CS-6750 Summer 2022 Project

Kyle King
kking303@gatech.edu

***Abstract*—**Many developers utilize some form of "dotfile" management where they use tools to backup, version control, and/or synchronize text-based configuration files across computers. One popular tool, *yadm: Yet Another Dotfile Manager*, has many strengths, but has a few usability issues as identified in a Think-Aloud interview and two surveys. Based on the need finding insights and a heuristic analysis, a Terminal User Interface was proposed as a potential way to resolve the most prominent usability issues. Additional iterations through the User Centered Design process are necessary and an Evaluation protocol is proposed for future work.

1 INTRODUCTION

Many developers spend time personalizing their computers, and over the years, may accrue an extensive set of custom configuration for everything from basic Shell PATH manipulations to extremely personalized vim configurations (Hahn, 2022). There are dozens of ways for tracking, syncing, and generally managing these files, which are broadly known as "dotfiles" because of the Linux convention of a leading period (Borkiewicz, 2019). Based on the list of top tools on https://dotfiles.github.io/utilities/ (Glovier, 2022), I reviewed the top three projects based on star count and selected *yadm: Yet Another Dotfile Manager* (Stars: 3540) (Byrne, 2022). I chose *yadm* over chezmoi, because chezmoi requires separate steps to edit and apply changes, while with *yadm*, the files are edited directly (Payne, 2022). When compared to DotBot (Athalye, 2014) or Stow (Invergo, 2012), *yadm* appeared to be more opinionated in a way that may be better suited for beginners who will need to make fewer decisions.

yadm can be found on Github and has a documentation website. To start, users can either manually install using the command below (Byrne, 2022) or seek out an installer for their operating system. Make sure that the *yadm* executable is present on your PATH and that git is installed.

```
# As a generic way of installing for most Mac and Linux shells, run:
curl -fLo /usr/local/bin/yadm https://github.com/TheLocehiliosan/yadm/raw/master/yadm \
    && chmod a+x /usr/local/bin/yadm

# Quick Start

# Initialize
yadm init
# Add the first file (example file)
touch ~/.test-file
yadm add ~/.test-file
# Begin tracking changes
yadm commit
# Now edit (or remove the file) and see the difference
rm ~/.test-file
yadm status

# From here, there are plenty of more advanced features, such as templating,
#   encryption, and files that are OS-specific
# For examples, take a look at: https://yadm.io/docs/examples#
```

2 INITIAL NEEDFINDING

2.1 Initial Needfinding: Think Aloud

In order to gain a better understanding of user tasks and needs, I will conduct a single think-aloud interview. The participant will talk through the task of cloning a repository; make local edits, commit, and push; and the task of pulling new changes.

The user was a Senior Software Engineer who was comfortable with the command line. After resolving a few issues installing the necessary package and *transcrypt* dependency, the participant attempted to clone the shared dotfiles repository, but immediately ran into an issue with git for differences with their local files. Before and after cloning, the participant was unsure what would happen to these files. In particular, the git labels were not necessarily clear for which file was from where. To resolve the issue, the participant did not know the necessary git commands by memory, but could quickly reference them from documentation or by web search. Making changes, pushing, and then pulling were straightforward.

2.2 Needfinding: Survey 1

Based on the Think Aloud, I developed a survey to gather feedback from a wider audience and understand the generalizability of the initial insights. To target the right users, the survey was shared with the CS-6750 class asking for participants who were at least partially familiar with dotfiles and on Reddit in *r/dotfiles*. After ten days, the survey received eighteen responses and the raw results are captured in Appendix A.1.

Who are the users?—The participants are highly technical and frequently use a terminal every day (Figure 1), they have multiple synced computers (Figure 2), and they use a wide breadth of tools for managing dotfiles (Figure 3). They are predominantly Associate, Senior, Principal, or Staff Software Engineers, but also include two Technical Leads, a Cloud Security Engineer, an Analyst, and a Data Scientist.

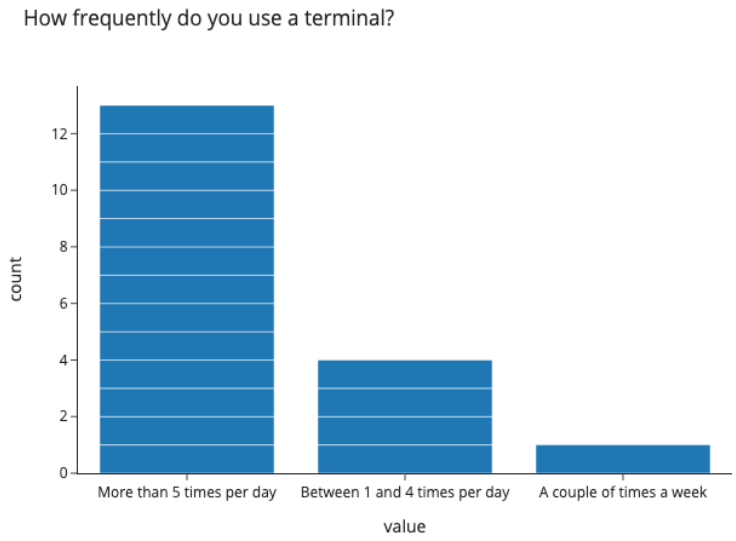

Figure 1—How frequently do you use a terminal, such as Windows Command Prompt or Mac Terminal?

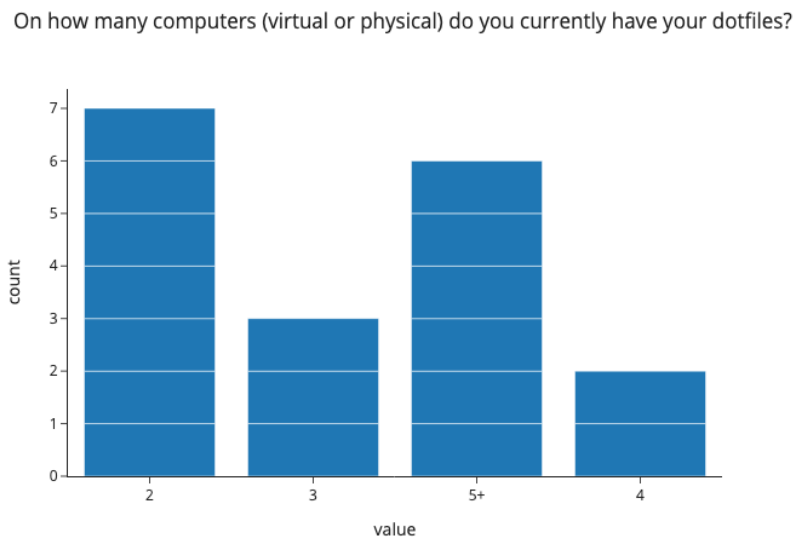

Figure 2—On how many computers (virtual or physical) do you currently have your dotfiles?

