Grokking Git: A Fantasy Story

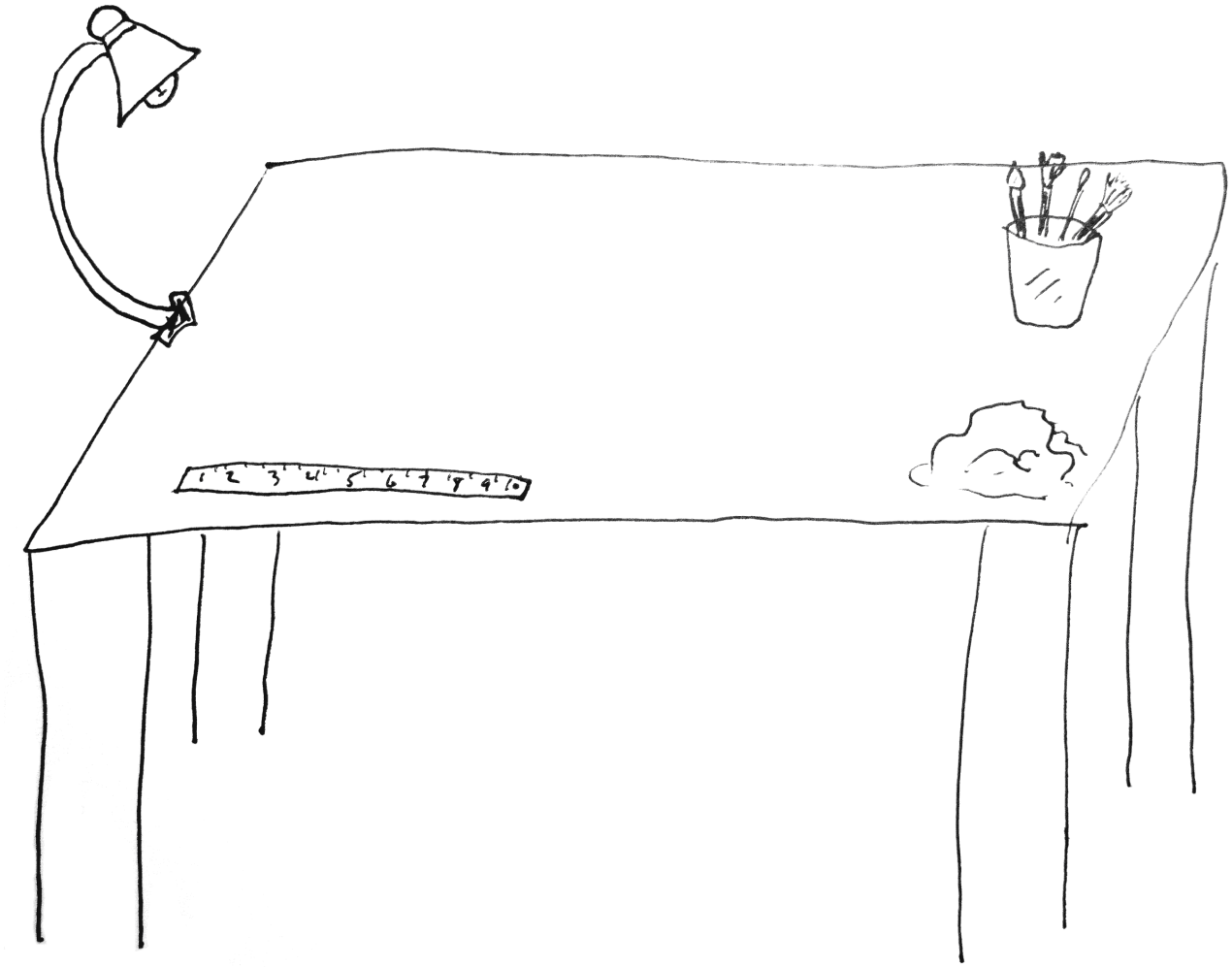
Git is a big part of a lot of tech worker's every day lives, but it can be confusing to use. There's good news and bad news.

The bad news is, Git is complex; there's a lot of confusing commands.

The good news is, that the way Git is designed under the hood makes more sense than its interface.

We're not going to learn any command incantations, and we're not going to learn recipes on how to do specific things. We're going to explore *why* Git works the way it does. We're going to learn this, through a fantasy story.

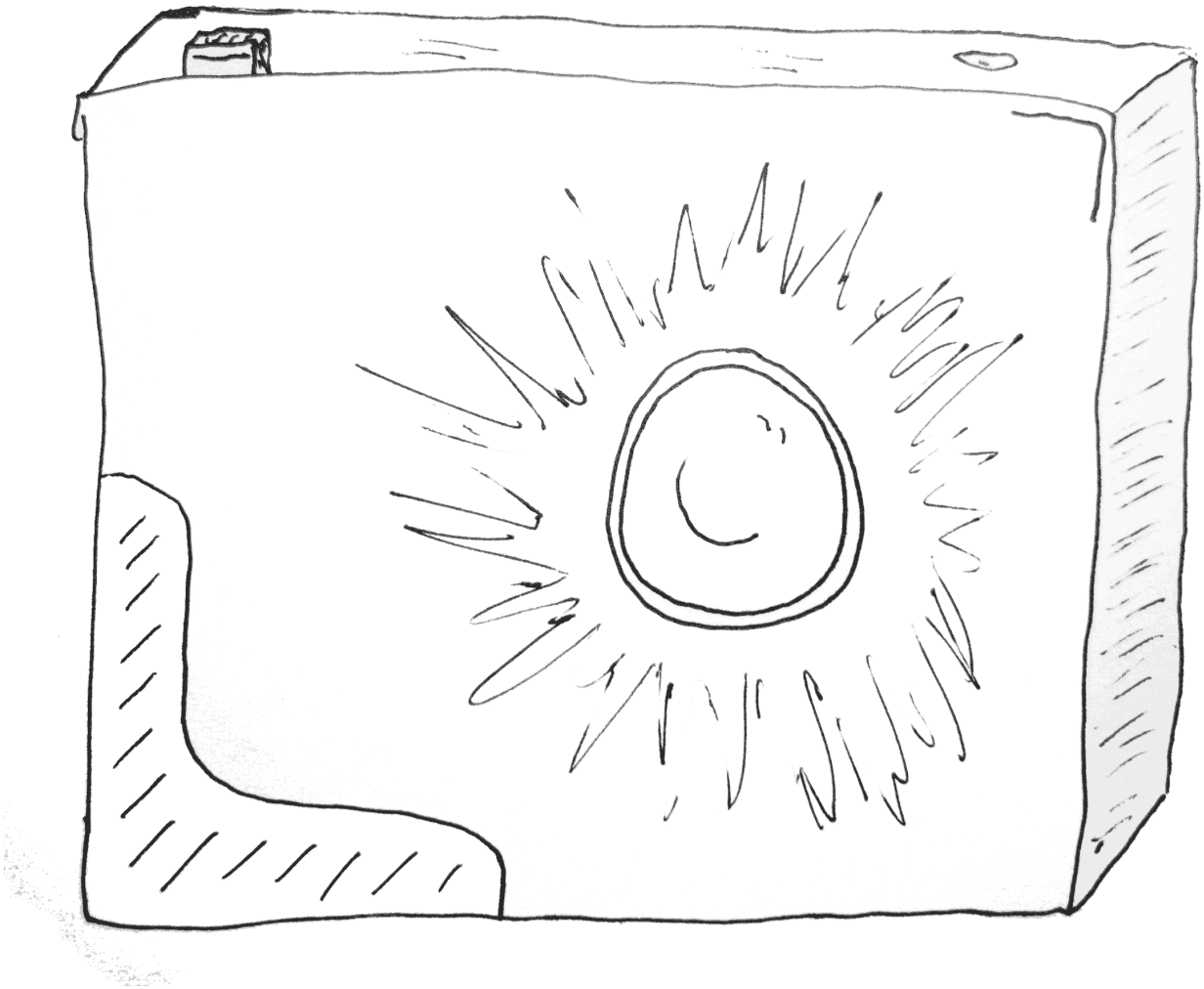
— Snapshots —



Imagine you’re a sculptor. You mold esoteric critters out of clay. You’re working on a huge diorama with lots of different characters for a mysterious donor.

One day, you're working on a tricky critter. You get the nose right, but then the ears are wrong. You fix the ears, but now the hands don't make sense. You wish you could go back and see all the different versions of the critter. Then, you could get everything looking how you want.

Julia, your mail carrier, pays you a visit. She asks about your critter. "It's not going well," you say, despondent, and you tell her your problem with the different versions. "You are in luck," She says, "Because you have a special delivery." She takes a gleaming camera out of her pack.

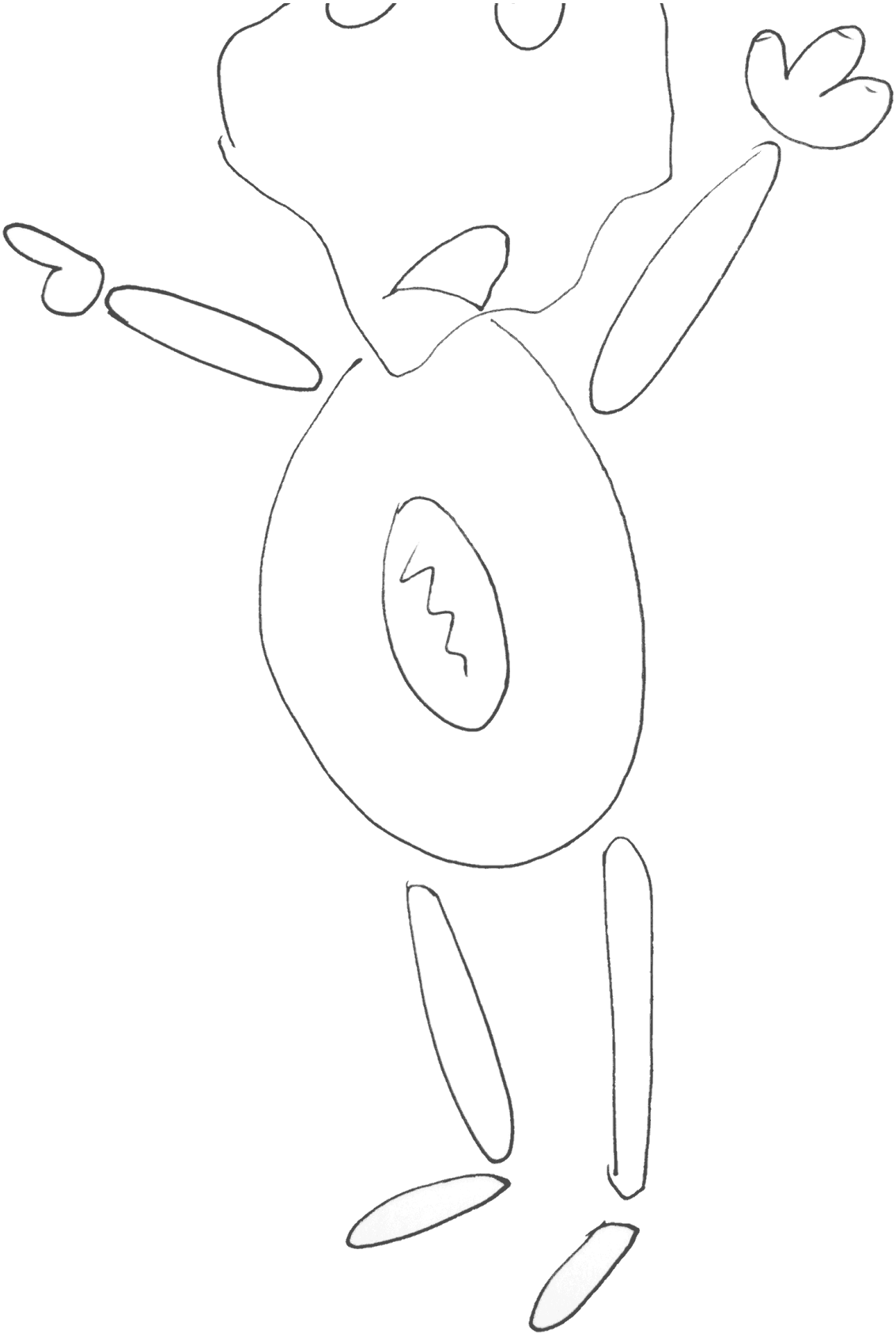


“Thanks...” you reply, confused, “but how does that help me?”

