MAXWELL MANN: I'm going to tell you why I think that organization of your code is the bee's knees. It's the coolest thing ever. If you organize your code well, then you can end up having not a horrible player. I think lecture three robot release is kind of long. I released some code, and it's long. Look at all that stuff. It's especially large when you want have 16 point font so that everybody can see from back at the end. But it's just too much, so here I'm going to propose that you organize your code into, as you can see here on the left, many separate files.

So this is-- I think it's [? Hitav's ?] code. He's one of the devs this year, and you can see here his robot player. He's got the package name, and he imports some things, and right after that, he defines these new types, like base player, headquarters player, soldier player, and encampment player. And each of these things is part of base player. You can see here base player is this public, abstract class that he defines here. And it has certain things he wants to use later. And then later when he has other players, like an encampment player, he can just say, OK, I am going to call this public class encampment player extends base player.

And now, this guy has access to all of this guy's method. And it's also located in his own convenient place. So I'm going to post some example that shows how to get your code separated in a convenient way if you haven't done that before. So you've got your code separated, and you want to make life easy. You don't want to code in a way that's going to make it hard for you to make changes and make updates. Steven Arcangeli, who won Battlecode two or three years ago, has posted on his blog a description of coding best practices that enabled him to really do well in Battlecode. He did things like make sure that things are in separate files, that they have subpackages. And that he worked on this framework code most of the time.

He wasn't trying to just mash out a strategy. He was building systems that he could reuse and use multiple times. He also made a rapid iteration system so he could test his code against other ways of running the same code. And today, Aaron is here with me to set up a system for trading code between us so we can work more efficiently. So according to Stephen, he was doing strategy only in the last week of the competition. The rest of the time, he was basically just writing navigation code, or something very basic, and then he would sort of figure out the strategy after watching the tournaments, like the sprint tournament and the seeding tournament.

AARON EPSTEIN: OK, just steal the strategy instead of writing it yourself. Awesome.

MAXWELL MANN: Yeah, I guess that's one way to do it. So I'm going to go ahead and switch on over to Aaron, and he's going to show how he would set things up to start sending me files in the most efficient way.

AARON EPSTEIN: Hello, I'm Aaron, the less effective lecturer, and I'm going to tell you that the most efficient way to share files between me and Max is Git. Spoiler alert, it's Git. So I have my directory, it's called user test, but it's a Battlecode installation, the same as you guys have. I have some stuff in here that I want to share with Max, particularly something in the team's directory. This awesome player-- awesomer player-- and I want Max to have it. So how do I get it to him? So first, I made a Bitbucket account.

It's free, it's similar to GitHub, except the problem with GitHub is if you want a private repo, you'd have to pay for it. So you'd probably abusing a public repo, which means anybody who can guess your user name just has access to all your robots, which makes it even easier for them to your code and strategies. It's kind of silly. Bitbucket gives you a private repo with up to five users. We only give you four users, so a Bitbucket's perfect. So you make your Bitbucket account. And then I want to make a repo for Max and I to share code. So I'm going to call it Battlecode example. I want it to be private so that other teams can't steal our stuff. It's going to be Git, because that's I all that I know how to use. It's going to be Java. Now create it.

MAXWELL MANN: So this repository, Aaron, is going to be on the internet, and so we'll really be able to access it from anywhere. It's sort of its own backup system, isn't it?

AARON EPSTEIN: Yep. It's backup, it's version intelligent, cross platform, et cetera. It's perfect.

MAXWELL MANN: Yeah. Yesterday or the day before, my computer gave me a bunch of Blue Screens of Death. I think one of my memory sectors is failing or something. But I wasn't that nervous, because we have everything stored online. True story.