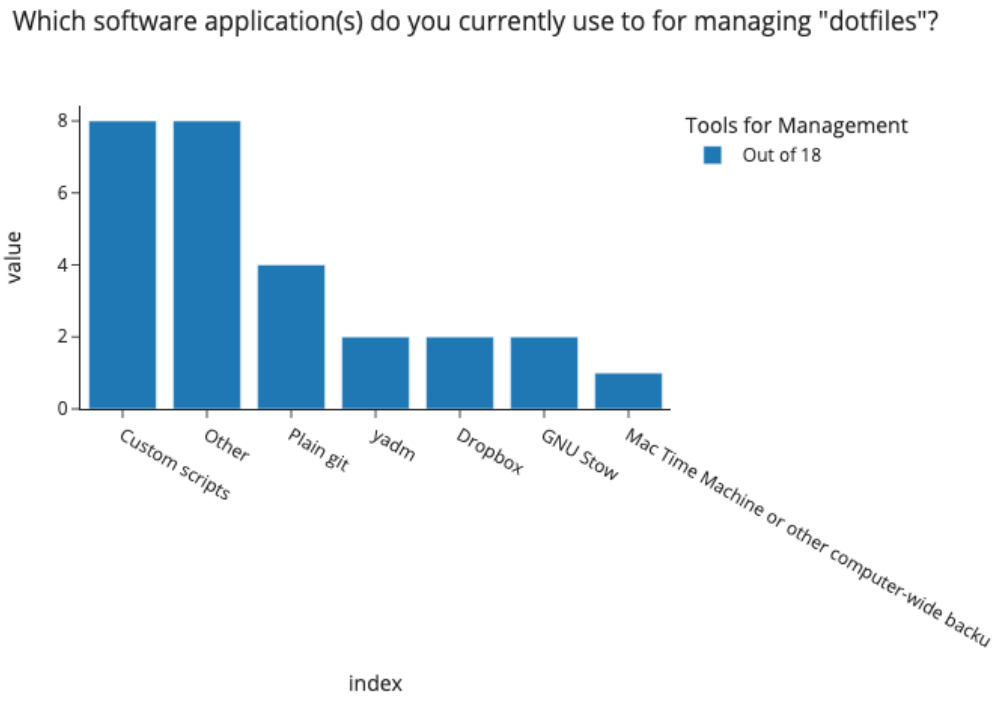

Figure 3—Which software application(s) do you currently use to for managing "dotfiles" (Desktop configuration files, such as ".bashrc" or " /.config/nvim/init.vim")?

Where are the users?—The participants are managing dotfiles either at work and/or on their own time.

What is the context of the task?—The participants are working with coworkers to sync configurations and/or managing their individual dotfiles separately.

What are their goals?—The participants reported variations of the need to have a "consistent user experience across my various machines" and to manage changes to complex configuration files.

What do they need?—Different participants had competing needs. Some valued privacy and security, while others were open to public files at the benefit of ease of use. One participant reported the need to sync dotfiles offline through a flashdrive rather than on hosted services, like Github. Multiple participants indicated that they wanted to use a tool they were familiar with and didn't want to lean something new. They also indicated that they didn't always know what they need from a dotfile manager until they started tracking files and developed custom scripts that fit their immediate needs.

What are their tasks?—Participant tasks and subtasks include:

- Initialize, add, and remove files
- Store the files in version control and resolve differences
- Encrypt and conceal secrets stored in the configuration files
- Sync files across computers and account for machine-to-machine differences
- Manage installed software related to synced dotfiles
- Coordinate and share snippets with team members (which is still done largely through Slack, Wikis, and other manual processes)
- Forking and nitpicking snippets from public dotfiles
- Separate work and personal configurations

2.3 Targeted Needfinding: Survey 2

Based on the first survey and initial Think Aloud, there were a few areas that could be explored with the redesign. To gain quantitative insight and help further focus the prototype, I created a second survey of the design changes being considered and sent the survey to a few individuals who had completed Survey 1.

The survey had only two responses and the entire results are available in Appendix A.2. The first participant identified a few potential bugs that were difficult

to recover from, which identified the need for potentially better Tolerance and Constraints in the design since software bugs in open source software are inevitable. Additionally, a theme that emerged was that *yadm* was difficult to comprehend and debug when pulling changes and managing templates. Two notable quotes from the survey are highlighted below:

> "I would have yadm be more verbose about what it was doing under the hood. It's a great tool, but feels a little too magicky for me."
>
> "Can be confusing when first setting up to align shared dotfiles with existing local - 'git reset –hard HEAD' seems too forceful. Templates break too easily / don't error when they should."

One of the participants also highlighted a feature from *yadm* and almost all dotfile management tools that is currently missing:

> "While syncing dot files is fantastic among collaborators at any project, I would prefer if the entire files didn't have to be synced. A selective sync/composability would be the killer feature for me."

3 HEURISTIC EVALUATION

3.1 yadm: Heuristic Strengths

yadm meets five of the course's superset of Human-Centered Design Heuristics, which include *Simplicity*, *Comfort*, *Documentation*, *Consistency*, *Mapping*, and *Flexibility*.

Simplicity—*yadm* meets the usability heuristic of simplicity by keeping all files where they natively. Other tools may have more complicated symlinking or multi-step edit and apply workflows, while *yadm* doesn't change the user's existing filesystem. Having the files located in their normal locations means that developers can make changes and see the results immediately maintaining a fast *feedback* cycle on changes.

Comfort—*yadm* provides an entirely textual interface that utilizes the same user peripherals as any other computer application. A user can interact with *yadm* with their existing inputs, such a joystick-based text entry interface or an ergonomic keyboard. Additionally, *yadm* uses short subcommands and avoids unusual symbols, which addresses both *comfort* and *ease*.

Documentation—*yadm* provides documentation in several ways. On the *yadm* website, there are examples for getting started, an API reference, and documentation on advanced features. From the CLI, *yadm –help* or *man yadm* provide offline documentation. Lastly, the *yadm* source code is public and includes comments that are contextual relevant when a developer seeks to understand the implementation.

Consistency—The primary *yadm* user is a developer who is comfortable working with command line (CLI) tools. As such, *yadm* follows standard CLI conventions, such as providing subcommands (*yadm init*) and utilizing flags (*yadm –help*). For version control, *yadm* wraps the git CLI, which means that all commands from git are available in *yadm*, with only minor modification where needed. Each git subcommand through *yadm* provides consistent functionality, such as how *yadm clone* "clones" the hosted repository in the default location for *yadm*.

Mapping—Being a text-based command line interface, *yadm* can only implement mapping by the vocabulary used for the subcommands and flags. All twelve of the *yadm-specific* subcommands are closely related to the associated task (*init, clone, config, list, alt, bootstrap, encrypt, decrypt, perms, enter, git-crypt, transcrypt*). *yadm encrypt* encrypts the specified file, which can be used with the inverse, *yadm decrypt*. In another case, *yadm transcrypt* is used to interact with the external *transcrypt* library.

Flexibility—*yadm* does not explicitly provide flexibility for user configuration, but flexibility is inherent in the shell environment where *yadm* is used and for the git CLI utilized by *yadm*. Users can choose to add shell functions or aliases then sync those with *yadm* to make running frequent commands easier and more efficient. *yadm* provides information so that tab completion is possible where users can start typing, then press tab to finish entry of the command.

3.2 yadm: Heuristic Weaknesses

While *yadm* has many strengths, there are a few heuristics areas where *yadm* could be improved.

Discoverability—*yadm* has sufficient documentation on their website, CLI, and source code as described as a strength, but a developer must reference these resources to learn what functionality is available and how to use *yadm*. The need to reference this documentation contributes to a steeper learning curve and can make learning how to use *yadm* more difficult, but since the documentation is

available, this can be considered a PASS with Difficulty heuristic finding and still a weakness.

Gulf of Evaluation—For users who sync changes across computers or between users, a user's set of local files managed by *yadm* may frequently become out of sync with the hosted versions. Typically this can be resolved with *pull* which will update all files in-place, but any local changes may create conflicts that require manual resolution. This is currently an area where *yadm* doesn't provide any specific tooling and relies on existing git tools, which are typically configured for normal git repos and not this edge case. This application state can be replicated with the steps shown in Appendix A.3.