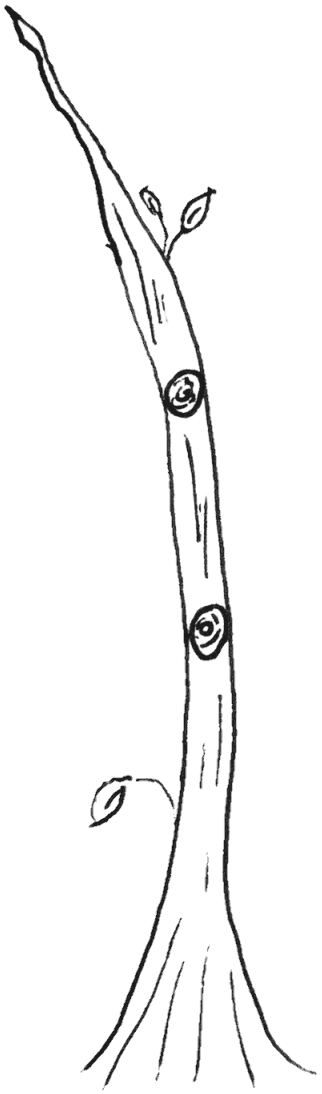
“Oh,” she says, “The camera is magical. See that little tree out in your yard? Every time you take a picture of the art on your workbench, the tree will grow a knot. When you say the magic words over a knot, the art on your workbench perfectly transforms your piece into how it looked in the photo. Say some more magic words, and it goes back to how you last had it. Pretty neat, huh?”

“Whoa, that’s incredible!” You cry. “How do you know all this?”

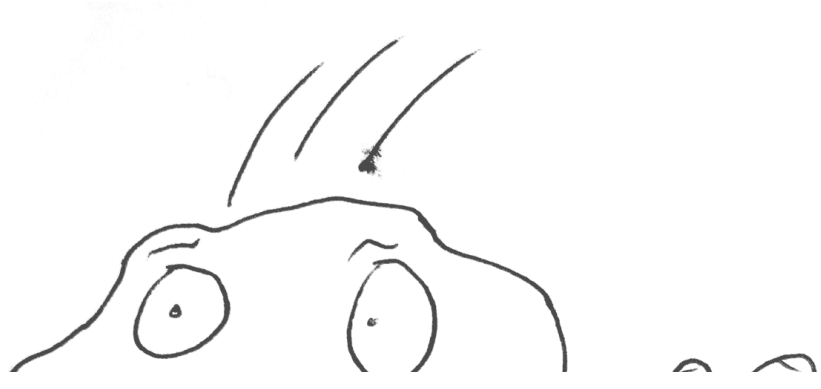
But Julia is gone.

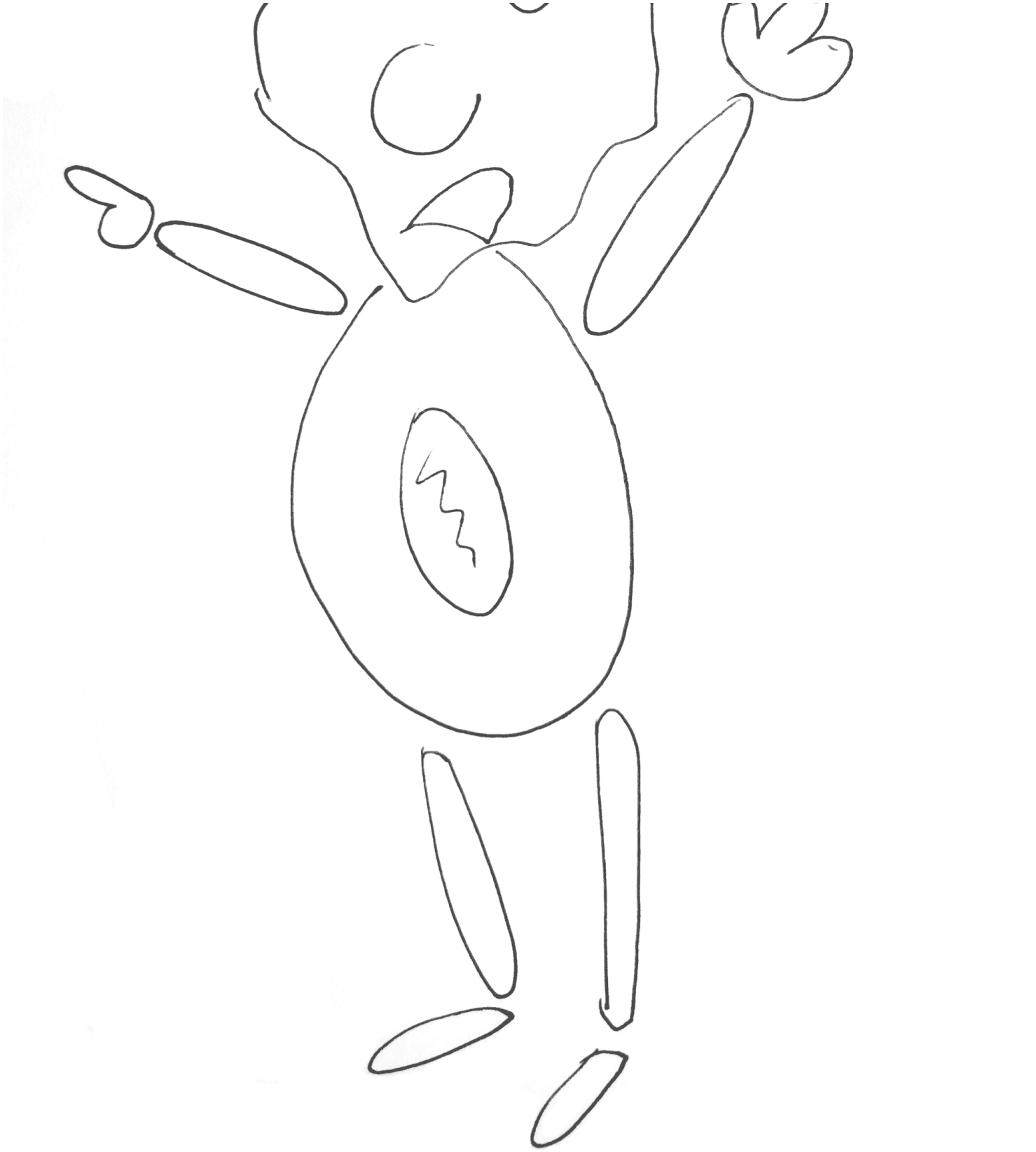


Excited, you use your new magical camera to take a picture of your critter sculpture. You lob off its nose, and take another photo.



You watch as your tree grows two knots, one after the other. You say the magic words over the first knot, and *voilà!*





Your critter has its old nose back.

These magic photos are like Git commits. Each one captures your work in a given state. Once a state is captured, you can always retrieve it.

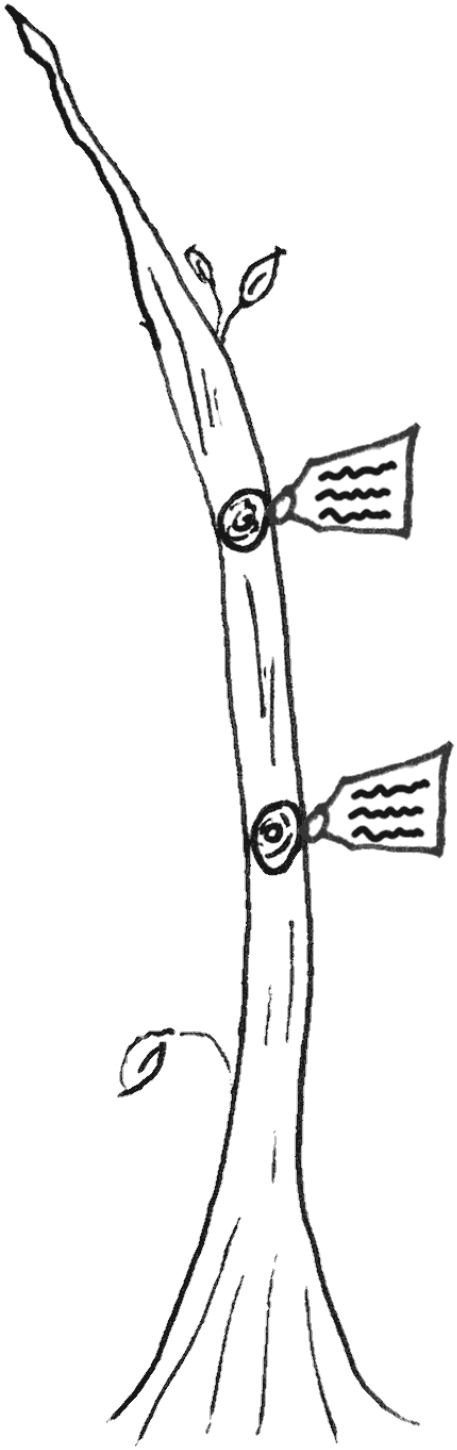
The tree growing knots for every picture is like Git history. Like the tree growing knots one after the other, Git keeps track of when you made each commit.

```
git switch --detach HEAD~4 # 4 commits ago!
git switch - # back to where you where!
```

— Messages —

You keep working, and taking photos. The pictures help you get each critter just right. You end up taking *a lot* of photos. It's hard to keep track of all the knots on the tree!

You have a plan.



You tie a small piece of paper to each knot, with a note about what you changed. You write messages like, “Remove an eyebrow”, “Add a bone necklace,” and “Increase size of lip wart.”

For complicated changes, you leave a whole paragraph or more of notes to yourself about why you made the change you did. Now, when you look at your tree, you can quickly get a sense of your work over time.

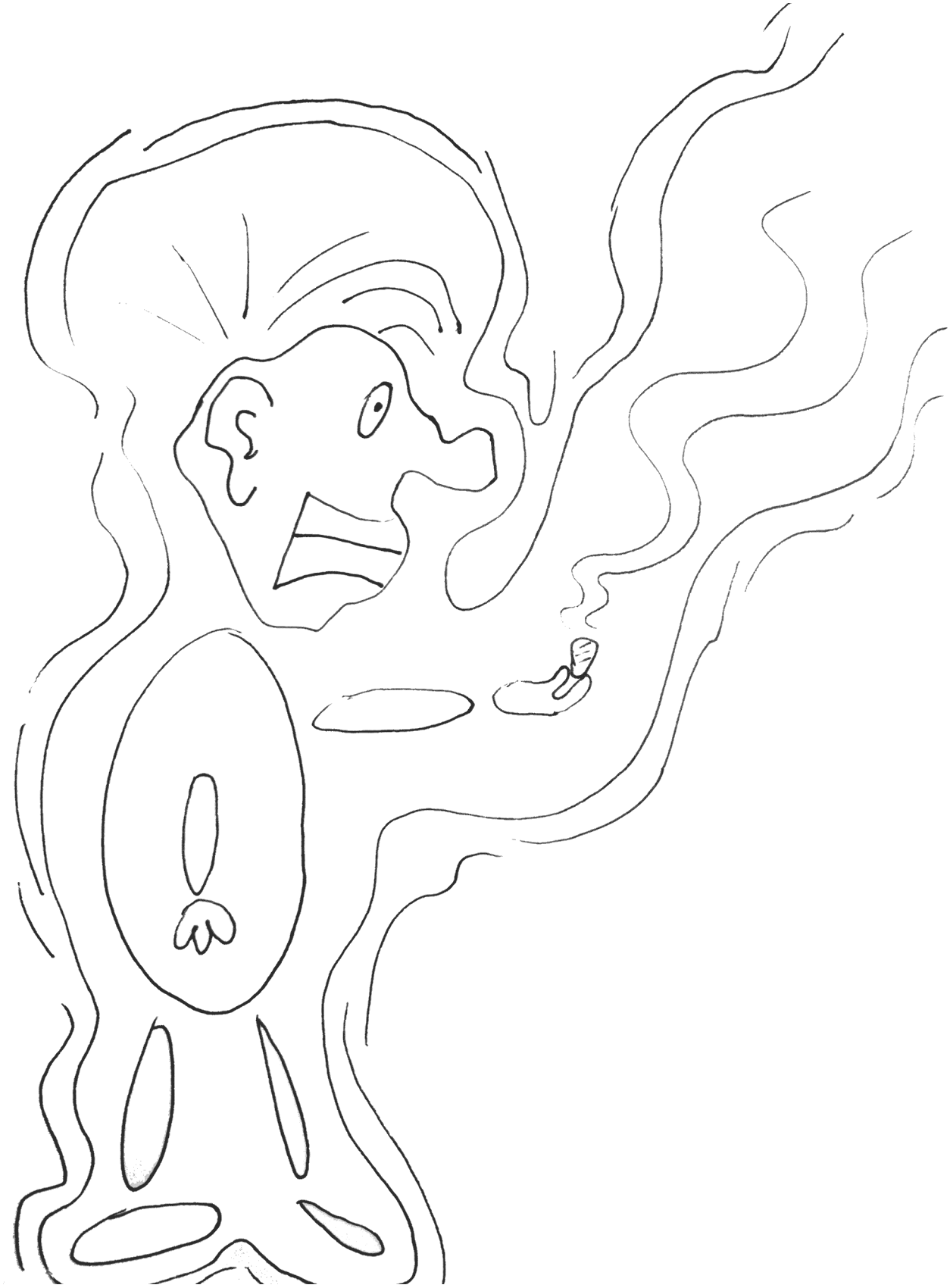
```
git commit --message "My message" --message "My extended explanation." # Or, use -m i
```

The notes tied to the knots are like Git's commit messages. A commit message is always associated with its commit. They provide invaluable context about the change you made.

