Repositories (repos) – a set of files being tracked together is a repo. A git folder

Remotes repositories (remotes) – a repo hosted on a server that is used as a centralized copy of the game

Changes – changes to files in a repo; adding, removing, or replacing a file

Commits: a block of changes to files in the repo

Branches: parallel versions of a repo; each with different commits

Working directory: the local copy you can edit; the live copy on your computer that can be edited

GIT vs GITHUB

* Git is a free, open source version-control software
* GitHub is a website that hosts remote repositories; no attachment to Git; a company that sells servers to hosts your repositories

Git Clients

* Many different clients exist but they are all compatible with each other

The Basics

* cd ~/ <directory> : changing into directories
* joining a repo: cloning
  + take a remote repo and making a copy of it
  + git clone <url>

The Basic Git Cycle

1. git add .
   * marks all the current changes you have made to your working directory as changes you want to keep
2. git commit -m ‘<message>’
   * puts added changes together into a packet called a commit. Those changes are now committed
3. git pull
   * this pulls all new commits that have been added to the remote to your local copy (and applies the changes in them); receive the commits the collaborators pushed onto the remote repo
4. git push
   * this pushes all commits you’ve made that you haven’t pushed yet onto the remote ; gives your commits to the remote repo

Saving your name and password

* git config --global user.name “<username”
* git config –global user.password “<password>”

Merge Conflict

* when two commits (usually from 2 different programmers) apply two different changes to the same line of code, a merge conflict happens
* golden rule: one person works on a file at a time

How to deal with Merge Conflicts

* delete the lines you don’t want
* repeat cycle
* commit message should be “fixed merge conflict”

Stashing to Pull without Commitment

1. git stash
   * save all your uncommitted changes, then roll them all back, so only your committed changes remain; you can now pull normally
2. git pull
3. git stash pop
   * re-applies all the uncommitted changes you made then stashed. Do this after you have pulled

* do this when you’re not ready to commit your current changes but need the committed changes from your team members

Branches

* classic simple setup: master branch and dev branch
  + all changes are made to the dev branch and then merged into the master branch each time it makes it to a stable state
* git checkout -b <name of branch> - creates a new branch named <branch> currently identical to your current working branch and the new branch becomes your new working branch
* git checkout <branch> - changes your working branch to the specified branch
* git merge <branch> - applies all commits made to branch since branch and the current working branch forked to current working branch
* Don’t do branch manipulations with uncommitted changes lying around