AARON EPSTEIN: Alright. So Bitbucket goes ahead and gives me the instructions. So I'm basically just going to follow these instructions. I've already made the directory and I'm already in it, so I don't need to do this. So right now the installation's not a Git repo. So I'm going to go ahead and make it one. Get in it, now it's empty. And then it tells me to add this thing as remote. I'll do that later. OK, so right now it's an empty Git repository. It's not very useful, because it's empty. If I use Git status, it tells me that it's basically empty. Yeah, it tells me that there's all these things that are untracked. It's not tracking anything. And there's nothing in it.

AUDIENCE: What does it mean when they're tracked or untracked By Git?

AARON EPSTEIN: So it said when I initialized it here that it was empty. So even though there's a bunch of things in this folder, Git doesn't really know or care about them yet, because I haven't told them where they are, or what it should pay attention to and what it shouldn't pay attention to, so it's just empty. That's why it's labeled "everything untracked", because they're not yet tracked. So let me make some of these things tracked. And by some, I mean only the awesomer player, because that's the only thing I want to give Max right now. So I'm going to do Git add, because I want to add it to the list of things that's tracked. And go team/awesomer player/robot.

OK, cool. Now it's added. So now if I look at the status, I have one new file. I didn't make it, so it's not new with respect to the hard drive, but it's new with respect to Git, because Git didn't know about it before. And then these are still on track. So now to share it with Max, I have to commit this change. Right now, it's just been added, but it's sort of floating around in space. So "commit" really solidifies it into a block of changes that I can give to max. So "initial commit", just because that's the best practices name for the first commit.

MAXWELL MANN: You see here he's put a message with it, so that when we look at the commit logs, we'll be able to see his descriptions of what went on for all the changes, all the way back to the beginning. And we'll have all these things posted in the handy-dandy guide-- gosh, I said it again-- so you don't have to worry about memorizing it. Right now you just want to get the overview of how we're doing it, and how many millions of times more awesome it is to do it this way, than to email to one another.

AARON EPSTEIN: I should mention hyphen a means "all", like for everything, and m means give a message here with strings rather than having to type it into an editor. On Windows, if you leave out the m switch, you get notepad, which is not that big of a deal, because it's just notepad. Just type it in and hit Save, and it's okay, and you remember to use the m switch next time. But on Linux, if you forget the m switch, most likely it'll give you Vim, which is so complicated and arcane that it took me many times before I even knew how to close it.

MAXWELL MANN: It's like escape:wq or something. It's absolutely crazy.

AARON EPSTEIN: I think I've gathered that colon q quits it, and colon w saves.

MAXWELL MANN: I think you have to push s to start typing again. It doesn't even let you put commands in.

AARON EPSTEIN: You want to say out of Vim if you're not initiated.

AUDIENCE: [INAUDIBLE]

AARON EPSTEIN: OK. Got it. Well I just the m switch so I can stay out of Vim, because it's only for the initiated.

MAXWELL MANN: I see one question.

AUDIENCE: So do you guys recommend only sharing specific files at a time, instead of the entire project?

AARON EPSTEIN: Yes, and I will discuss that eventually, but for now the only things you want to share are things that you wrote yourself and are changing yourself. But I will discuss more of that in the future. So right now I just want to share that one specific file. So now it's committed. I need to get onto here, so I do this thing that it tells me to do. Basically I'm going to add as a remote, like a thing that's not here. It's a remote, it's somewhere else, so the server. I'm going to add one called "origin", because origin is the canonical name for this sort of thing, and then I follow the URL.

MAXWELL MANN: You can't copy and paste into that?

AARON EPSTEIN: No.

MAXWELL MANN: Oh, you can't?

AARON EPSTEIN: No.

MAXWELL MANN: Oh, mine's better. Right now, he's using MinGW shell to type these commands. If you just put it in Windows command prompt, I don't think it would work. I'm going to use to Git Bash, which you can get--

AARON EPSTEIN: No, you're using Git Shell, which opens Windows PowerShell.

MAXWELL MANN: And you can get that online just by Google searching it.

AARON EPSTEIN: Cool, well there you go. That's how you remove remotes. Two birds, one stone.