Based on git configuration, the resulting git status may appear to show that the file was deleted, which could be very confusing to a user who is now unsure of what to do. Additionally, the deleted file is staged, but the local file is still present as verified by running *cat*. A user might reasonably expect to either see the diff or have an error raised that the clone could not be done safely. This particular usability issue likely arose from *Expert Blindspot* where managing state before cloning or recovery from this state is something avoided subconsciously.

Tolerance—Actions in git can sometimes be undone, but that can be a complicated process and has a high risk of accidentally losing local changes if not careful or losing track of what had changed. However, the more pressing issue is the sensitivity of the configuration files controlled by *yadm*. In situations like the one described above, git can add text to the file indicating a diff while working on a rebase. If the configuration file is loaded at the same time or used by other applications, the syntax error could have substantial consequences and potentially lock out the user from their computer. While this corner case is unlikely, the risk is high and inherently a side effect of the design for *Simplicity*.

4 INTERFACE REDESIGN

The interface redesign is shown as a series of sequential mockups. The first card shows the context of running on MacOS while the rest focus only on the Terminal emulator which is generic to any supported OS. The tool can be run on any Linux-based system including *Windows Subsystem for Linux*. Importantly, this version of the prototype is read-only and changes must be made in the user-specified Editor.

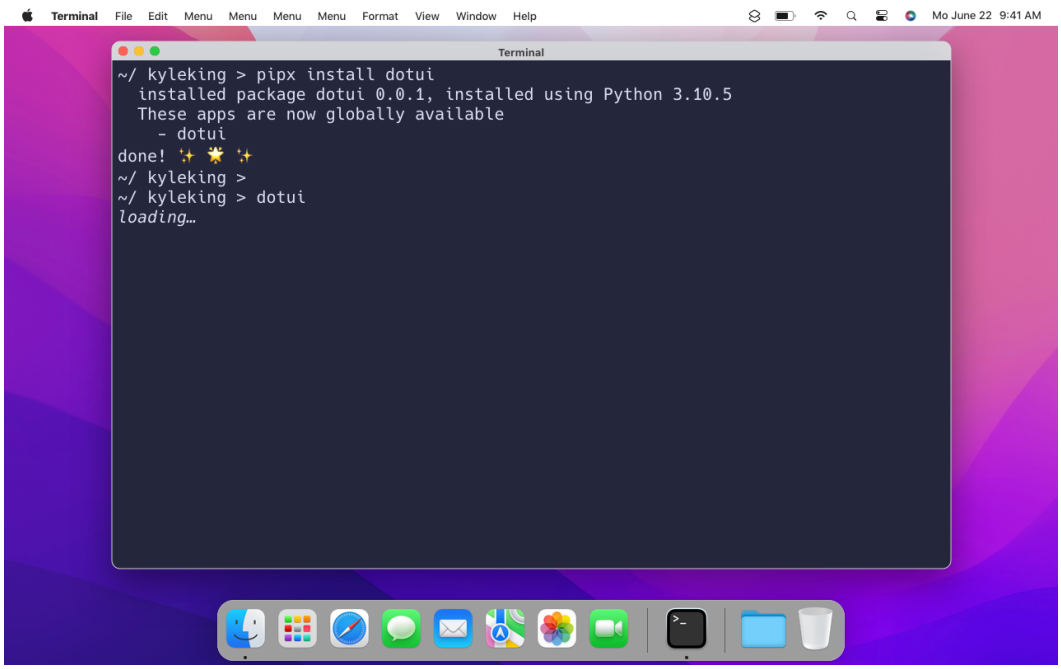

Figure 4—Card01

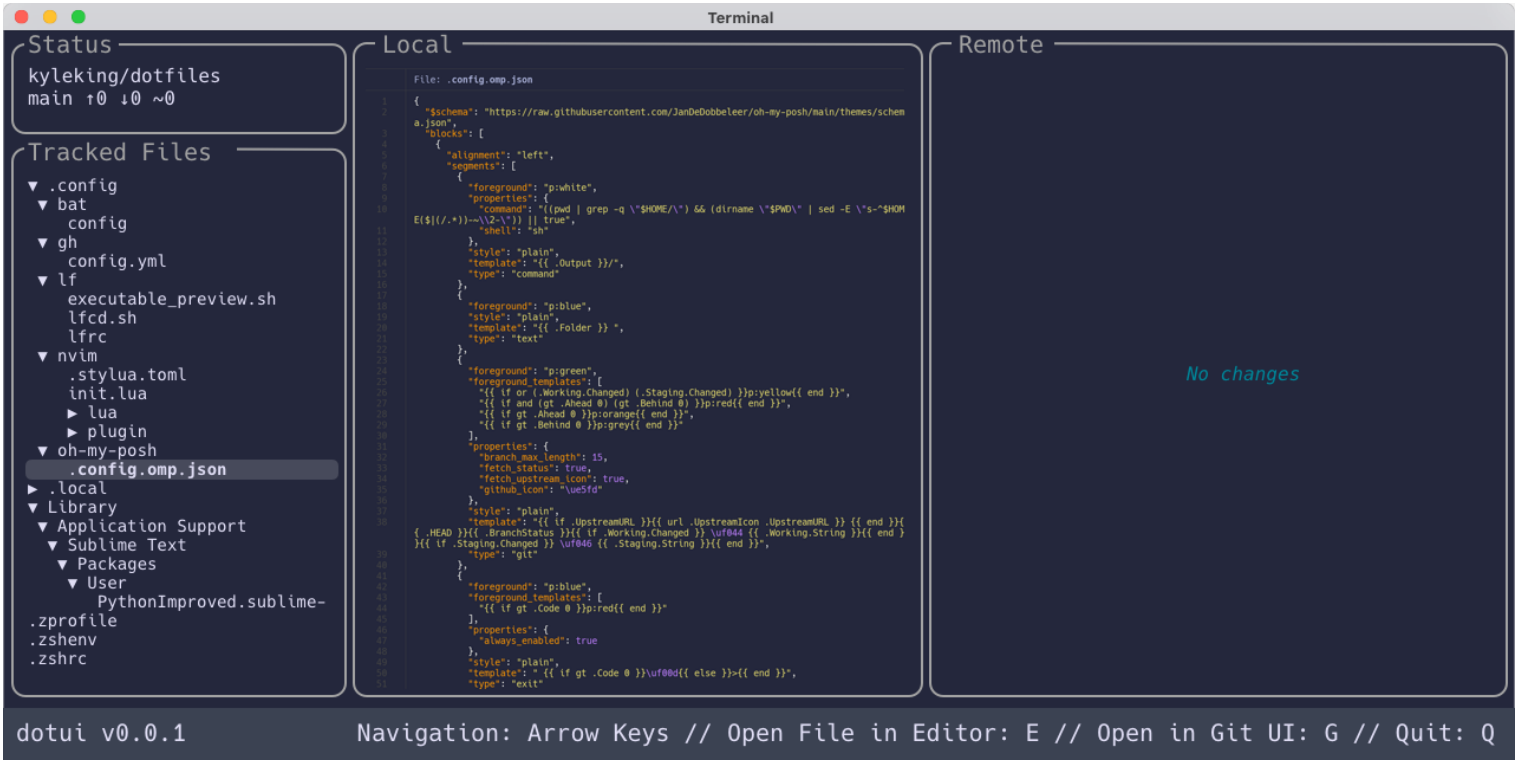

Figure 5—Card02

Figure 6—Card03

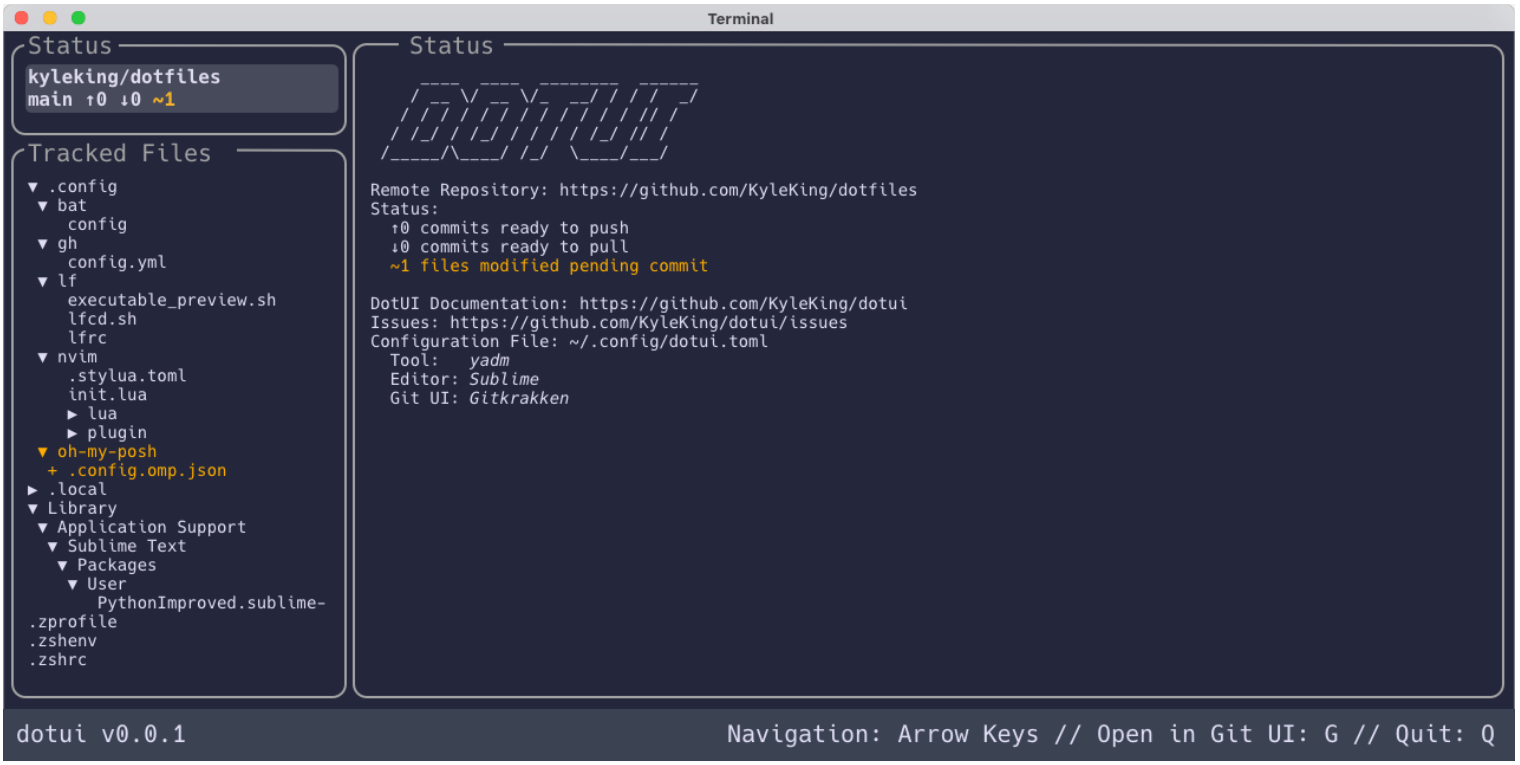

Figure 7—Card04

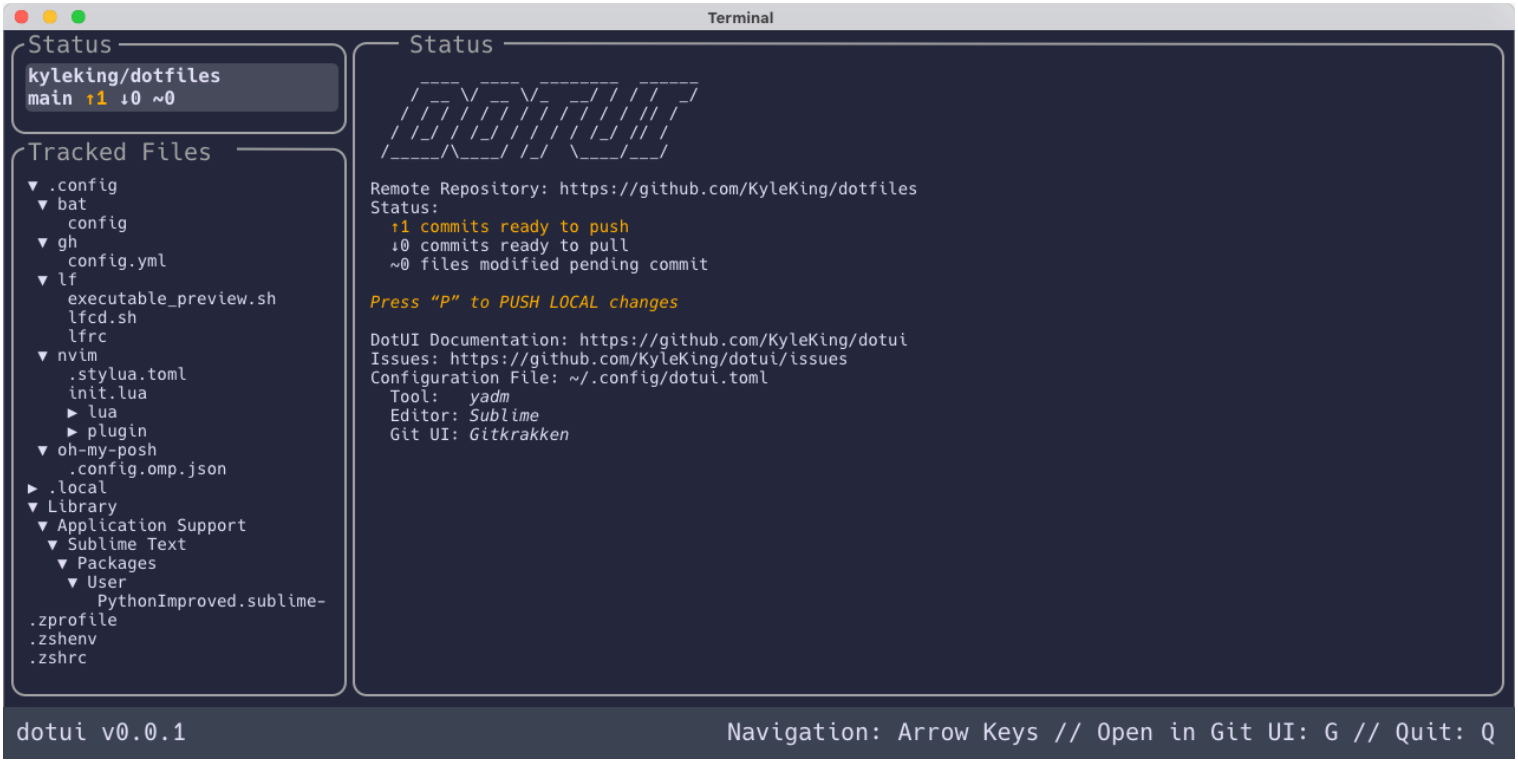

Figure 8—Cardo5

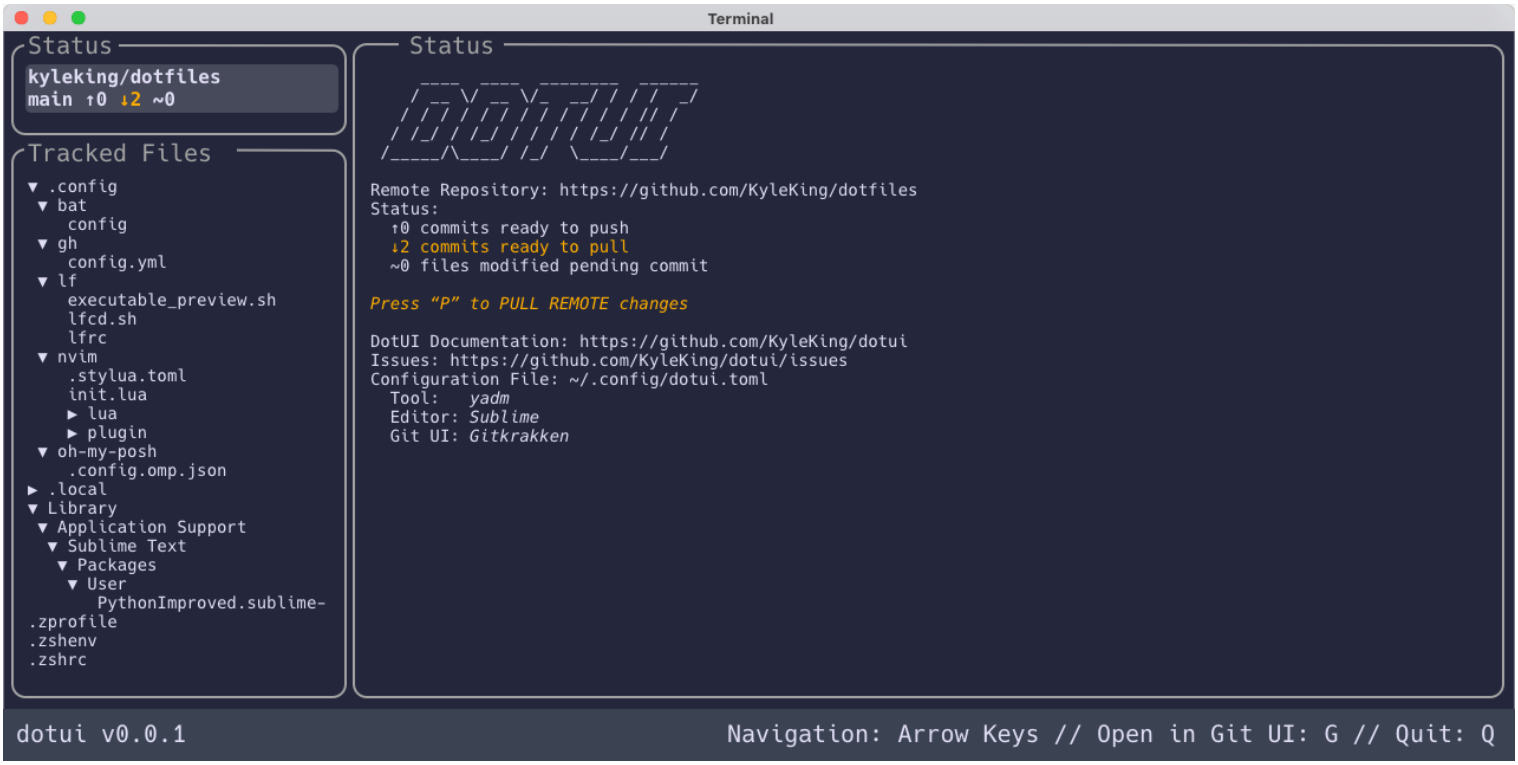

Figure 9—Cardo6

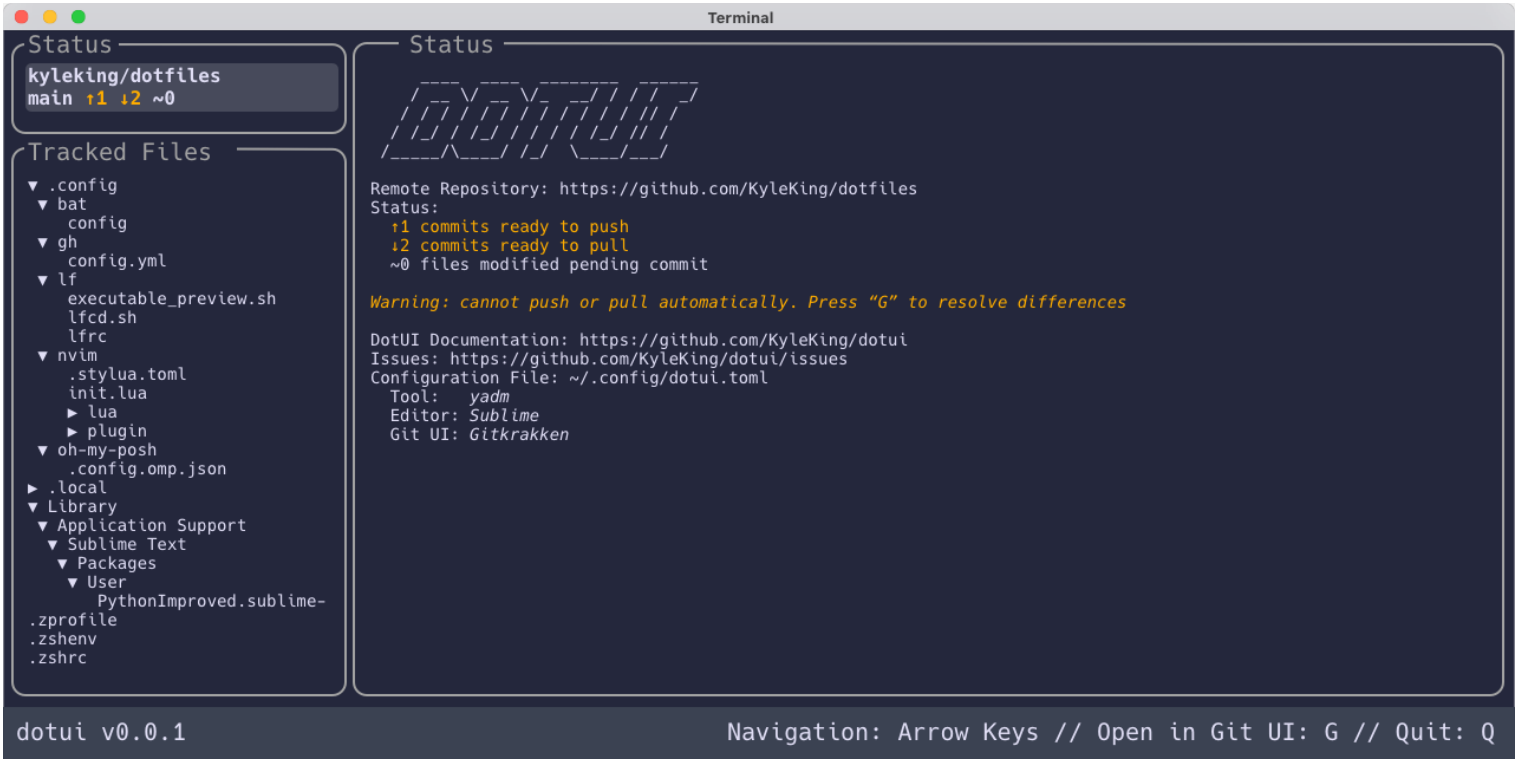

Figure 10—Card07

Note—Figure 11 is shown with a proposed *v0.0.2* change where a new section, *Templates*, is added. If the previous cards were to be updated, the *Templates* section would be shown in the bottom left with a shorter *Tracked Files* section and scrollbar. Like the other cards, the bottom help text is context-dependent and updates based on the main panel view.