Inspecting work in progress

You get very focused on your work, changing lots of different things over the course of the day. When you're ready to take your magic picture, you have a hard time remembering what changes you made.

On a whim, you look through your magic camera at your workbench.



You see a glowing red aura around the critters you changed. Neat!

Peering more closely, you can see *how* you thickened a critter's eyebrows. Super neat!

Now you know exactly what you've changed since your last commit.

The red aura is like using Git status. This shows which files you've made changes to since your last commit.

```
git status
```

Seeing the changed eyebrows is like running Git diff, which shows what additions, deletions, or modifications you've made to the files you've changed since your last commit.

```
git diff
```

These are very useful ways to inspect the work you've done.

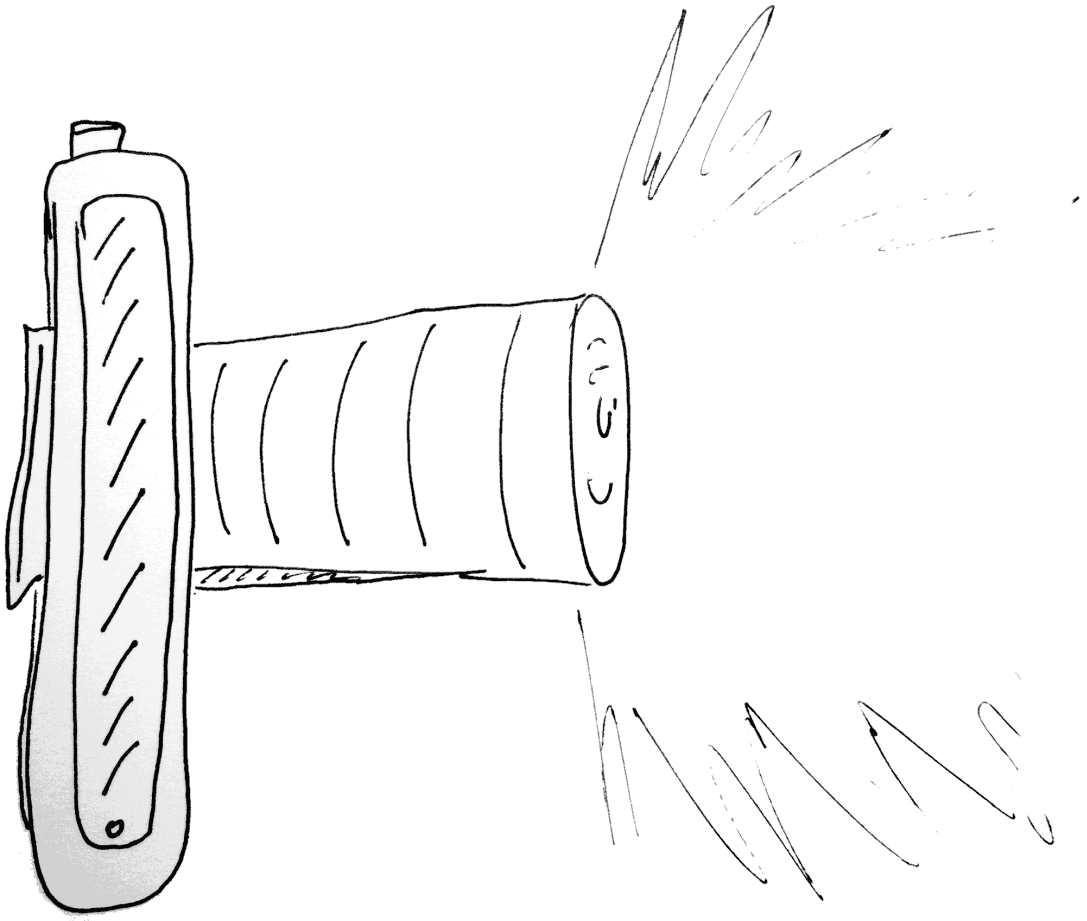
— Staging —

While you're working on a critter chasing a butterfly, an idea strikes you for a different critter smelling a stinky cabbage. You sculpt that, then return to tweak your butterfly critter.

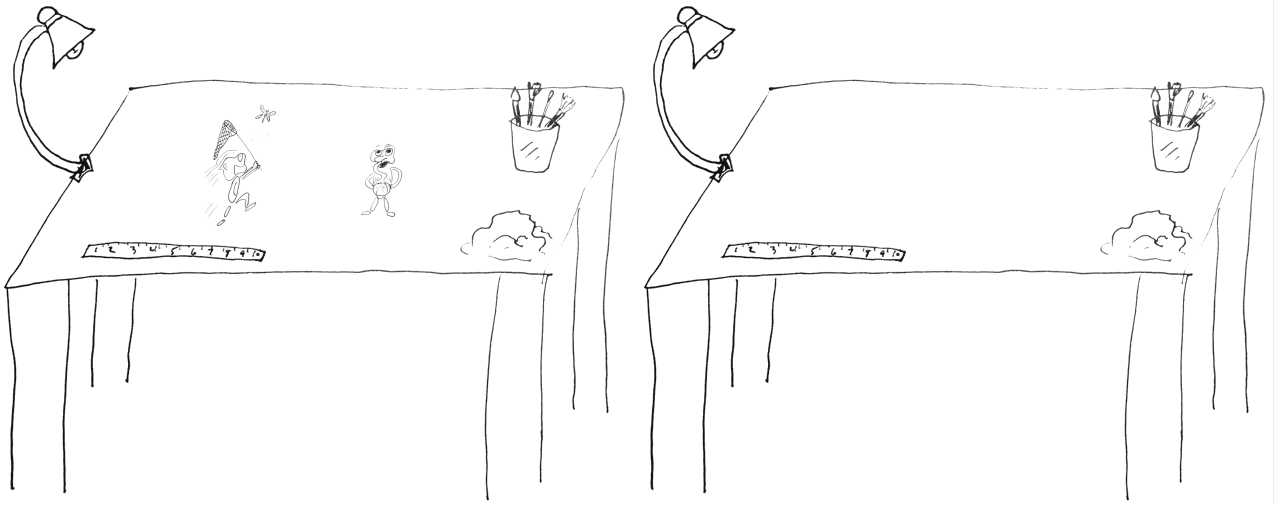
Now you have a dilemma. You'd like to keep your commits organized for easy reference, but the changes you've made are to two separate parts of the scene. If you take a picture, the changes to both critters will be in one commit. You'd like to have those critters as two separate commit knots on your tree trunk.

That's when you notice a toggle on your magical camera that says "staging." You switch it on, and two things happen simultaneously.

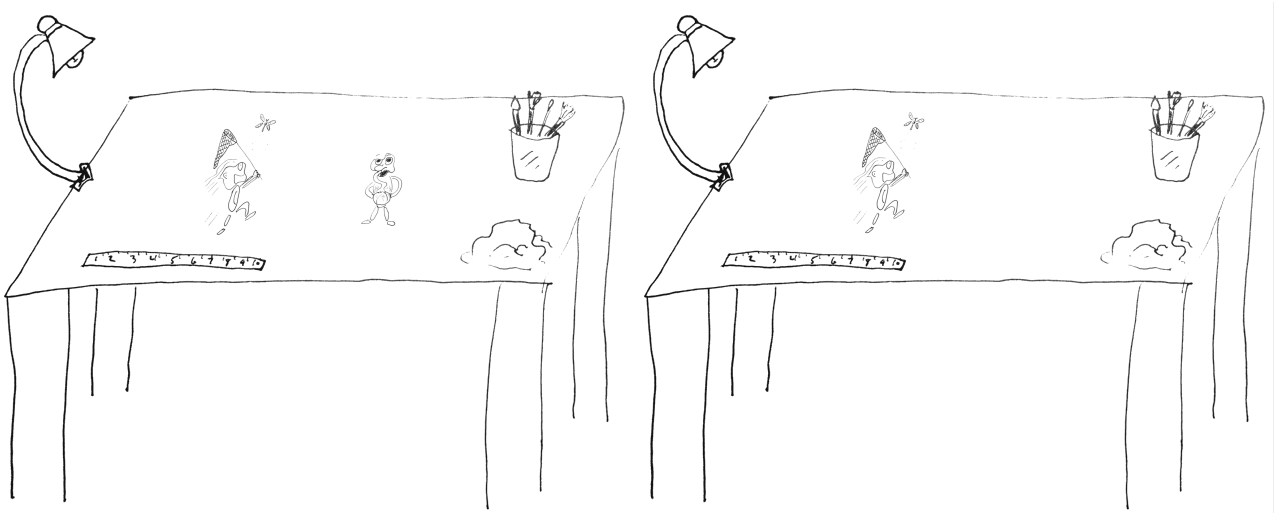
The camera morphs; it now has a long zoom lens.



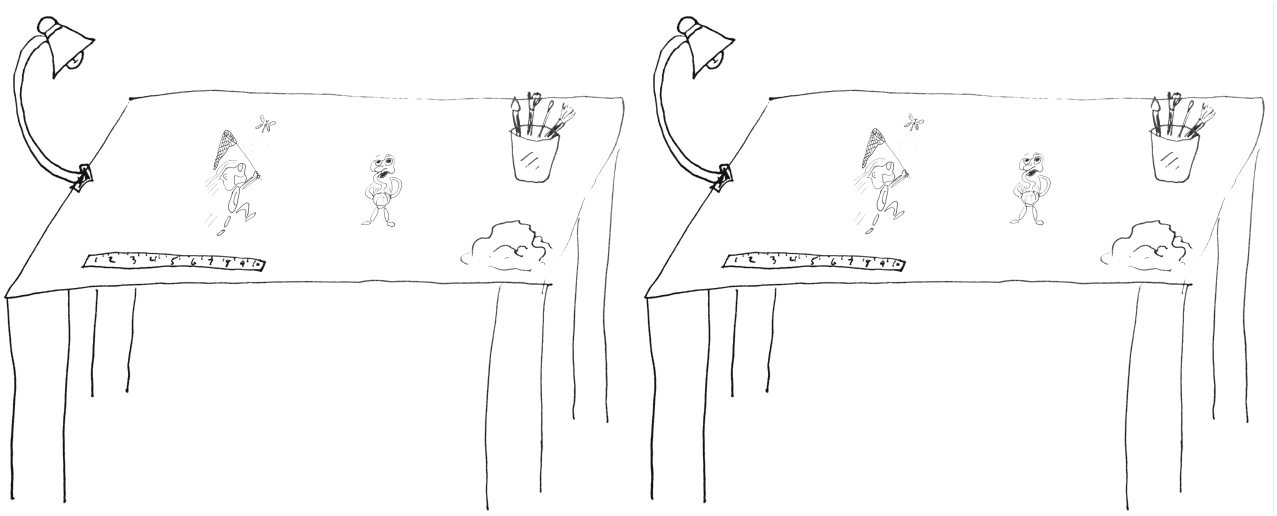
A perfect copy of your workbench appears next to it, only it’s empty.



You play with the new lens, snapping a zoomed-in shot of the butterfly chasing critter, and it is perfectly duplicated in your second workbench.



You take a picture of your second workbench, a new knot grows on your tree with the butterfly chasing critter changes. You do the same with the cabbage sniffer.



There! Everything is organized.

The “staging” mode on the magical camera, which creates the second workbench, is like the staging area in Git.

```
git add path/to/file
```

Git also calls the staging area the index and the cache. Fun.

Whenever you make changes, you have the opportunity to choose precisely what will go into your next commit.