MAXWELL MANN: That's pretty useful. We did that on purpose. That guy's a plant. I'll pay you after lecture.

AARON EPSTEIN: So to check what I have, I do the hyphen v. You might be familiar with a verbose, hyphen v. So I have this, it looks spelled correctly. Cool. Now I push to it, like take my changes and push them, to "origin", which is the name I've given to this URL. "Master" is the master branch. It's going to ask me for my password, because I haven't configured SSH correctly, but Bitbucket has instructions for setting this up so that your computers recognize each other. You don't need to type in the password every time. I just haven't done it yet. OK, now it looks like I successfully wrote things to the server. So Max should be able to get them. Oh, no, he shouldn't be able to, because my repo is private, and nobody's allowed to access it but me. Let me change that. Settings, access management.

MAXWELL MANN: So I'm M4X Mann.

AARON EPSTEIN: And I want to give him read and write, but I don't trust him to administer it.

MAXWELL MANN: It's a good idea. I have a certain tendency of screwing up certain administration jobs, this one not included. So I'm going to go ahead and switch on over to Me. Now I'm me. I wasn't before, but I am now. Let's make some space here. On the left, this is where I have Battlecode installed, on the desktop. And I guess what I want to do is I want to make this existing Battlecode directory the main directory for the root for my Github version of the thing that he has. Yeah, that's the technical term. So I'm going to go into it. So now I'm in this directory. And what I'm going to do is I'm going to initialize GIt so that now it knows that I'm interested in using Git in this folder. And now that I'm in it, I will do Git remote add. And I'll go onto this website, I'm just going to hit refresh here. Is this it, "Battlecode-example"?

AARON EPSTEIN: Yeah.

MAXWELL MANN: OK, so you named it the same thing as you named the previous one?

AARON EPSTEIN: Yep.

MAXWELL MANN: OK. So I'm in here. And I can just right click and hit Paste because I have PowerShell. So now I've now I've added it. And so I'll then follow that up with a "git pull" . And that's going to give me-- oh, it says it's not my-- I need a name?

AARON EPSTEIN: Yeah. You didn't name it. Yeah, you just did Git remote add, and you just put the URL.

MAXWELL MANN: OK, I'll name it origin.

AARON EPSTEIN: No, the other way around.

MAXWELL MANN: OK. So I'll name it-- what do I name it?

AARON EPSTEIN: Origin, and then the URL.

MAXWELL MANN: Add origin, and then and then the URL. We'll have this all written down so you have the instructions in front of you when you do it.

AARON EPSTEIN: Now you should be able to pull.

MAXWELL MANN: OK so now I can Git pull. And this should get me the stuff he wanted to give me. Now it may end up being a problem--

AARON EPSTEIN: No, you've got to specifically from origin master.

MAXWELL MANN: Ah, git pull, origin master. Yeah. For those of you who know Git already, I think it may be giving you trouble because I may already have awesomer player. so we're going to pretend that I don't have it. Yeah, so let's go let's go back to Eclipse. Oh, it's black. What is that? I think Eclipse. is very unhappy. Have you ever had this? OK, where's the package--

AARON EPSTEIN: There are Windows, so we should probably just restart both computers.

MAXWELL MANN: Restart both computers?

AARON EPSTEIN: Yeah, and every Windows computer in the room.

MAXWELL MANN: Oh, right. Now let's see, which is the button that gets me the package? Package inspector window? Open perspective? Show view, OK. I've got. Hey look, it's not dead. So let's refresh it and now it's updating. You see it hasn't got awesomer player there, so that when I do this. Yeah, that wrapped around. But now I've done this, if I push that F5, is that going to-- No? Did it not work? It says here, already up to date. I think we broke it. What are we gonna do, Aaron?

AARON EPSTEIN: I don't know. I don't know what happened. So you don't have awesomer? Do you have awesomer?