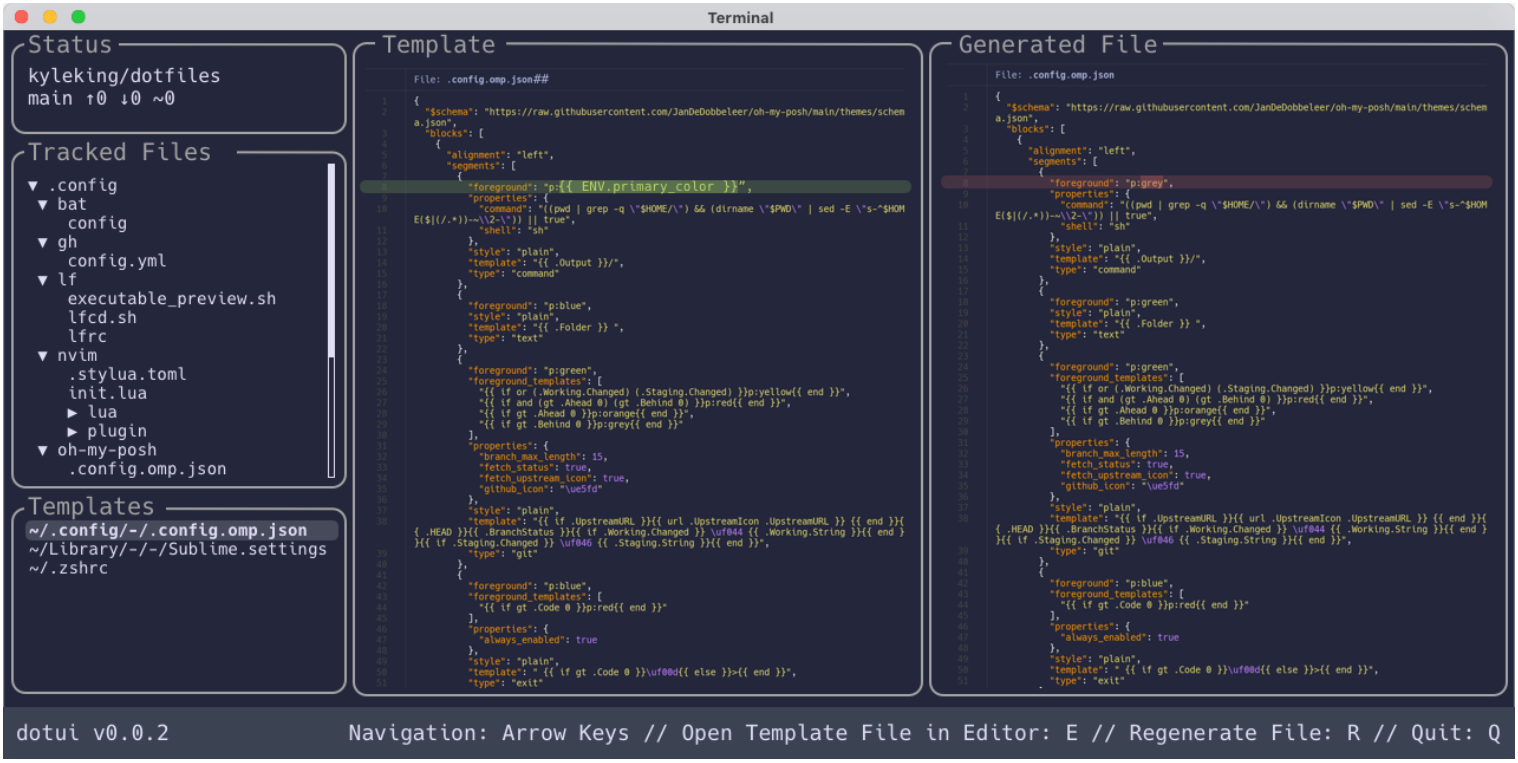

Figure 11—Cardo8

5 INTERFACE JUSTIFICATION

The interface was redesigned with a focus on user tasks, such as better external tool integration; improving discoverability; reducing the cognitive load; and improving constraints to avoid errors. Further, the prototype *dotui* interface is not mutually exclusive with the *yadm* CLI and both will likely continued to be used, but most tasks can be done from the TUI or user-specified applications rather than in the CLI directly.

Reduce Cognitive Load—Based on issues raised during the needfinding stage, a TUI became a likely candidate to improve usability. Providing a TUI around the *yadm* interface means that users have a visually persistent and live-updating list of all tracked files and git status, which reduces the *Cognitive Load* for users. Users currently need to re-run the *yadm list* or *yadm status* commands to see which are tracked and what has changed respectively, but that can now be seen in one place without the need to scroll back through the terminal buffer. Further, the TUI polls the remote and local changes on a reasonable interval, so that the user has all of the information in one place and doesn't need to remember the state.

Flexibility—A major theme in the first survey was that users wanted to use tools that were familiar to them, which was a major motivation for why they created their own scripts or used a combination of tools they were familiar with. The redesign took this into account by proposing a design that is not tool-specific and external tools are prioritized.

The initial prototype would support only *yadm*, but the implementation could utilize plugins and a Model-View-Presenter architecture that would facilitate support of additional tools in the future. The TUI Views were designed around general tasks that are commonly found in dotfile management tools regardless if the tool is a custom script or a competing project. Users can contribute plugins that meet the prescribed interface to return the list of tracked files, list of templates and generated files, and other details about status, which can then be polled and displayed in the TUI. The prototype does make some assumptions about the use of git, but that was the version control tool of choice and something that could be later refactored to be configurable.

Throughout the *dotui* prototype, there are reminders of the keyboard shortcuts that directly open a user-specified external tool. For files ("E"), the specified Editor is opened, which could be used with any application from VSCode to TextEdit or which could fallback to the system default (i.e. *open ...* on MacOS). For the bare git directory ("G"), the user-specified editor can be opened, which is something that requires advanced knowledge now, but several editors could be supported by default.

Overall, the core *Flexibility* heuristic analyzed as a strength of the *yadm* CLI is extended and improved upon with this prototype. The user-configurable integrations with external tools and ultimately allowing *yadm* to be optional leads to much greater flexibility.

Simplicity, Comfort, and Documentation—The prototype still relies on *yadm* and doesn't make any fundamental changes to the feature-set. Users will still need to access the *Documentation* for certain advanced features and the *Simplicity* of the fundamental design decision for a bare git repository is unchanged. Additionally, the TUI continues to be entirely keyboard driven so that the user's preferred inputs will continue to work and the number of commands were kept to a minimum.