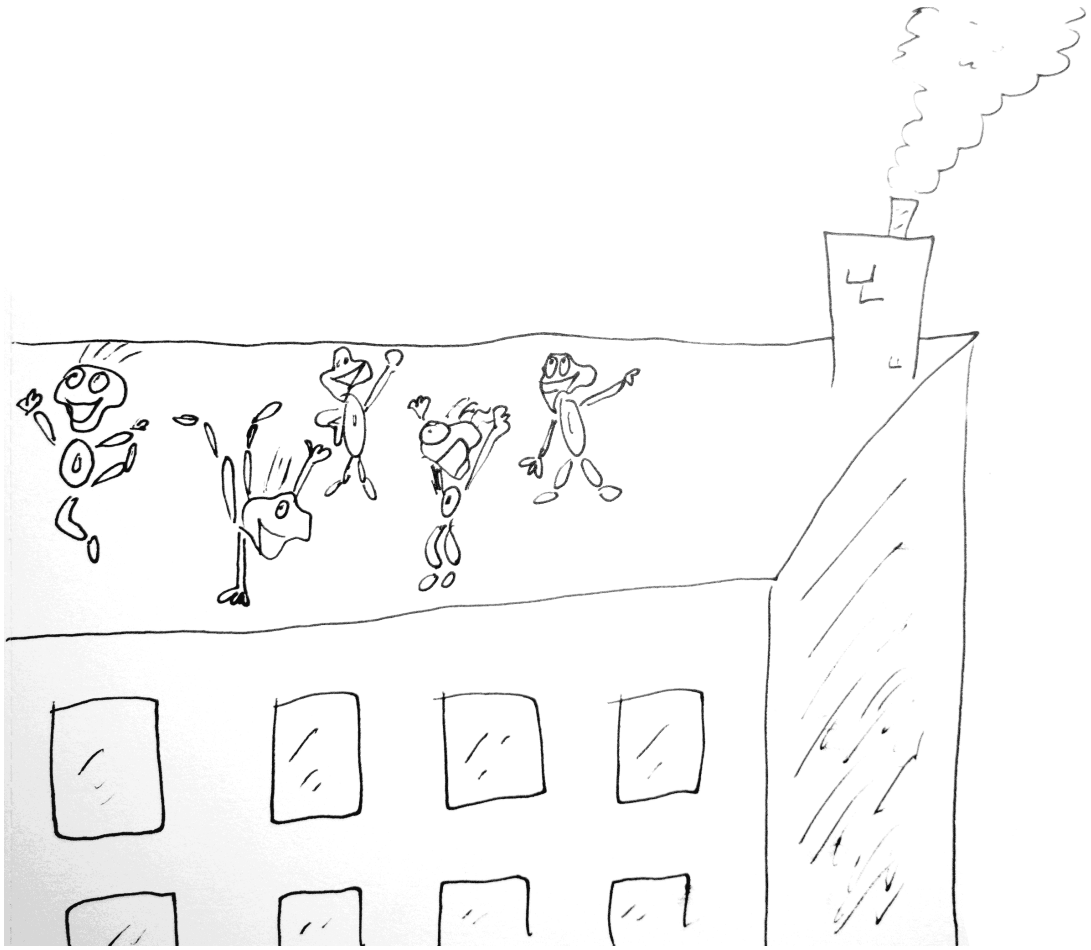
Taking a photo of the cabbage critter is like adding one file to the staging area. Whatever is added to the staging area will be part of the next commit.

You can also stage parts of a file with the interactive option.

```
git add --interactive # Or, use -i instead of --interactive
# Type "u" to add files, or "p" to add parts of files
```

— Branching —

You have an art show coming up, and a lot of fancy people are coming to see it. Thanks in part to your magical camera, you've completed the scene you're showing—a tableau of critters frolicking on a rooftop.



However, this scene is part of the much larger work commissioned by your mysterious donor. You don't want to work on any of new scenes, because you can't show incomplete work at your show. Yet, to meet the larger project's deadline, you can't afford *not* to work on them for the week between now and your art show.

It's a quandary. What can you do? You think it over critically from every angle, and finally have a brilliant idea: you decided to give up in despair.

Julia, your mail carrier, comes in. "Hey, whatcha sobbing on the floor about?" She asks, cheerfully. You hiccup-weep as you explain your situation, along with all the progress you made up to that point with the magic photos.

"Oh, is that all?" Julia asks, puzzled. "Use the magic words to have your tree grow a new branch."

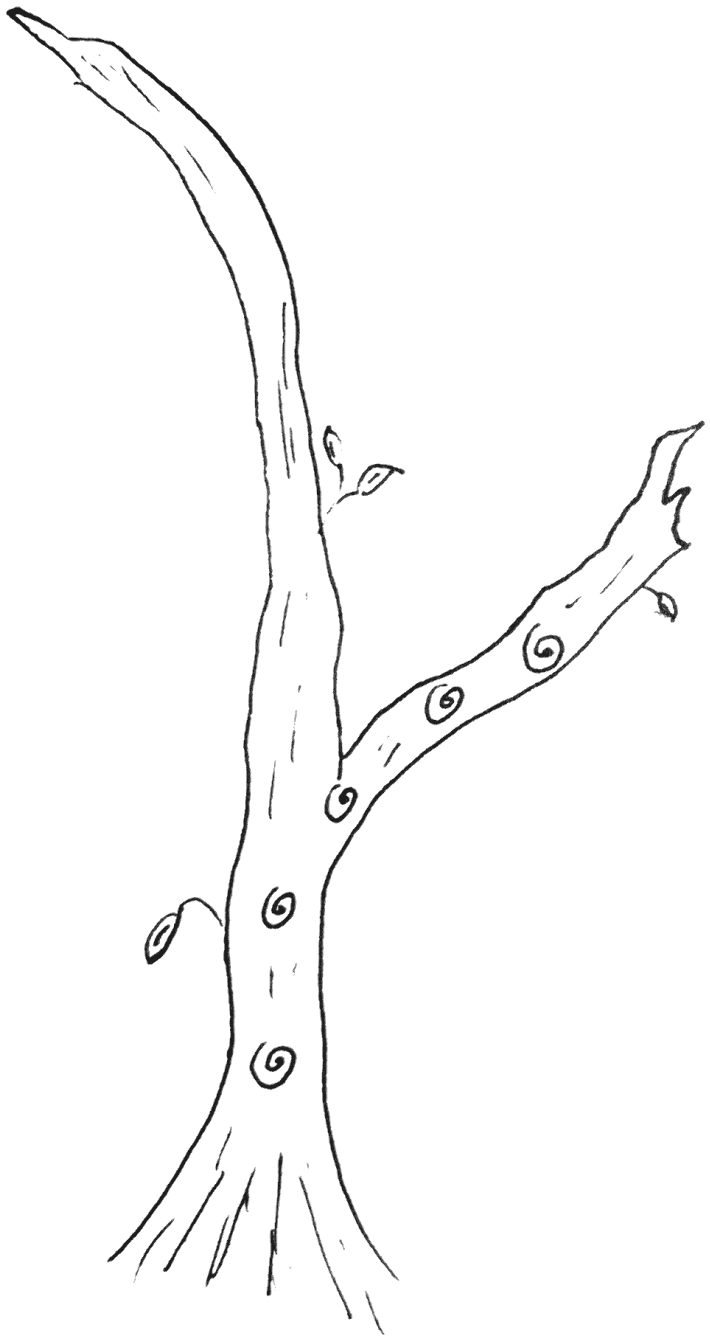
You stare at her. "How does that help me?"

Julia rolls her eyes. "Because, the knots on the branch are separate from the ones on the trunk, silly goose. You can preserve your work for the art show while being able to work on the next part of the project."

"That's incredible!" You shout. "Why didn't you tell me about this earlier?"

But Julia is gone.

Excited, you say the magic words and your tree grows a whole new branch!



You start working on your new scene, which you're calling the "Sewer Gala". Each picture you take grows a knot on the branch, saving the work for the art show pristinely on the trunk.

These magic photo tree branches are like Git branches. Instead of being limited to a linear work stream, you can have as many branches in different states and histories as you like. This is makes work much more efficient, especially when collaborating with others.

```
git switch --create branch-name # or, use -c instead of --create
```

— Merging —

Half way through work on your Sewer Gala scene, you have an idea for another scene with critters gardening. You want to capture those ideas while they're fresh; you create a new branch from the tree trunk.

```
git switch main
git switch --create garden-branch
```

You get so carried away sculpting the garden and its inhabitants, you almost finish the scene before you realize how much time has passed!

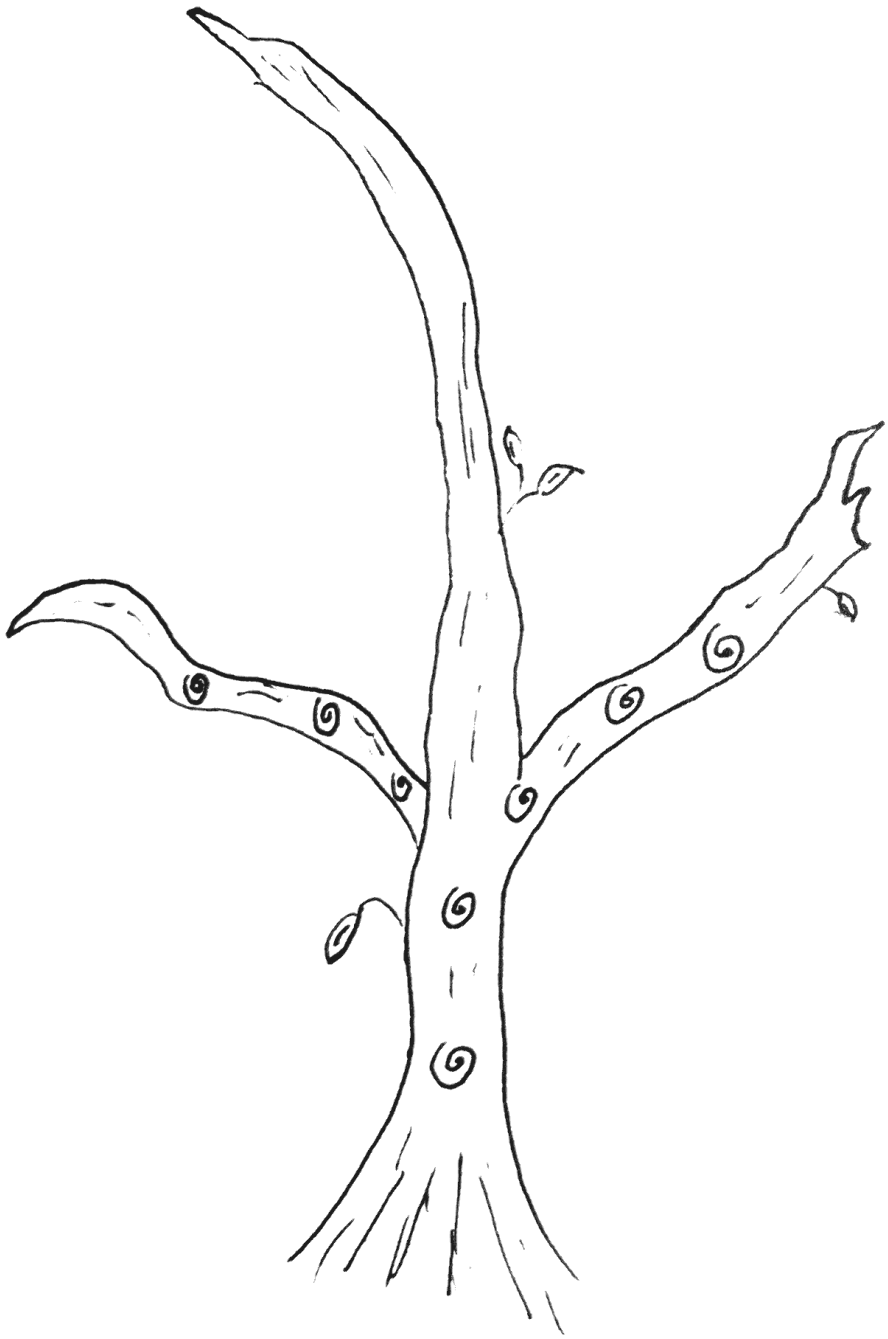
Switching back to your sewer branch, you continue your hot streak and almost finish *that* one before the day of your art show.

You can see patrons winding up the mountain road to your cottage. You switch back to the trunk. Now, only the work there appears on your work bench, without anything from the sewer or garden. Then, you spend the rest of the time before people arrive dusting.

Your show is a success! The visitors all like your work, except for the critic Harold Martimus, but he doesn't like puppies, so his judgment is suspect.

The next day, you finish both the sewer and the garden scenes. Great! Yet, now you have a problem.

You have three separate branches, each with different work. The trunk has the rooftop scene, the sewer branch has the rooftop scene and the sewer scene, and the garden branch has the rooftop scene and the garden scene.