MAXWELL MANN: I may have awesomer without knowing I have awesomer. I'm that awesome. Nope, there's nothing in there. Should we kill everything and start over?

AARON EPSTEIN: Yep.

MAXWELL MANN: Right. Doing it the right way. You saw it here first.

AARON EPSTEIN: OK, so Bitbucket tells me--

MAXWELL MANN: OK, I've got an idea. Make a new package and send that one. Something else. Something different. OK, let's go back to Aaron's computer. So you made a a super thing?

AARON EPSTEIN: A super awesome player.

MAXWELL MANN: A super awesome player. Oh my goodness. So now you're pushing that onto the master, and then once that's done, I'm going to go back to me, right? OK. SO I'm going back to my Battlecode folder. OK. And now it's already got Git in it. I'm already good to go. I've set up the remote by doing Git, remote, add, origin and then the HTTPS URL that was available on the Bitbucket website.

AARON EPSTEIN: Well, why don't you Git status. Just to check.

MAXWELL MANN: I like that idea. OK, so I deleted a thing. Maybe if I just push the fact that I deleted that thing.

AARON EPSTEIN: Yeah, maybe that was a bad idea.

MAXWELL MANN: Well in any case, we're going to see what happens. Maybe if I add, so that it's tracking the thing that isn't there, then that'll improve matters.

AARON EPSTEIN: Let me just make the reaper. Oh no, but then you'd have to type in the URL again.

MAXWELL MANN: Yeah, I could just paste, but let's do Git pull. And let's see if it works, shall we?

AARON EPSTEIN: Nothing to lose.

MAXWELL MANN: Only our dignity.

AARON EPSTEIN: I'm going to pull from the origin master specifically.

MAXWELL MANN: Well, what happens if I just do it as it is? It's no good?

AARON EPSTEIN: You've got to configure it before you do that.

MAXWELL MANN: It said, unpacking objects. I think it did things.

AARON EPSTEIN: Well it unpacked a bunch of objects, but it didn't know where to put them, I think.

MAXWELL MANN: It didn't know where to put them. Well, I put them somewhere now. It says, one file changed. It's fussy? I'm fussy. We work together. OK. Now the thing is there. And if we wait 10 minutes, Eclipse will start. But while I'm doing that, you could talk about the next thing. Yay, there it is. Super awesome. Yeah, that's pretty cool.

AARON EPSTEIN: You should watch it. It's super good.

MAXWELL MANN: I should to watch it? Is it the same as the other one?

AARON EPSTEIN: No, it's mine. It's my special one.

MAXWELL MANN: Oh, it's your special one? Oh my goodness, you guys are in for real treat. Aaron is a coding beast, I gotta tell you. This is the representation of his might and his capability, right here. Super awesome is the red fellow and there he is, going, capturing things, defusing mines. [LAUGHTER]

AARON EPSTEIN: Oh, it didn't work. Oh well.

MAXWELL MANN: Sorry, Aaron. Next time.

AARON EPSTEIN: It's just a small map. So you could rush, strategy, rush OP, et cetera.

MAXWELL MANN: I'll tell you what, we'll change the game specs. We'll change the game engine.

AARON EPSTEIN: I've already done it.

MAXWELL MANN: Oh, OK. That makes sense. I just didn't get the latest version. The "Aaron wins all matches" version.

AARON EPSTEIN: Yep. OK, anyway-- What?

AUDIENCE: Why don't you change your team name to Aaron?

AARON EPSTEIN: From what? What do you mean? Sorry, I don't understand.

MAXWELL MANN: It's OK. It's got to do with corruption. I'll teach you sometime.

AARON EPSTEIN: OK. Yeah, I don't know anything about, of course. So I did Git status, which is always a good thing to do. Between every command that you execute, that you ask Git to do, before you should do a status, and after you should do a status. Which means that since I wanted to execute status to show you, I should have done Git status three times.

