Online, etc.
- On-Premise solutions
  - GitHub Enterprise, Stash, Team Foundation Server,



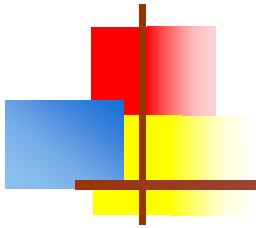
git-tower.com

GIT state operation



.git directory structure

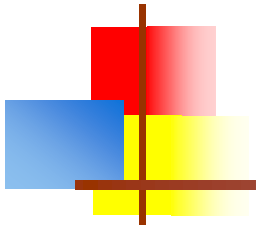
- ├─.git
- ├─ HEAD/ (A pointer to your current branch)
- ├─ config/ (contains all configuration preferences)
- ├─ description/(description of your project)
- ├─ Index/ (is used as staging area between working directory and repo)
- ├─logs/ (keeps records to changes that are made in ref)
- ├─objects/ (all data are stored here: commits, trees and tags)
- ├─ hooks/ (shell scrips that are invoked after executing a command)
- ├─refs/ (holds your local branch remote branch and tags)



Introduce yourself to Git

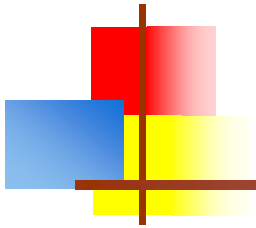
- Start git / gitBash
- Enter these lines (with appropriate changes):
  - `git config --global user.name "John Smith"`
  - `git config --global user.email jsmith@seas.upenn.edu`
- You only need to do this once

- If you want to use a different name/email address for a particular project, you can change it for just that project
  - `cd` to the project directory
  - Use the above commands, but leave out the `--global`



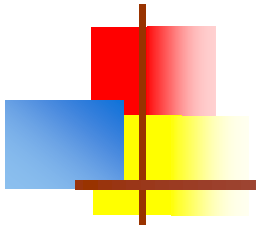
Using your repositories on panther

- Get the files from your repository before starting
  - Make a local respository as a clone of master
    - git clone https://github.com/erenko147/git2024.git
    - git clone git@github.com:erenko147/git2024.git
    - See all the contents of the folder
- Make changes
  - See what changed
    - git diff
  - Stage changes
    - git add –all (or particular files)
    - git add .
    - git diff –cached
    - Still only in your repository



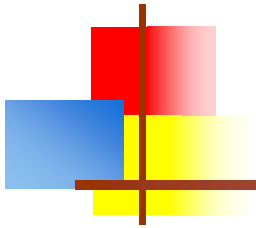
Choose an editor

- When you “commit,” git will require you to type in a commit message
- For longer commit messages, you will use an editor
- The default editor is probably `vim`
- To change the default editor:
  - `git config --global core.editor /usr/bin/vim`
- You may also want to turn on colors:
  - `git config --global color.ui auto`
- See your options:
  - `git config -l`



Using your repository

- Put changes back up into repository
  - Commit your staged changes in your repository
    - git commit -m "the reason for the change"
  - Update the respository:
    - git push origin
- See what is on the repository
  - git remote
- Get what is on repository
  - git pull

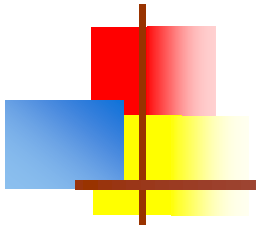


Typical workflow

- `git pull` *`remote_repository`*
  - Get changes from a remote repository and merge them into your own repository
- `git status`
  - See what Git thinks is going on
  - Use this frequently!
- Work on your files
- `git add --all (or just changes)`
- `git commit -m` *“What I did”*
- `git push`

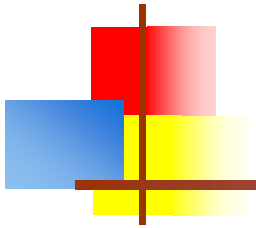
Helpful gitBash commands

- Show staged differences: git diff -- cached
- Show status : git status
- Show branches: git branch
- See history: git log
- Checkout a branch: git checkout branch
- Fetch so you can look but maybe not take: git fetch
- Pull will fetch and merge with what you have: git merge

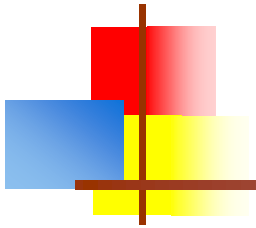


- git config --global user.name "Your Name" # Sets your global username for all repositories on your machine.
- git config --global user.email "you@example.com" # Sets your global email, which will appear in your commit metadata.
- git init # Initializes a new Git repository in the current directory, making this directory track versions with Git.
- git clone <url> # Clones an existing remote repository from the given URL onto your local machine.
- git status # Shows the status of the working directory and staging area. # Tells you which files are untracked, modified, or staged for the next commit.