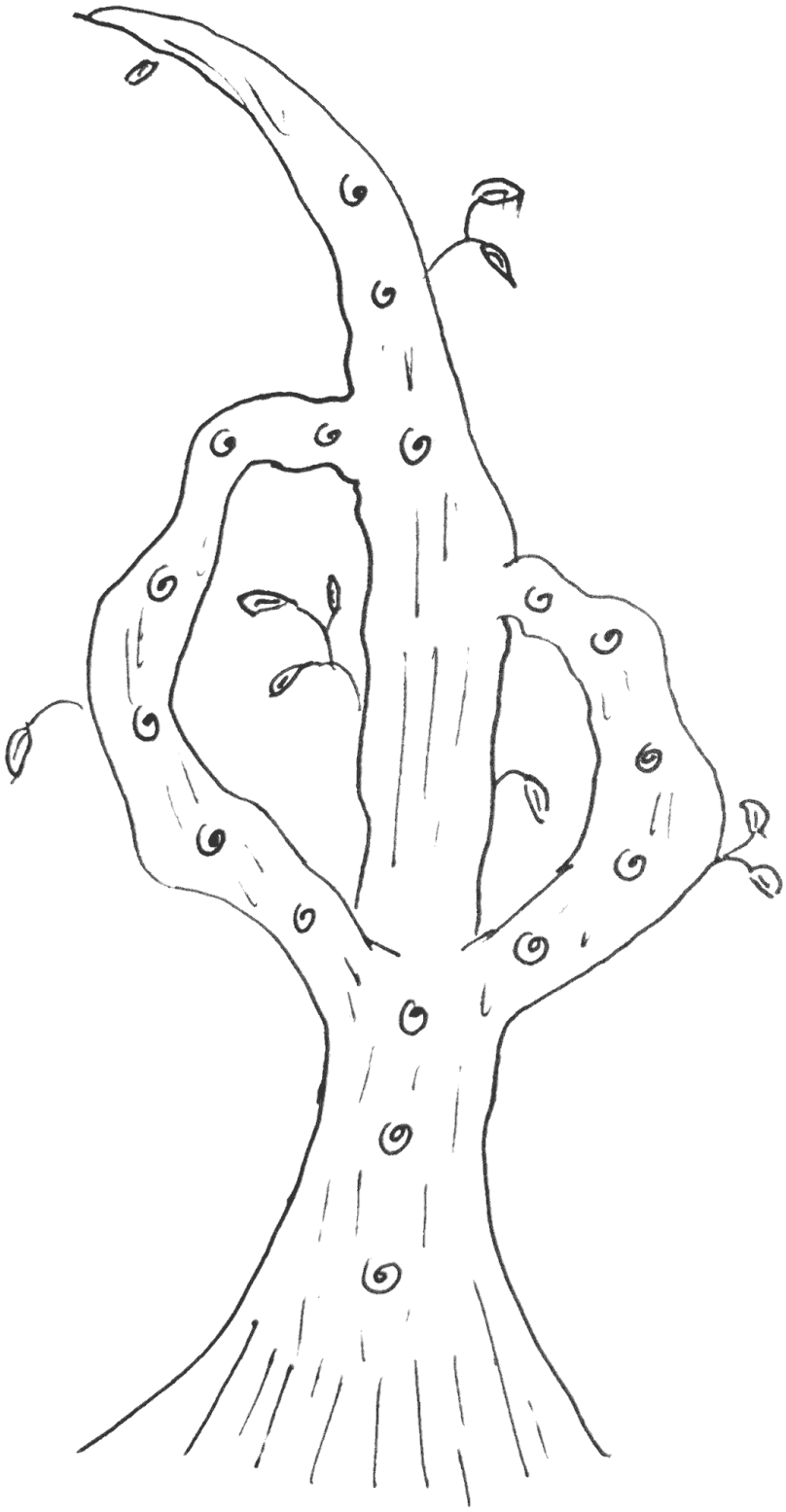


Now that all that separate work is done, you need to put it all together! You think about collapsing in a tear streaked heap again, but remember that mail carrier Julia always knows what to do in these situations. Luckily, Julia appears in front you.

“What’s up?” She asks.

“I need to put all my branches together,” you scream, terrified.

“Simple!” She says. “Those magic tree branches can grow back into the trunk—or each other!—if you say the magic words. That combines all your work!”



You ask the tree to grow its branches back into its trunk, and stare in awe as the branches bend and then merge with the tree trunk. Now the tree's trunk has all your branch knots!

The tree branches growing in are like Git merges. The concept is pretty simple; take those changes over there, and combine them with these changes over here.

```
git switch main
git merge sewer-gala-branch
git merge garden-branch
```

Git keeps track of *how* the branches came apart, and then back together, like the tree's bending branches. This is important for Git to keep things organized internally, but it's also useful for seeing how your work has evolved.

Running Git log with the graph option is like going outside and looking at how your tree has grown.

```
git log --graph
```

Sometimes, a merge will not create a bending history. In our example, this would happen if our trunk, aka main branch, didn't change at all since we made our changes in `sewer-gala-branch`. Since the `sewer-gala-branch` contains every commit the `main` branch does, and every commit in the `sewer-gala-branch` happened after the commits in the `main` branch, the newly merged commits go right on top of the others in the `main` branch, without a bending branch visual. This is called a fast forward merge, and happens automatically unless you tell Git not to.

```
git merge sewer-gala --no-ff # always bend branches
```

— Tags & Hashes —

As you're working, you happen to stare closely at a knot on the tree, aka a commit. You are surprised to see a long series of letters and numbers.

"Oh, that's a hash!" Hollers mail carrier Julia, walking up your drive.

"I didn't say anything out loud." You say, scratching your head. "How did you know what I was thinking?"

“Every knot has a hash,” Julia says, ignoring your question. “The tree creates them from a combination of what you changed and the message you wrote. It uniquely identifies each knot. You can use hashes any time you want to refer to a knot.”

“That’s pretty cool,” you say, “and let me guess, the hashes are created through magic.”

“Nope!” Julia hoots, “It’s math.”

You frown. “How does the math work?”

Julia’s eyes dart back and forth. “Uhh...magic?”

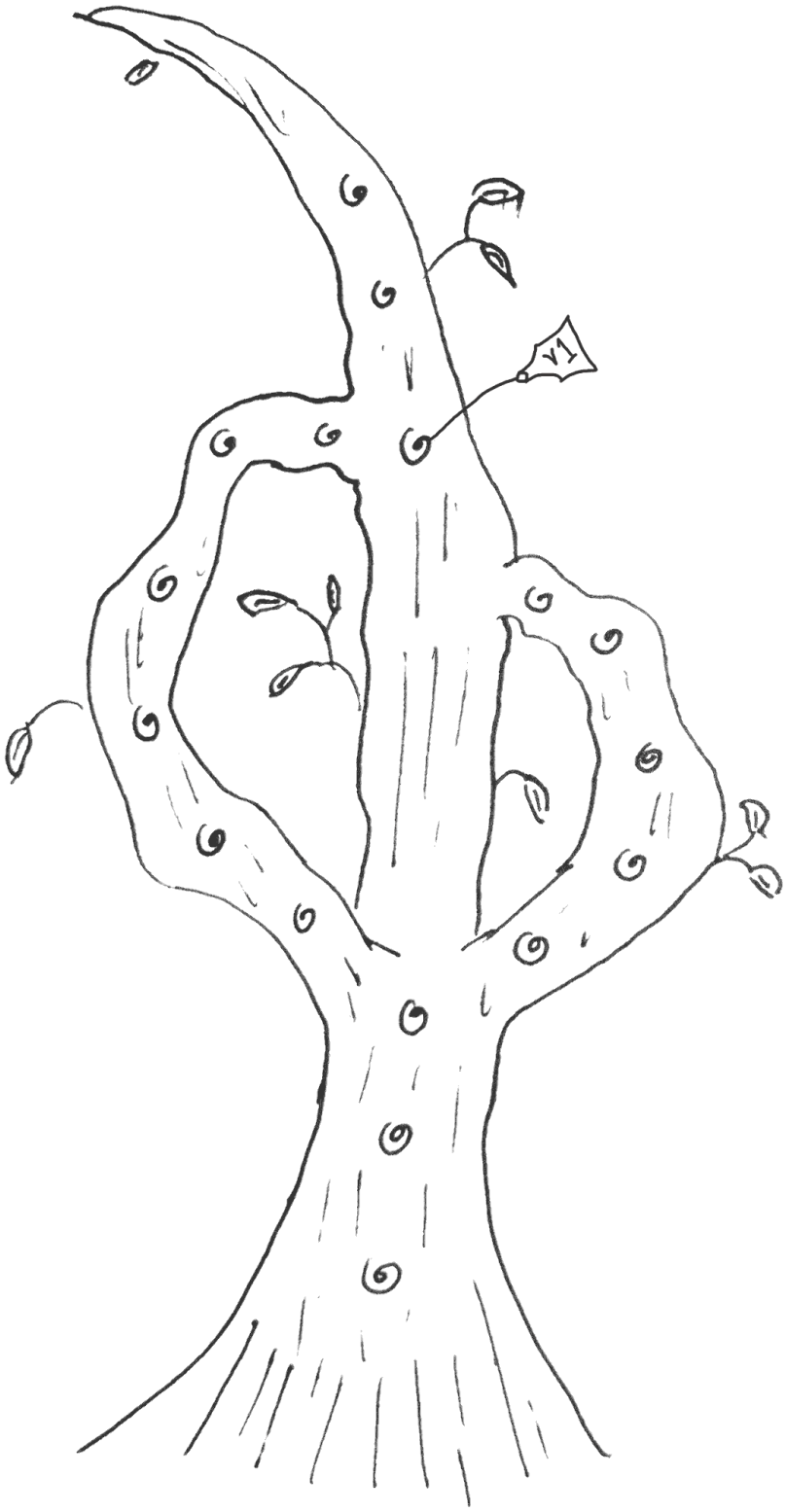
“Wait, but you said...”

But mail carrier Julia is gone.

You start thinking about the hashes, and while it is nice that every knot has one you can use, it would be even nicer to be able to give it an easier to remember name.

You’ve had to jump back to the knot of your art show a few times now, and it would be nice to be able to mark seminal moments like that for quick reference.

You etch a name on the knot from your art show with your pen knife.



To your delight, you can now use that name instead of the hash.

Every commit has a cryptographic hash to uniquely identify it. Git uses this to keep track of each commit, and for compression. You can use this hash any time you want to reference a commit.

```
git show 7afd021 # display commit information
git switch --detach 7afd021 # switch to commit
```

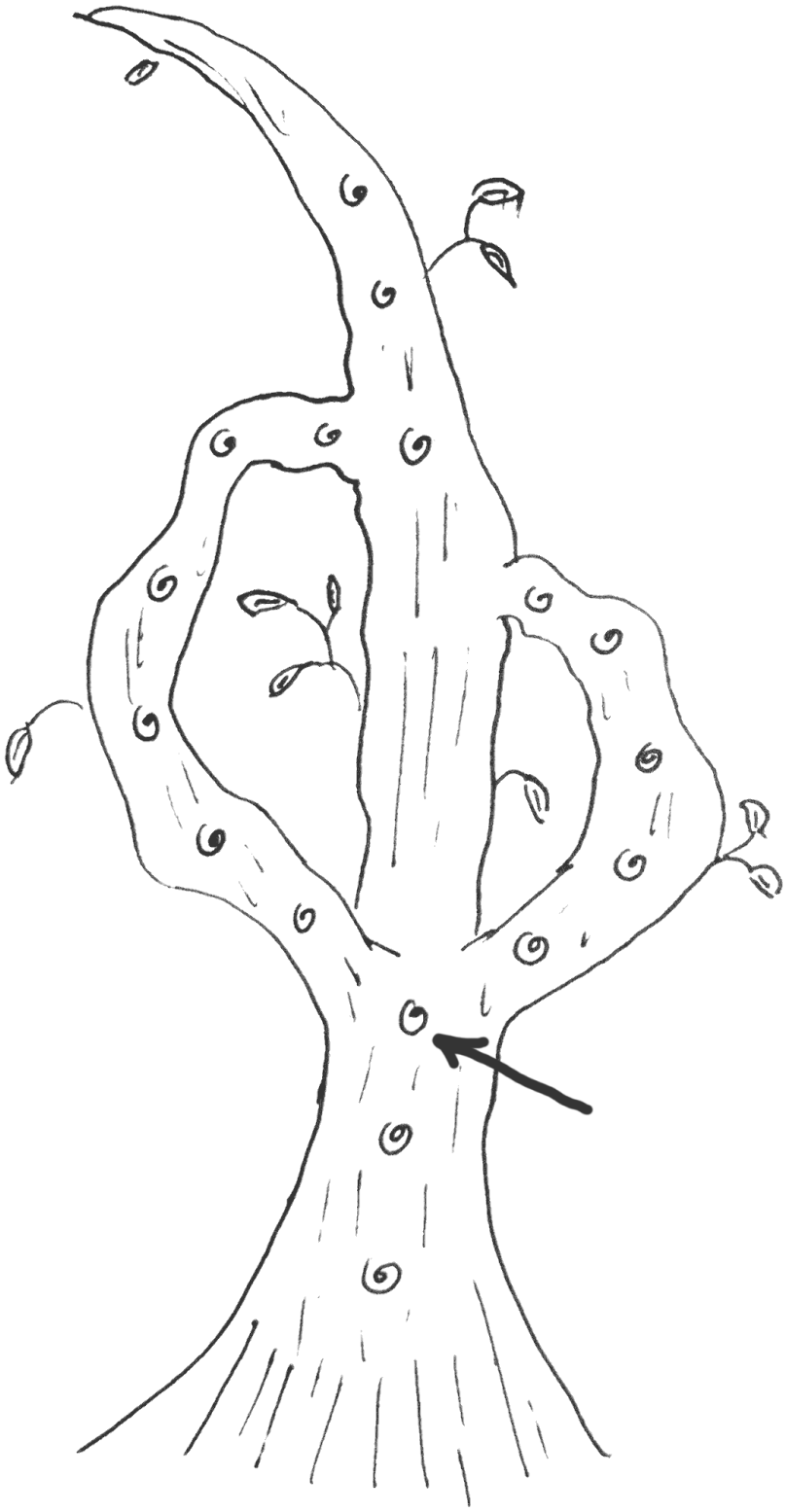
Etching a name into a knot, while a bad idea with real trees, is a good idea in Git. It is like Git tags. You use them to name a commit. Unlike branches, which by nature change over time as commits are added, a tag always points to the commit it's assigned to.

```
git tag v1 # tag the current commit with the name "v1"
git tag 7afd021 v1 # tag commit hash 7afd021 with the name "v1"
git switch --detach v1 # switch current commit using the tag
```