Mapping—The selected keyboard shortcuts for the prototype attempt to reasonable map the shortcut to the name by picking appropriate letters. For example,

"G" for Git UI, "E" for Editor, "P" for Push and Pull (contextually aware), and "Q" for Quit. Additionally, icons and scrollbars are used where appropriate.

Consistency—The prototype borrows common design elements from general TUIs and comparable Git UIs to make for a consistent user experience across similar tools. The core navigation is managed arrow keys, which is a common interaction method for most TUIs and would be intuitive for most users. When there is a diff from version control the prototype is *externally consistent* by placing the local version on the left. The prototype is *internally consistent* between the diff view and the template and generated file pair view because in both, the local tracked file is on the left.

Discoverability—One of the first partial weaknesses identified in the heuristic evaluation, was that the most relevant *yadm* features were not readily available. With the redesign, the core feature set used for most interactions is now made more clear while there is no change to the more complex features. The TUI doesn't need any separate documentation because it is either self-explanatory based on the user's experience with similar tools or clearly labeled in the help bar along the bottom.

Gulf of Evaluation, Constraints, and Tolerance—The major issue with *yadm's* current design is that errors are very difficult to recover from. This is not easy to address, but there are design changes that can help. The prototype makes two core changes to address this. First, the interface polls for updates so that the current upstream and local state are known at all times that TUI is open. Secondly, when the local and remote states diverge in a way that would make recovery difficult, the TUI informs the user of the possible issue and directs them to a dedicated Git UI that can more adequately resolve the issue. Now, when opening *dotui*, the application launches to the "Status" view where the actionable Pull, Push, or resolve-manually message is shown in plain language.

Future Work—There are several areas where the interface could be improved. One of the areas not addressed with this initial design is the onboarding to an existing *yadm* repository. The initial clone is nearly guaranteed to have some conflicts with the local files, so there could either be changes to *yadm* or a new feature for the prototype to make this step more predictable for the user or generally better align with their mental model.

New Directions—One of the other ideas that was requested during the needfinding stages and explored while brainstorming was a way to compose snippets together from multiple sources. This would require a substantial redesign and is not a trivial problem, but improvements to visualization of templated-files and a better understanding of the needs of this area would help refine possible approaches. There are is at least one product working in this space that was recommended by a participant.

6 QUALITATIVE EVALUATION PLAN

Prototype Adjustments—The prototypes presented for this assignment are possibly too precise and detailed for the stage of development, but were necessary to meet the constraints of the assignment. For evaluation, the prototypes would be redrawn as abstract wireframes where the color scheme, icons, and font selection are excluded in favor of focusing on core features, such as layout, functionality, and interaction methods. By downgrading the prototype fidelity, participants would be much more likely to critique the design that looks incomplete rather than the current high-fidelity prototype because of Social Desirability bias.

Goals—The goals of the evaluation are to determine if the proposed prototype would be something that the participant would use and to understand the reasoning for either choosing to use the prototype or choosing not to. The qualitative feedback would inform the next iteration of the design cycle as the prototype is refined to best address user tasks, needs, and context.

Interview Logistics—The planned evaluation would be a live demonstration coordinated by two interviews and a single participant at a time. One interviewer will be leading the interview by asking questions, while the second takes notes. At least five participants should be interviewed and participants should have a range of prior experience to best represent the pool of potential users. The participants should be recruited from technical groups on LinkedIn and Reddit. Like Survey 1, each participant should complete a few background questions at the start of the interview by providing their job title, how often they have used a terminal in the past week, and how they manage dotfiles currently. The total interview is expected to take between 15 to 25 minutes.

The interview would be conducted remotely with a web-based whiteboard application that the interviews and participant can see and draw-on. The video may or may not be recorded based on participant comfort, but screenshots of the white-

board will be captured when appropriate during the interview and at the end of every interview. The whiteboard would be prepared with one page per Card so that the interviewer can simulate moving between views by changing the visible card. The interview prompt is that the participant is setting up a new laptop at work and needs to figure out how to use the tool to sync local file changes then make a change in *.config.omp.json*.

1. "Describe what you see in this screen."
2. "Is everything on the screen legible?"
3. "What kind of additional labels or help messages would you add?"
4. "What next action would take? What button or mouse movement would that involve?"
5. "What information was missing on this screen?"
6. "What information was not necessary?"
7. "What did you dislike about this prototype?"
8. "What did you like?"
9. "How would a tool like this fit into your existing workflows?"

The lead interviewer will step through each card and ask the above questions. The interviewer will be sure to reiterate that the user can draw on the prototype and provide feedback visually. The interview should reinforce this by also making small and major changes by drawing on the card in the whiteboard as the open-ended discussion continues. The interviewers should minimize how much time they explain the interface and instead capture notes on any difficult areas that will be addressed in the next design cycle. Once the interview is complete, the participant will complete a survey where they can provide feedback on the interview and rate the prototype on a a five point scale for how likely they are to use it if available and how likely they are to use if their changes were implemented.

Biases—As described above, the primary concern is Social Desirability bias. It will be clear to participants that we had some role in designing the prototype and they may be tempted to provide only non-critical feedback. There are two primary ways the evaluation plan has been designed to minimize this bias. First, the planned questions specifically ask for critical feedback. Second, lower-fidelity card prototypes will be created specifically for these interviews so that participants will feel more empowered to suggest changes. Third, the participant will be asked to quantitatively rate the prototype and their contributions.

To address *Confirmation Bias*, the quantitative post-interview survey will help ensure that the user feedback is properly coded according to the quantitative result. The interview questions are provided to ensure that a base set of qualitative data is captured between subjects, but left open ended because of the stage in the design lifecycle. Finally, the note taker should be someone who has a neutral or different opinion on the topic to the lead interviewer so that the notes best represent the user opinions.

7 REFERENCES

1. Invergo, Brandon (May 2012). "Using GNU Stow to manage your dotfiles". In: URL: https://brandon.invergo.net/news/2012-05-26-using-gnu-stow-to-manage-your-dotfiles.html.
2. Athalye, Anish (Aug. 2014). "Managing Your Dotfiles". In: URL: https://www.anishathalye.com/2014/08/03/managing-your-dotfiles/.
3. Walladge, Samuel (Aug. 2017). "Dotfiles configuration management survey results | Samuel Walladge". In: URL: https://www.swalladge.net/archives/2017/08/07/dotfiles-config-survey/.
4. Borkiewicz, Filip (Feb. 2019). "Dotfile madness". In: URL: https://0x46.net/thoughts/2019/02/01/dotfile-madness/.
5. Byrne, Tim (2022a). "Getting Started - yadm". In: URL: https://yadm.io/docs/getting_started#.
6. Byrne, Tim (2022b). "TheLocehiliosan/yadm: Yet Another Dotfiles Manager". In: URL: https://github.com/TheLocehiliosan/yadm.
7. Glovier, Joel (2022). "General-purpose dotfiles utilities - dotfiles.github.io". In: URL: https://dotfiles.github.io/utilities/.
8. Hahn, Evan (May 2022). "A decade of dotfiles". In: URL: https://web.archive.org/web/20220719030747/https://evanhahn.com/a-decade-of-dotfiles/.
9. McGugan, Will (Apr. 2022). "Textualize/terminal-survey". In: URL: https://github.com/Textualize/terminal-survey.
10. Payne, Tom (2022). "Comparison table - chezmoi". In: URL: https://www.chezmoi.io/comparison-table/.