One before the Git status, one to execute the Git status, and one after. Just so that you get a really good feeling for what the status is. And you should actually read it a bunch of times. So Git status, that's the command I type. On branch master, master's a branch. Got it.

MAXWELL MANN: We'll do branches in the lab.

AARON EPSTEIN: Yeah. Untracked files. OK, that's a lot of untracked files. Nothing added to commit, but untracked files present. OK, nothing added to commit, means I haven't done anything. But there's untracked files. That's bad. When I tell you status is really important, I mean it.

But if all this red stuff comes up every time you type status, you're going to start not paying attention to it. So an important thing to do is to tell Git to ignore these things that I never want to add. So I just want them to be ignored. So what I do is I open up my favorite text editor eMags, and I go to directory, and I make a file called dot Git ignore. And I no longer exist. OK. Oh man, eMags is too clever sometimes.

MAXWELL MANN: Because he made this file before the lecture in preparation, and then deleted it. But eMags was like, are you sure you didn't want to not have the things you deleted and then looked for later in a different place? And then he said, yes.

AARON EPSTEIN: Now it's empty. OK, cool. So I have this empty file, and to make it ignore this dot installation information, which I'm never going to add to the repo, I just type in dot installation information. And then to make it ignore everything in the doc folder, because I'm never going to have to share that with Max, because he has from the installer, do doc slash star, and then version dot text, et cetera. Or I get this handy-dandy one that I've already written, and just copy all of it.

Yeah. So now that I've made the dot Git ignore and saved it, now the only untracked things are the other teams. So I could specifically add those. Teams slash example, teams slash macro player, team slash suicide player.

MAXWELL MANN: Yet another one of his classics.

AARON EPSTEIN: Yeah, right. Save that, Git status. The thing with the squiggly is just another eMags Folder. I'll just ignore everything that ends in squiggles. Cool, so now the only thing that's untracked is a dot Git ignore. So I'll just add the dot Git ignore. Because I do want to share that with Max so he doesn't to go through all that trouble I just went through. Now if I do status I get this really nice-- oh, no. Now to commit the fact that I've added the Git ignore.

I do the AM switch again. A for everything in the index. And m for not bringing up VIM and pulling my hair out. Make a nice commit message so Max knows what this file's for. OK, now if I do status I get this really clean status. It even says that my working directory's clean, nothing to commit. Perfect. So I'll push that to Max and he can get it, if he wants. Type my password again, 123ABCD.

MAXWELL MANN: So I noticed that we untracked a bunch of things because we didn't want to include them. But I was saying before this that it might be convenient for us to just share the whole of everything. Why wouldn't we do that?

AARON EPSTEIN: OK, there's a lot of reasons not to do it. A couple things you don't want in your Git repo. Anything that's platform specific, probably you don't want in your Git repo. So right now I'm using Windows and Max is using Windows, but Steven doesn't use Windows. So anything that's on our team-- if I have a path that is something like C backslash users, that's going to look like nonsense once Stephen checks it out. So anything platform specific doesn't go in the repo.

Another thing you don't want in the repo is stuff that we distribute. By we, I mean we as Battlecode people, not we as in this false Battlecode team that has the repo. So we as Battlecode people release an installer that gives you things like example [? flux ?] player, and documents, and stuff like that. You don't want those in the repo because, let's say I add example [? flux ?] player to the repo. I check it in-- now, this is I as in this guy-- I check in example [? flux ?] player. Max is working on really complex navigation code. I have version 1.3, and Max has version 1.0. An example [? flux ?] player, the one I just checked in, makes a 1.3 specific function call that breaks if you use it in 1.0. So I check it in, I upload it. Max downloads my code expecting to get some more navigation stuff that he really needs.

But he also gets this new version of example [? flux ?] player that makes a 1.3 specific function call, and now his code won't compile at all. And now he has to interrupt the workflow that he was really deep into navigation, and isolate where is this red X coming from, and find out that it's because it was in a different version. Then he has to go download it from Battlecode.org, and then install it again, and then update, and then find that the update doesn't work on his computer. Then he has to fix that. All of this, and then in the process, forgetting everything he knew about navigation.