A tag's immutable nature makes them popular for marking releases.

— Detached head —

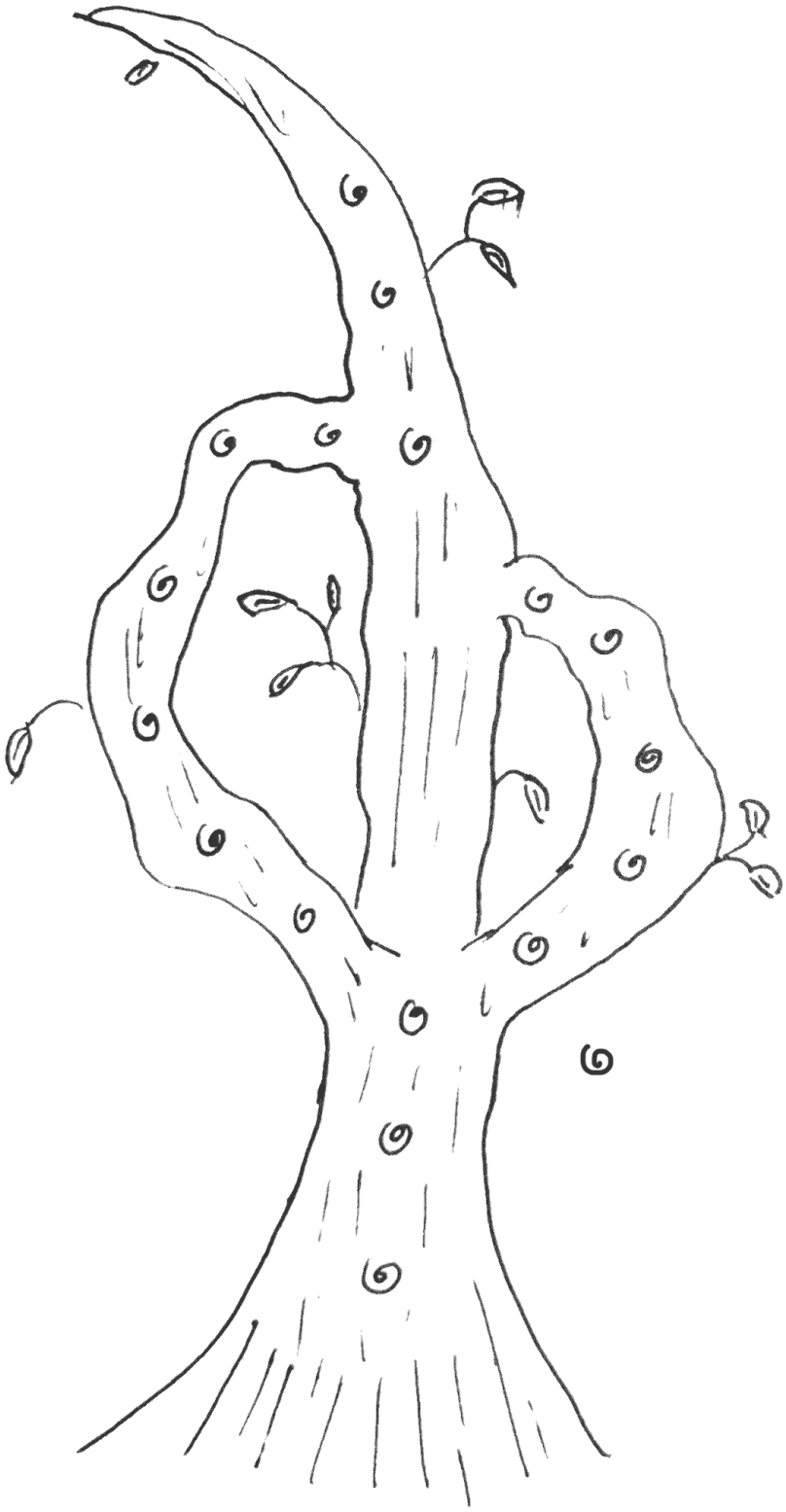
Your friend Laird pays you a visit. You want to show them the progress you've made since your show, so you use your tag to switch back to your workbench then.



Laird is impressed with the work, but you notice a mistake on one of the chimneys.

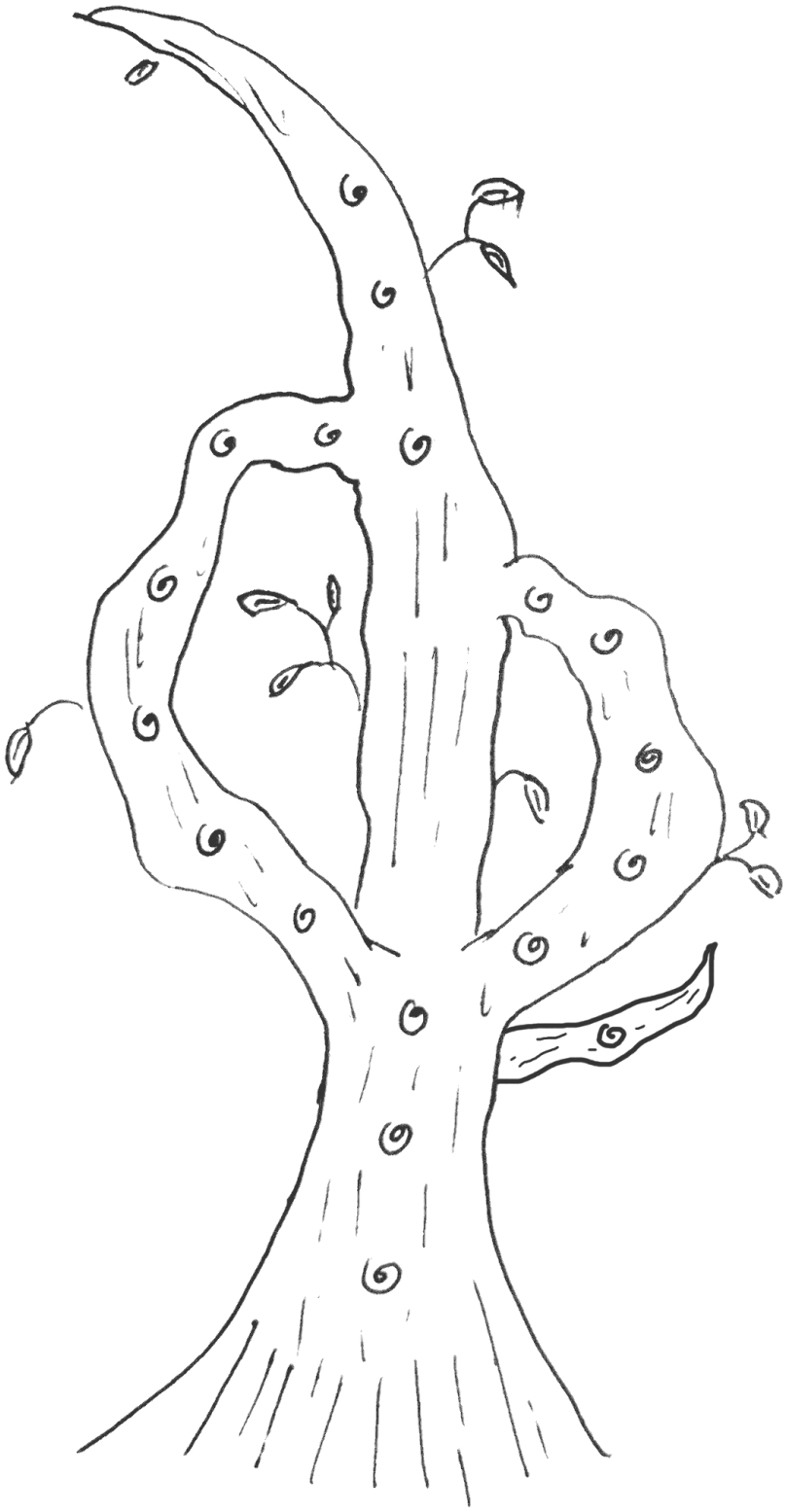
You fix the mistake, and make a new commit. You smile and chat with Laird for the rest of their stay, but you feel like something is wrong.

Your knot isn't growing out of your tree. It's not growing out of *anything*. It just floats next to the tree.



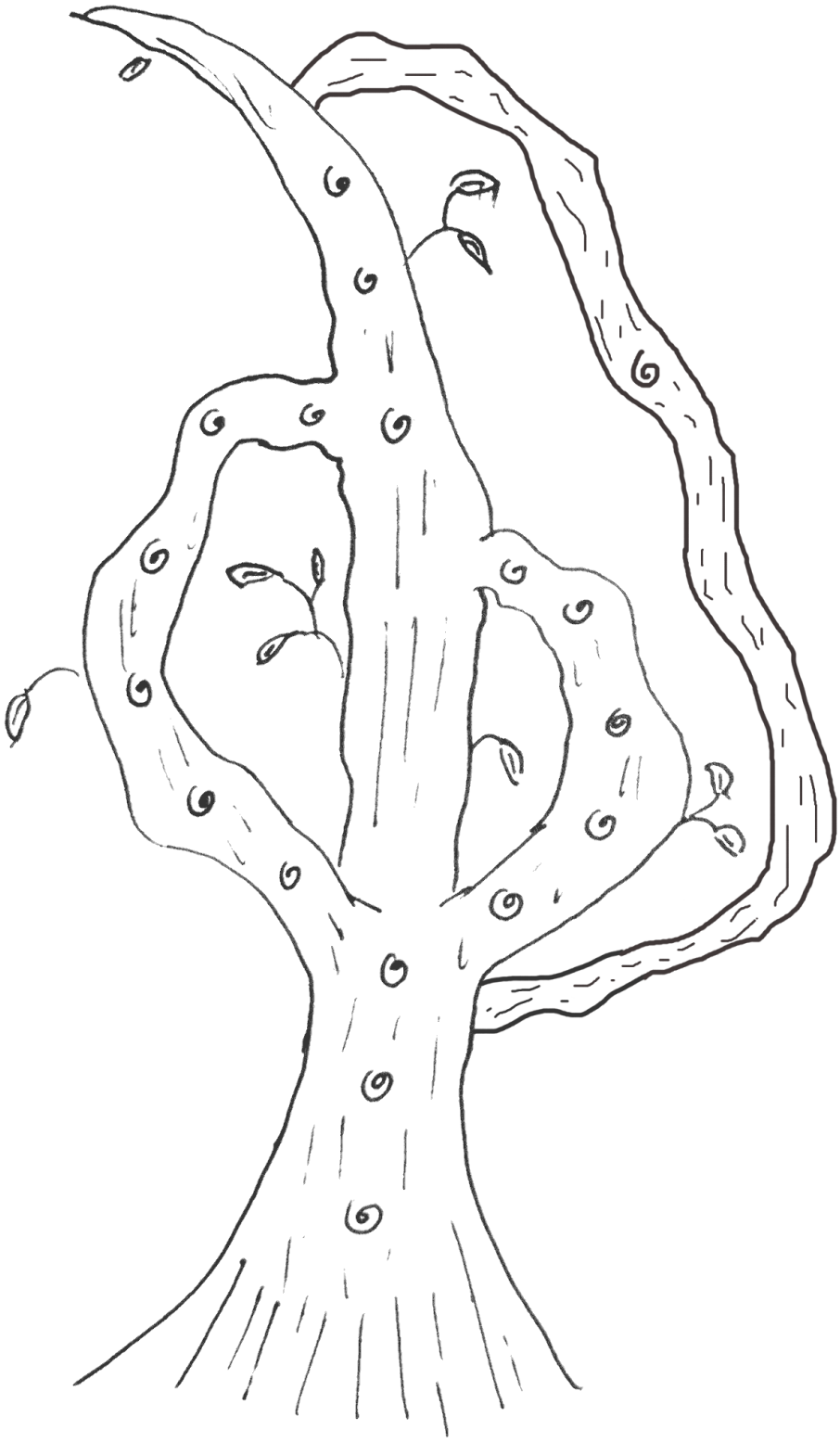
“Why on earth...” You start to say, but catch yourself. Every other knot has always grown at the newest tip of a branch. That’s how trees work; new growth comes at the ends. Your workbench is pointed to a knot that’s not on the end of the branch.