- git diff # Shows the differences not yet staged or committed. This helps you see what has changed line-by-line.
- git diff <commit> <commit> # Compares changes between two specific commits. Useful for reviewing history.
- git add <file> # Stages a single file's changes to be included in the next commit.
- git add . # Stages all changed files in the current directory for the next commit.
- git commit -m "commit message" # Records a snapshot of the staged changes into the repository’s history with a message describing what was done.
- git commit --amend # Modifies the most recent commit. Useful for quickly fixing a typo in the commit message or adding missed changes.



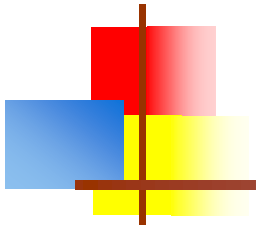
- git branch # Lists all local branches. Highlights the currently checked-out branch.
- git branch <branch_name> # Creates a new branch based on the current commit.
- git checkout <branch_name> # Switches your working directory to the specified branch, updating your files to match it.
- git switch <branch_name> # Another way to switch branches. In newer Git versions, `switch` is recommended over `checkout` for branch switching.
- git checkout -b <branch_name> # Creates a new branch and immediately switches to it.
- git switch -c <branch_name> # Same as above, but using the newer `switch` command to create and switch.



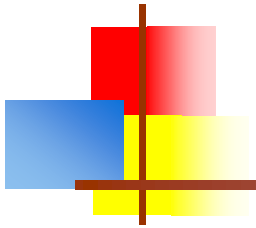
- git merge <branch_name> # Merges the specified branch's history into the currently checked-out branch. Often used to bring feature branch changes back into main.
- git branch -d <branch_name> # Deletes a local branch that you no longer need, once it has been merged or is no longer useful.
- git log # Shows the entire commit history in chronological order. Press `q` to quit.
- git log --oneline # Shows the commit history in a simplified, one-line-per-commit format. Useful for a quick overview.
- git show <commit_hash> # Displays the changes introduced by a specific commit.
- git remote -v # Lists the remote repositories connected to your local repo. Shows their fetch and push URLs.
- git remote add <name> <url> # Adds a new remote repository reference (like "origin") to your local project.

- git fetch # Downloads changes from a remote repository without merging them into your local branches. It updates remote-tracking branches.
- git pull # Fetches and merges changes from the remote into your current branch. Equivalent to `git fetch` + `git merge`. git push # Uploads your local commits to the remote repository.
- git push -u origin <branch_name> # Pushes the specified branch to the remote “origin” and sets it as the default upstream branch for future `git push` or `git pull` commands.
- git stash # Saves your working directory changes temporarily so you can switch branches or pull updates without committing. After stashing, your directory is clean.
- git stash list # Shows all stashes you have stored.

- git stash apply # Applies the most recent stash to your current working directory, restoring those changes.
- git stash pop # Applies the most recent stash to your current working directory and removes that stash from the list.
- git tag # Lists existing tags. Tags mark specific points in the repo's history, usually used for releases.
- git tag -a v1.0 -m "Version 1.0" # Creates an annotated tag named "v1.0" with a message describing the tag.
- git push origin <tag_name> # Pushes a specific tag to the remote repository so others can see that tagged release.
- git checkout -- <file> # Discards changes in your working directory for that file, reverting it to the last committed state.