So when he gets back to, he's already forgot. It's just a pain. So anything that we update, just get it from the installer. Don't put in the repo. There's no need.

Anything binary, don't put that in the repo, because it's inefficient. Git has really good algorithms for compressing and storing only meaningful things about text, so if there's a huge one megabyte text file and I change one line, Git stores one line. None of those algorithms that it has applied to binaries. So if I upload a binary, and I change one line in a binary-- a binary meaning a zip archive or a picture. If I change one pixel in the image, Git just has to store both copies of the image, which is inefficient.

So if you store our whole installer, it's like 30 megabytes. Every time we update, you'll add another 30 megabytes. So instead of this 500 kilobyte repo that Max and I have right now, if you kept going like 30 megabytes per installer, you'd end up with 500 megabytes. Bad. Not good. And then, anything that's personal, like project settings you don't need them.

So what do you put in the repo? Text documents that you've created, like source code, or new maps that you want, or a script. Or all the incredible documentation that you're going to have for your players, all of that, that can go in the repo. Cool.

MAXWELL MANN: In some past years, we've offered another one of these side prizes for best documented code. Are we going to be doing that this year, Stephen? Think a best documented code side prize this year?

AUDIENCE: Probably

MAXWELL MANN: Probably. Something to keep in mind. Keep those forward slashes on your left ring finger, which is to say, ready. Do you hit it with your pinky finger?

AARON EPSTEIN: Mine is on the left bracket, because I use Dvorak.

MAXWELL MANN: Right. But it's still the same finger.

AARON EPSTEIN: No, it's the pinky. How do you go all the way over there with the ring finger?

MAXWELL MANN: I think I was mistaken.

AARON EPSTEIN: I think-- yes, it is the same finger. Yeah, pinky. Pinky active coding. Cool. So then we just showed you an example where I made a repo and put some files in it and push to server. Max made his own repo, pulled some stuff. He got it. He could do the same thing, where he can add files and push it, and I can get them. And that's great. But now we have to show you what happens if Max and I both try and change one thing, the same thing. Then we get a conflict. It's really scary, and the reason why-- or, it's a reason that some people give up on Git. But we're going to navigate it. It's going to be OK.

MAXWELL MANN: OK, let's do this.

AARON EPSTEIN: OK, so let's go to my super awesome player. I have this number capturing channel here. I think it's not big enough. So I think you should make it bigger.

MAXWELL MANN: Well, I think it's not well commented enough.

AARON EPSTEIN: OK, I'll make it bigger, but you try and explain it. Which is silly because you don't know what it's for. But I'll make it bigger, because I think that's important. It's a radio channel, so in case you didn't get it, making it bigger's not going to change anything. But anyway, I'll pretend.

MAXWELL MANN: Okay, I've saved my changes.

AARON EPSTEIN: OK so I did a status, because I'm about to execute commit. So first I did a status. Yeah, you're getting the hang of this? Now I'm going to do a commit, because modified, that's what I wanted. I wanted this player modified. Do a nice commit. OK. I would use an exclamation mark here, but no exclamation marks in Git commit messages because bash handles them correctly. So they can't have exclamation marks in them.

MAXWELL MANN: Alright, I've pushed my excellent change. I hope you learned something from it. If I may say so, it's a wonderful comment. I think that you really couldn't expect much better.

AARON EPSTEIN: Alright, I'm also pushing my change, but wait. It didn't work. Why? Updates were rejected because the tip of the branch is behind remote counterpart pull. OK, I think what that means is that because Max pushed and I tried to push to the same place, Git doesn't like that because then it wouldn't know which is mostly recent version. So it tells to resolve it by pull.