You have a hunch, and decide to follow it. You ask the tree to grow a new branch.



Bingo! The new branch grows around your floating knot!

New growth at the ends, right? You ask the branch to merge, and *Eureka*, the branch grows into the top of the tree!



Checking out the previous knot is like Git's detached head state. Detached head sounds scary, given how fatal that is for humans. However, we're dealing with Git, and Git is completely inhumane, so it's fine.

Anytime you check out a commit that isn't at the tip of a branch, including the main branch, you are in detached head state. Like the magical tree, Git only adds new commits to the tip of a branch. If you make a commit when you're not checked out at the tip of a branch, you can only merge it if you create a new branch.

```
git switch --detach HEAD~4 # Go back 4 commits
# make some changes
git add . # "." means everything in the current folder
git commit --message "My detached change"
git switch main
git branch my-branch-name [commit-hash] # switching branches will show help that give
git merge my-branch-name
```

— Repositories —

The deadline for your mysterious donor looms. You've planned a massive scene with fireworks over a bridge, and critters crawling everywhere. The problem is, there's no way you can do all the labor needed to complete it before your deadline.

Fortunately, you know two other artists who are experts in model bridge making and model fireworks. With their help, you could complete the scene in time. Unfortunately, they live far, far away.

You are having a picnic with mail carrier Julia, and bring up the problem. She grins.

“The magical camera, workbench, and tree system were *designed* for collaboration,” she says, through bites of her Huckleberry jam sandwich.

“You can connect to anyone who has the same magical system,” Julia says. “You can connect to them directly, but that’s hard for everyone to follow, and setup. It’s easier if everyone points to the same central hub.”

She points at the sky. “See that big cloud over there? The one that looks a bit like a cat, and a bit like an octopus? That’s a magical cloud. We can send it your work, and others far away can receive it.”

Julia removes a seed from a pouch, and flings it into the sky. It shoots straight into the cloud, disappearing with a puff. She points at the spot it vanished, handing you a telescope. You peer through it.

“See that?” She says. “That seed is nestled in the cloud. It’s the same type of seed that grew the magic tree planted in your yard. When we send the cloud your work, that seed up there will grow exactly the way your tree down here did.”

The magical cloud is like a Git service provider, such as Github or Drupal.org. A Git service provider acts as a central place for many people to access the same code and change history. You can collaborate with others without one, but it’s annoying and technically challenging.

The seed of the tree in the cloud is like setting up an empty repository in a Git service provider. Every Git project, wherever it is, needs this basic structure. That's because a Git repository is basically the same anywhere it exists, which allows it to be hosted anywhere. That keeps it flexible and decentralized!

Remotes

"How do I get my work up in the cloud?" You ask.

"Follow me to the fireplace." Says mail carrier Julia.

"You can connect your chimney to the cloud."

You scratch your head. "Uhh, why?"

Julia rolls her eyes. "You'll see. Just say the magic words."

You sigh, say the magic words, and nothing happens. You wait. Still nothing.

“Is something supposed to happen?” You ask. Mail carrier Julia is beaming.

“It already did! You’re connected.”

“Oh,” you reply, “so is my work in the cloud?”

“No, silly goose, you just made the connection. We’ll get your work in the cloud in a moment.”

Connecting the chimney to the cloud is like adding a Git remote. A remote is the location of your project in another place, like a Git cloud provider, a sever somewhere, or even another folder on your computer. As long as it has the basic Git structure, you can add it as a remote.

It may seem like extra busy work, but this is part of Git’s flexibility and decentralization. It allows any instance of a Git repository to connect to any other versions of the project else where. You can even connect to *different* projects, although that’s beyond the scope of this article.

```
git remote add origin git@github.com:you/critters.git # Add a remote
git remote --verbose # See which remotes are set up; or, instead of --verbose use -v
```

If you clone a repository from a Git service like Github, the remote will automatically be set up for you.

```
git clone git@github.com:you/critters.git
```

— Pushing —

“We did a bunch of stuff, but I just want to collaborate with other artists.” You say, thinking about your deadline.

“Don’t have a cow,” mail carrier Julia says. “Switch your workbench to the branch you want to send, and say the magic words.”

You say the magic words to change to your main branch, then, say some more magic words to send your changes.

As soon as the phrase has left your lips, a ghostly image of your magic tree's trunk separates from the tree itself, wafts into the living room, then zooms up the chimney. You race outside to watch it streak up into the cloud, vanishing in a flash of light. You train your telescope on your spot in the cloud, and there is your tree, perfectly recreated, along with all your work on your work bench!

You had been working on another branch for a scene with wheelbarrow races. Why not send that one up? You switch to that branch, say the magic words, and the ghost images shoot up the chimney and into the cloud. With your telescope, you can see that both branches live side-by-side in the cloud, like folders on your computer.

You tell your artist friends; they have been using this magic system for years! They can now receive your changes in the cloud and start their own work.

Sending ghostly images up through a chimney is...insane, but it is also a bit like pushing with Git. A git push sends your currently checked out branch to a remote. If you have more than one remote, you can choose which one to send to.

```
git push
```

If you have more than one remote configured, you'll need to say the name of the remote you want to use.

```
git push upstream
```

A Git push recreates everything about your current branch at the remote location: all the files, the history of commits, their messages, author data. Everything.

The one exception is tags, which are not pushed by default. If you've made a tag on your computer, and would like to push it to your remote, you'll need to add the `--tags` option.

```
git push --tags
```

— Pulling —

Your artist friends work hard on the fireworks and the bridge, and pretty soon, they're ready for you to see it. This time, you grilled mail carrier Julia on how to do this before she disappeared. You're ready. Your artist friend who built the bridge added their changes to your main branch in the cloud, which was copied directly from the magic tree in your yard.

Famously, clouds don't talk to chimneys. That is, unless you ask them to. Your magical system needs to know about the new branches that exist up in the cloud, so you say the magic words to fetch that knowledge.

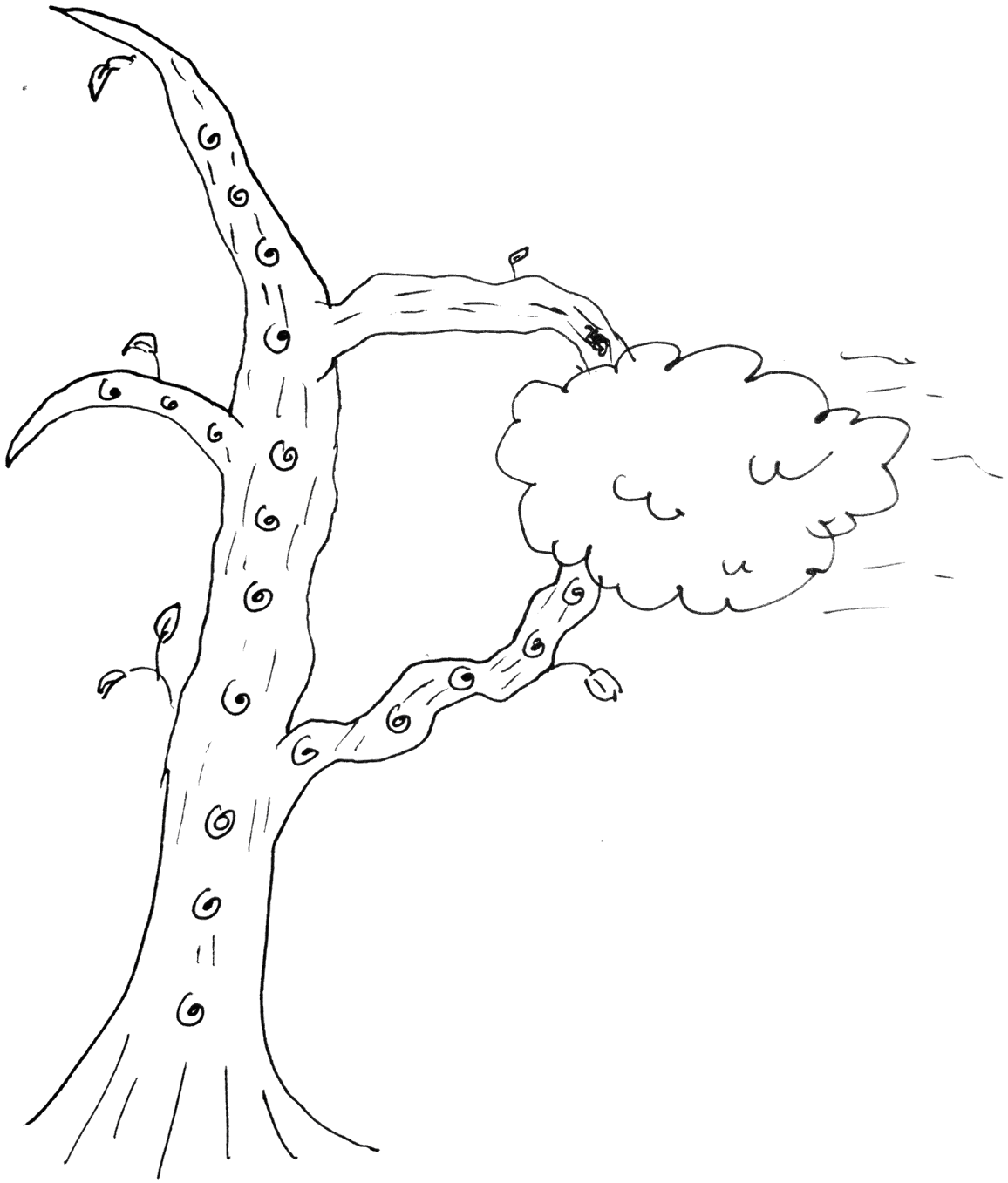
Your whole house hums with the news from the cloud, stashing everything about what's changed up there deep in the walls.

Neat, you think, but all that knowledge isn't useful if you can't put it to use.

However, you know what to do by using what you already know in a new context. It's time for a merge! You make sure your workbench has the tree's trunk, or main branch, checked out.

A merge can be between any two branches, even if one exists somewhere else, like the cloud! Since you've fetched the knowledge of all those changes, you're ready to merge. You say the magic words, and *poof!* your workbench updates with all the amazing work on the bridge!

You rush outside, and low-and-behold, your magic tree's trunk (aka, main branch) is updated with all the knots that built the bridge!



While you're dancing with joy, your bridge artist friend lets you know that there's another knot waiting for you in the cloud—they improved the bridge's structural integrity. You remember something Julia said about a shortcut. Instead of getting the knowledge about the changes in the cloud and merging the branch from the cloud with the one in your home in separate steps, you say magic words that combine the two.

"Git pull!" You shout. Boom, you've got the improved bridge, *and* you've saved your voice!

Git won't do anything unless you ask it to. That includes updating Git's knowledge of what exists up on the remote.