- git reset <file> # Unstages the file, removing it from the next commit but leaving the changes intact in your working directory.
- git reset --soft <commit> # Moves the HEAD (current pointer) to a previous commit, keeping all changes staged so you can recommit.
- git reset --mixed <commit> # Resets to a previous commit, keeps your working directory changes, but un-stages them.
- git reset --hard <commit> # Resets to a previous commit and discards all changes since that commit. Use with caution.
- git log --oneline --graph --decorate --all # Visualizes the repository's history as a graph, with references to branches and tags.
- git diff <branch1>..<branch2> # Shows changes between two branches, useful for comparing feature branches before merging.
- git clean -f # Removes untracked files from your working directory. Use carefully as this is irreversible.



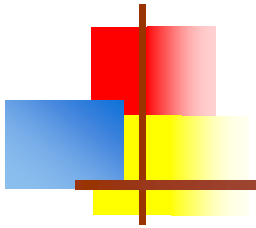
Download and install Git

- https://gitforwindows.org/
- https://git-scm.com/

Adding ssh key to the sysyem

- https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

- ssh-keygen -t ed25519 -C "your_email@example.com"
- ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
- Enter file in which to save the key (/c/Users/YOU/.ssh/id_ALGORITHM):[Press enter]
- 1st line Get-Service -Name ssh-agent | Set-Service -StartupType Manual
- 2nd line Start-Service ssh-agent
- ssh-add c:/Users/YOU/.ssh/id_ed25519
- cat c:/Users/YOU/.ssh/id_ed25519.pub



- LEVEL 1: INITIALIZING & EXPLORING

"You have just arrived in a mysterious land of version control…"

- GOALS:
- Run: git status (to see the state of your repository)
- Run: git remote -v (to view the remote connection)
- CHECK:
- Understand what "git status" and "git remote" show.

- LEVEL 2: STAGING & COMMITTING CHANGES
  "You find a README file that needs your mark."
- GOALS:

- Edit README.md and add a line of text.
- Run: git add README.md
- Run: git commit -m "Add initial documentation line"

- CHECK:

- Run: git log --oneline (to confirm your new commit is recorded)

- LEVEL 3: BRANCHING OUT

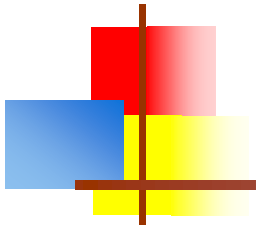
"At a fork in the road, create a new path without disturbing the old."

- GOALS:

- Run: git branch adınızı_yazın
- Run: git checkout adınızı_yazın
- Edit story.txt and add a new line.
- Run: git add story.txt
- Run: git commit -m "Add new storyline on adınızı_yazın"

- CHECK:

- Run: git branch (to see adınız_yazın as current branch)
- Run: git log --oneline (to see your new commit)



(make changes on story.txt ins.)

- LEVEL 4: MERGING & RESOLVING CONFLICTS

"Combine wisdom from both paths, but beware of conflicts! "

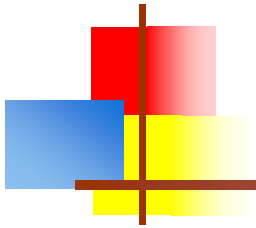
- GOALS:

- Run: git checkout main
- Run: git merge adınızı_yazın (expect a conflict)
- Open story.txt, remove conflict markers, and finalize changes.
- Run: git add story.txt
- Run: git commit

- CHECK:

- Run: git log (to see a merge commit)
- story.txt contains merged content from both branches.

- LEVEL 5: PUSHING CHANGES TO REMOTE
  "Share your newfound knowledge with the world.«
- GOALS:
- Ensure you’re on main branch.
- Run: git push origin main
- CHECK:
- View the repository on GitHub and see updated commits.



- LEVEL 6: TAGGING A MILESTONE

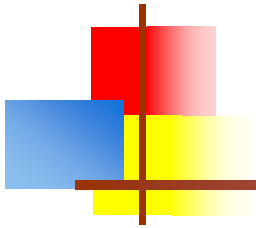
"Mark this moment in time."

- GOALS:

- Run: git tag -a v1.0 -m "First milestone reached"
- Run: git push origin v1.0

- CHECK:

- Tag appears in the remote repository's tags/releases.



Good aliases

- alias gs='git status '
- alias ga='git add '
- alias gb='git branch '
- alias gc='git commit'
- alias gd='git diff'
- alias go='git checkout '
- alias gk='gitk --all&'
- alias gx='gitx --all'
- alias got='git '
- alias get='git '