A lot of commands, when they give an error, will tell you what command to fix it with. So because I'm about to do a pull, first I'm going to do a status. OK, that's cool. And I'm going to pull. 123ABCDE. Oh no. Conflict. Look at that, all caps. Automatic merge failed, fix conflicts and commit the result. So team super awesome robot player dot java had a conflict.

MAXWELL MANN: So Aaron, I guess it's up to you now to fix that conflict.

AARON EPSTEIN: Yeah, it is. So I go to robot player dot Java.

MAXWELL MANN: Oh man, why don't you do window preferences, and change your size to something non-zero.

AARON EPSTEIN: Aw man, but I'm so excited about fixing the conflict. I just want to fix the conflict though. Is this going to fix the conflict?

MAXWELL MANN: Appearance. Some conflicts have been going on for thousands of years, I think it can wait. Yeah, then you go to Java. And it's only like another 16 Windows. Text editor font.

AARON EPSTEIN: This is terrible.

MAXWELL MANN: I know. I have to do it every lecture. 16 is a good number. Let's make it bigger, because you have better resolution. His computer's a beast. It's a pretty darn nice computer.

AARON EPSTEIN: OK cool. Alright, I've got tons of red. It didn't work. OK, so the conflict markers are as obvious as they can be. Really big, loud, not part of any programming language. Obviously errors that you want to clean up. Nice x's. So what this is, head is the head of my current stuff, so that's going to be my copy. And then this really long number here is the hash name of Max's commit. So this is Max's thing. Max made a comment, and I changed the number.

And so even though I know that all of my changes are going to be superior to anything that Max wrote, Git doesn't know that. So since it's conflicted, it asked me to fix it. In this case, I'll take what Max wrote and add it to mine. And then delete it. Not the other way around, by the way.

MAXWELL MANN: Everybody's got to have standards.

AARON EPSTEIN: Yeah. Oh, I didn't delete that. Cool, so now it compiles. Probably still doesn't beat the stupid [? rush ?] player OP stuff. [INAUDIBLE] It compiles, and now I have to tell Git about the success. So I do a commit, but before I do the commit-- yeah? Getting it yet? Oh, it deleted awesomer player. That's what Max did on his computer.

And we both modified robot player dot java. Pointing with my mouse, remember. That's cool, but I fixed it. So, "merge a success. Cake imminent." Is that the right spelling?

MAXWELL MANN: It's close.

AARON EPSTEIN: OK. And then I push that.

MAXWELL MANN: I'm going to get it. Git pull origin master. I'm ready to push this button. And-- yeah. Cool,

AARON EPSTEIN: So that's a conflict resolution, which happens more than is pleasant, but is not too hard to deal with once you know you're doing.

MAXWELL MANN: If we had edited different lines of the same file, there'd be no conflict because Github is so smart about which lines were changed and which weren't. It would just go ahead and it would merge those automatically.

AARON EPSTEIN: Alright, so before Max takes my changes and starts messing around with them, I'm going to just mention, but not thoroughly explain, a bunch of super useful Git commands. If this wasn't enough, then if you go look up these commands and what they're for, then you'll be convinced that Git is essential.

Git checkout. Git log. Let me do this. If I show you the log, which is something I forgot to do, if you do a status before and after each command, after you're done with the second status, do a log. And log tells you what's happened. Pointing with the mouse. OK, right? This is the whole history of what happened. So Git checkout lets you take any of these points in time and set your repository, like you're working copy, the copy on disk, to what was here. So if I want to go to what happened before the bigger numbers commit, I could check this out, and do Git checkout.

Git branch. Branches lets you have parallel workflows. It's really smooth. I have this document that I found online that describes a really good, really useful way of using branches. I'll discuss in the lab.

Git reset hard. So, commit is like a way of saying, these changes are great. They're solid. I like them, keep them. Anything you haven't yet commited can be obliterated by reset hard. So you accidentally RMRF one of your players. Or you make some change that actually causes a divide by zero, and you have no idea how, and you've spent half an hour looking for. You've just given up and you just want it gone. Git reset hard just obliterates anything that's not committed.