Running Git fetch will update Git's understanding of what has happened in the cloud since it last connected, and store that data locally, but won't change anything else.

```
git fetch
```

A merge is flexible. You can merge two branches you made on your computer together, or you can merge a branch you grabbed from the cloud with one on your computer.

```
git merge origin/main # here, you always need to say the remote name
```

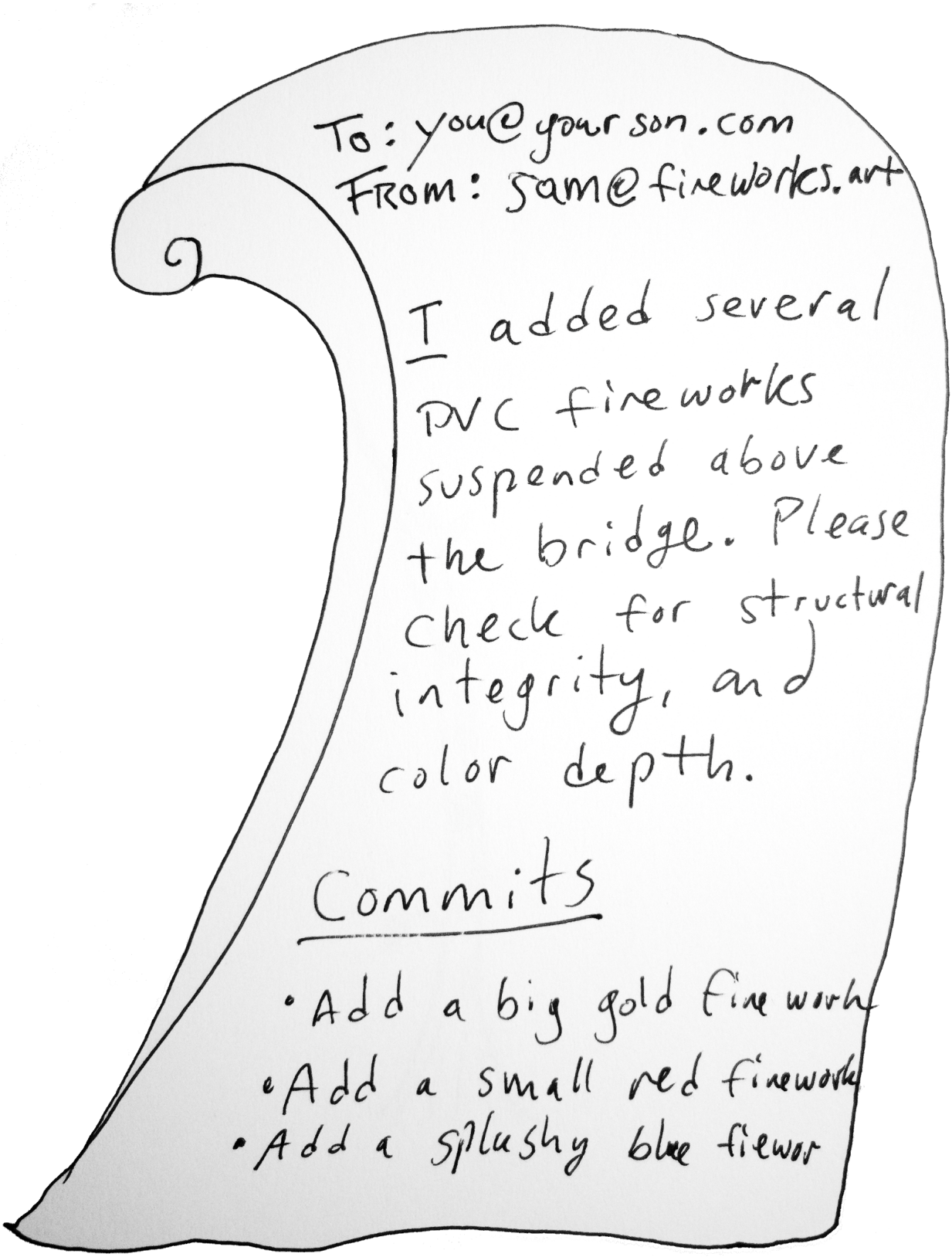
Git pull combines the fetch and merge step for the current branch.

```
git pull # Like git fetch && git merge origin/[branch-name]
```

— Pull Requests —

Your other artist friend who worked on the fireworks is ready for you to look at their work. However, their changes are more involved. They needed to affix stems to hold up the fireworks onto streets and buildings you sculpted, and they'd like to make sure everything is okay before you merge.

You get a letter detailing everything they changed, with where to look in the cloud to see their branch in detail. When you look through your telescope, you see cloudy images of what *would* happen *if* you merged the branch!



With that insight, you ask for your friend to change which glue they're using, since you know that will corrode the material under it. They do, push that change up to the cloud, and you see the result.

This letter is like a Pull Request or Merge Request. These are two words for the same thing, as adopted by different Git hosting providers; the fact that a pull is equal to a fetch and a merge, as explored above, helps explain the different language.

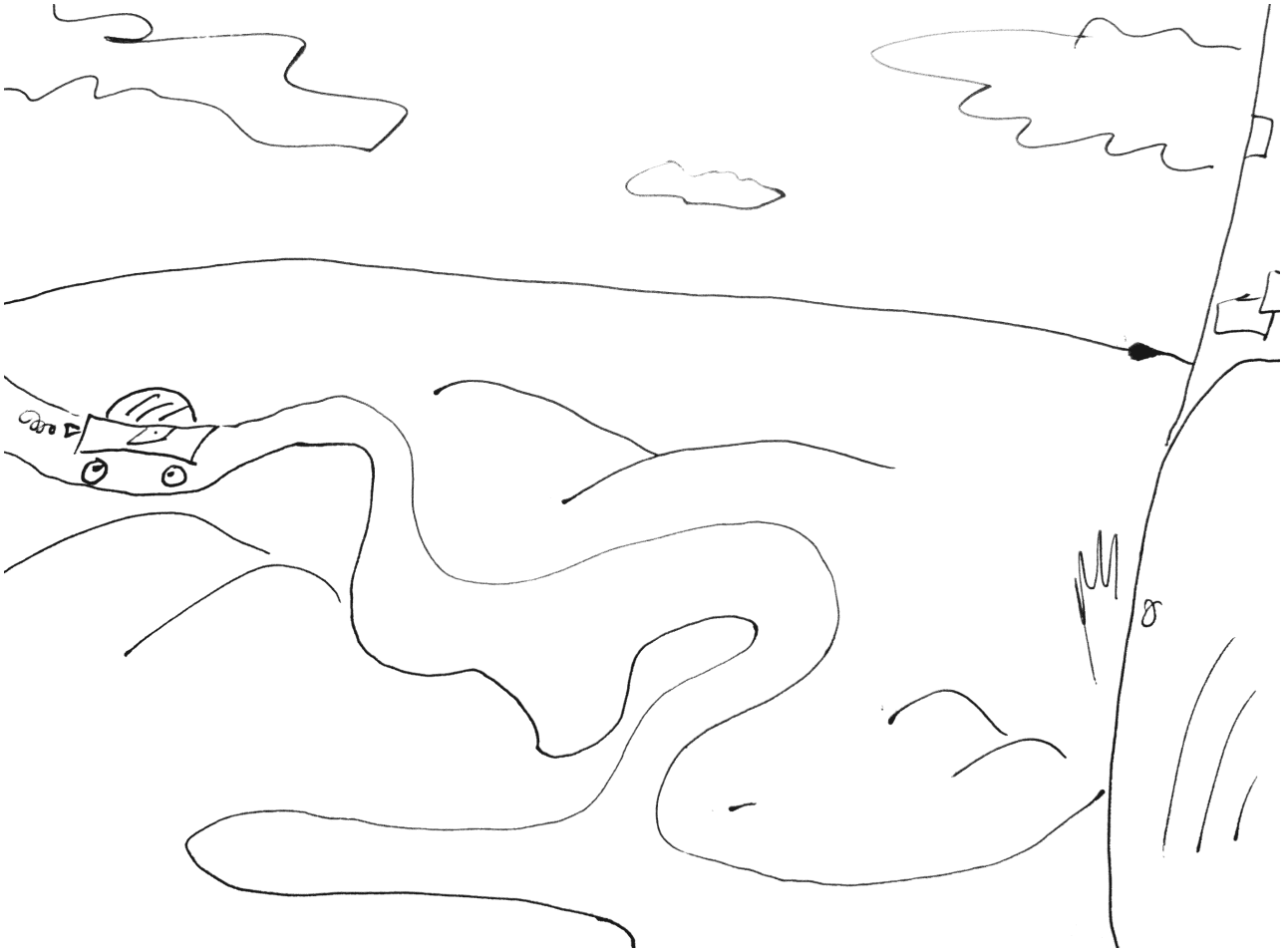
A pull or merge request is a accessible place to see what will be changed when the branch is merged, and offers an easier way to provide feedback and further collaborate on code.

— You're done, tada! 🎉 —

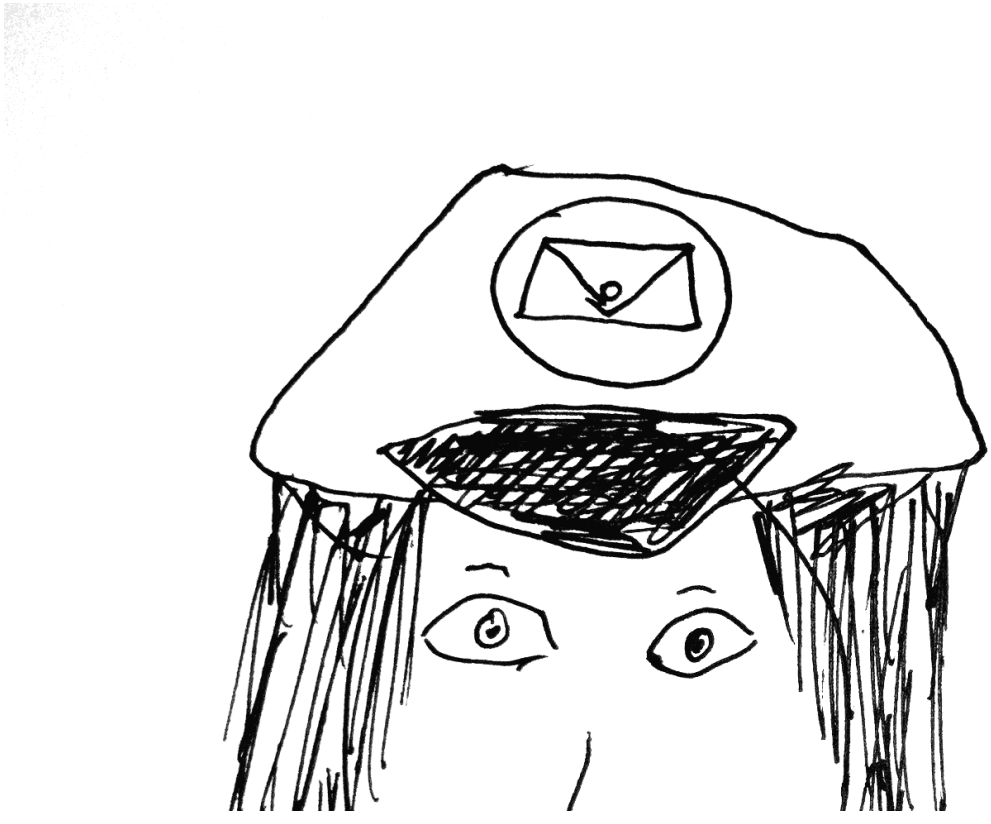
You're happy, you merge, and your big bridge scene has fireworks!

With the help of your friends, your magical system, and mail carrier Julia, your critter diorama is complete. Now, all that's left is to unveil it to your mysterious donor.

The day has arrived. They said they would be arriving in a green town car at noon sharp. Your palms are sweaty. It's 11:58. An emerald town car crests your drive.



Your palms are soaked. The town car stops in front of your door. You wring your hands into a bucket. The town car door opens, and out steps...mail carrier Julia!



“What? I love critter art,” she shrugs. “You look pale. Are you alright?”

You pick your jaw up off the floor. “Julia,” you say, preternaturally calm, “I have just one question. I think it’s pretty obvious. You’ve been coming over all this time, always teaching me about magically saving my art work, and now you’re here, revealed as my mysterious donor. I just want to know…how do you keep appearing and disappearing out of nowhere?”

“Easy,” she says, “I rewrite history with `rebase`!”

“Wait, what?” You say, but of course, mail carrier Julia is gone.

— That's it! —

I hope you enjoyed this little fantasy journey through Git. This is a high-level explanation or some of the whys and hows of Git, and there's plenty more to learn. If you're interested, I've collected some suggestions.

Thanks for reading!

Further reading

- "How Git Works", by Julia Evens.
- "The Git Parable", by Tom Preston-Werner
- "Pro Git", by Scott Chacon and Ben Straub
- "Git merge conflicts", by Atlassian
- `man git`
- `man gittutorial`
- `man giteveryday`