Stash does the same thing, but puts it somewhere. And then you can take it back off with stash pop.

Git k summons a GUI visualizer of the entire history. I'll just go ahead and summon it. There's our history, there's some diffs, what was changed. This was the merge, how it looks. Great for visualizing stuff if you're not, or even if you are good at reading the terminal.

Other commands. Diff shows you the difference between two commits. Like if I added comment to message channel, what? I don't remember that. What comment was it? I could do Git diff between 31d and the thing before it. Oh, 31d was ambiguous? One six. 490 was ambiguous.

MAXWELL MANN: Make it E36.

AARON EPSTEIN: Cool. Oh, that's where I deleted awesome player. It looks like you added awesome player, which means I probably specified in the wrong order. Yeah, so that's a comment. Cool. That's diff.

Revert makes the inverse of a commit, if you want to get rid of one.

Bisect lets you do a binary search through your history to find a bug. If you find out that something-- you find out some cache you wrote three weeks ago doesn't work anymore. It worked when you wrote, but it doesn't work anymore, but you haven't worked on it at all. So it's clearly something caused by something else. It doesn't work this commit, it doesn't work the commit before, so something you wrote somewhere in the last three weeks broke your cache system and you have no idea where it was.

Git bisect lets you binary search through your commit history to pinpoint the location of the bug. Look it up, it's too complicated to be explained in 30 seconds. But it saved me a lot of time once, where I was looking a bug for like four hours. I gave up. Bisect let me find it in 30 minutes, and-- you know, programmer time. I hadn't used bisect, I really would have just had to rewrite sections of the engine. There was no way I was going to find it. It was something really dumb.

Blame does what it sounds like. So if I do get blame on-- it tells me who was responsible for every line. And the evidence is clear. The only person I can claim blame for super awesome players loss is myself. There's nothing here except me. It's all me. If Mad had changed some line, like this one or whatever, if it wasn't checking this encampment thing right, and it would have said Max man. And I could have tracked it down. But no, it looks like this file is all me. It's the only person I can blame for my defeat. But anyway, blame. Blame people. It's great.

MAXWELL MANN: That's a pretty good summary. It seems like you've talked about quite a lot of functions.

AARON EPSTEIN: No praising, only blaming. I actually think there is a less negative version of blame that does the same thing, but use blame. OK.

MAXWELL MANN: So you've covered what you need to cover. We've shown you examples and things. We've given you summaries. I'm sure your mouths are watering already. I want to give a little bit of a demo, because the sprint tournament is coming up so soon. The sprint tournament submission deadline is Monday. So this is how you would submit.

So if you haven't submitted already, you right click on the robot you want to submit. And you go to export. And I think the instructions for this online. Yeah, we're going to switch to my computer. My goodness, that makes a lot of sense. So here we are, I'm going to right click on the robot player that I want to submit. This is the package. I export, archive file. And I have to name it my team. So I need to actually change the package name to team, and then my team number in three digits. Otherwise it doesn't work.

And then I'll just export it to an archive file. This would export to the desktop. And then later I can go on to battle code, and I can say, alright, upload my player. And I'll be able to choose the JAR file from the desktop where I've got it. And I can give it my own name and then I can submit it, and it'll tell me whether it was compiled and worked successfully.

So that's a complete introduction, an entirely complete explanation of Git. Everything you ever needed to know, except for the things we're going to go over in lab. Aaron's shaking his head. There's a lot to learn, and it only makes your life easier. It only helps. Except when it doesn't. So with that, the next lecture is going to be on swarm, and I'll also cover some topics in debugging. So you're going to watch robots dancing around in very mellifluous-- is that the right word? In any case, they're going to dance in fancy ways. Fancy pants dance. And that'll take care of it. So thank you for coming to this lecture. I hope you enjoy the Thai food, and I'll see you tomorrow.