A APPENDICES

A.1 Needfinding Survey Results

Below are the raw results of the survey here. Note that the second question was added after realizing that participants weren't expanding on why they chose Other. Additionally, *Ansible* and *Plain git* were added to the first question after the 10th response based on survey feedback. Although not needed for this paper, there are extensive recent surveys that capture a wider breadth of the tools developers use (Walladge, 2017). There is also a survey that could be useful conducted for *Textualize* (McGugan, 2022).

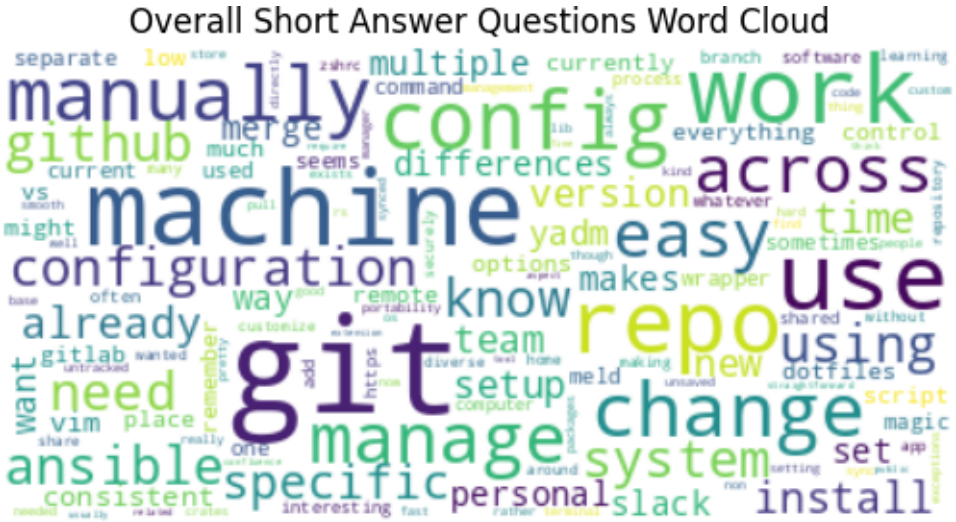

Figure 12—Word Cloud of selected Qualitative responses. Turned out to be not as useful as expected. Removed words include: "dotfile dotfiles file files yes sure nope n/a na via"

- Which software application(s) do you currently use to for managing "dotfiles" (Desktop configuration files, such as ".bashrc" or " /.config/nvim/init.vim")?
  1. Custom scripts
  2. yadm
  3. Custom scripts;Mac Time Machine or other computer-wide backup and restore software
  4. Other
  5. Other
  6. yadm;Dropbox
  7. Other
  8. GNU Stow;Custom scripts

9. Custom scripts
10. Other
11. Other
12. Custom scripts;Other
13. GNU Stow;Other
14. Plain git
15. Plain git;Custom scripts
16. Dropbox;Plain git;Custom scripts
17. Plain git;Custom scripts
18. Other

- If you selected "Other" or "Custom Scripts," can you elaborate on what tools you use?
  1. *Not part of survey when submitted*
  2. *Not part of survey when submitted*
  3. *Not part of survey when submitted*
  4. *Not part of survey when submitted*
  5. *Not part of survey when submitted*
  6. *Not part of survey when submitted*
  7. *Not part of survey when submitted*
  8. *Not part of survey when submitted*
  9. *Not part of survey when submitted*
  10. *Not part of survey when submitted*
  11. *Not part of survey when submitted*
  12. *Not part of survey when submitted*
  13. *Not part of survey when submitted*
  14. *Blank*
  15. Ansible
  16. *Blank*
  17. It's just a bash script that install and configures stuff.
  18. I have a friend who wrote a really cool tool for this called meld. https://lib.rs/crates/meld-config-manager
- What convinced you start managing dotfiles?
  1. I wanted a consistent user experience across my various machines.
  2. Kyle making it easy to do.
  3. I've wasted tons of time not being organized without it.
  4. Complexity

5. New environments
6. standardizing work and projects
7. Wanted to share my .zshrc between work and home computers.
8. my vim configuration got long
9. career
10. I have to for maintaining configurations
11. Consistent configs across many instances I manage
12. starting to customize my vim & zsh setup
13. Ability to use on different computers
14. I work on a team that requires a consistent .gitignore file to remove bloat from our git repository.
15. consistency across my systems
16. No
17. Save time when changing machines. Avoid losing my dotfiles
18. Literally writing a rust app to do this

- What is your favorite feature of your preferred dotfile management application (or why did you select it over others)?
  1. It's hand rolled and extremely barebones, so I know everything that it does.
  2. Similarity to git, no learning new commands. Manages files in place, no need to move them around.
  3. flexibility and feature extensions
  4. Ansible manages the system too
  5. Git or svn — version controls
  6. Manage your dotfiles across multiple diverse machines, securely.
  7. I just put it in a publicly accessable github repo, then git it from there from whichever machine I'm on.
  8. portability
  9. more control
  10. VSCode because I already use it for code editing
  11. Out of the way and low maintence.
  12. My preferred application for dotfiles management is git. It makes tracking changes easy, and I'm familiar with it already. It does everything I currently need. Syncing is easy via e.g. GitLab, GitHub, and/or own server.
  13. Simplicity, and there wasn't many options around when I started
  14. git is fast, straightforwards, works with the terminal and has a free, online presence, i.e. github.

15. I can manage the low-level details myself, and understand how to troubleshoot it
16. Common viewer
17. I use git and a custom script because it felt easy to do so. I haven't spent much time looking for alternatives.
18. It can use a flashdrive, a single file, or a remote

- What challenges do you face when managing dotfiles (or what made it difficult to start)?
  1. I didn't want to introduce any dependency that I wasn't already using, which limited options.
  2. Differences between machines - esp. work vs personal.
  3. honestly, didn't know what aspects were important. didn't know what I didn't know.
  4. Learning roles
  5. Finding a good place to store them
  6. Manage your dotfiles across multiple diverse machines, securely.
  7. Configuration setting specific to a given machine. I.e. configuration specific to my home PC which won't work elsewhere.
  8. yak shaving
  9. version control
  10. Hard to remember where the files are located
  11. Adding exceptions to set or ignore some config options on remote machines.
  12. Initial setup of git might be slightly confusing (setting up a bare repo, creating a custom 'dotfiles' alias wrapper for git). Having non-committed changes makes updates difficult (I have to either commit or stash), but that's actually good design as it makes it hard to destroy unsaved changes. However, it would be technically possible to try to pull and merge if there are no changes in same files as current machine has unsaved changes. But maybe I just don't know how to git properly.
  13. Not much, the most annoying thing is having untracked files for system specific usage
  14. Making sure that other team members get the same file in their system and that they're in sync.
  15. differences across platforms, e.g. Ubuntu vs RHEL vs OSX
  16. System generated filename maintenance

17. I sometimes forget to keep them in sync. I manually have to separate my personal and work config.
18. None now, meld is buttery smooth

- If you have used yadm, what do you like or dislike about the tool? (Optionally, if you haven't used yadm, but want to skim the README, feel free to leave feedback here as well)
  1. I don't like how much magic it does. I would prefer a solution that was a little more explicit in what it was doing, so people other than the creator could follow along well.
  2. Like: that it uses git commands and files stay in place. Dislike: no way to differentiate between machines.
  3. n/a
  4. *Blank*
  5. First I heard of this
  6. *Blank*
  7. Never used before. From reviewing the README, not sure what it does besides provide some kind of encrypted backup of dotfiles.
  8. * would rather use git directly and only learn one command * os-specific config seems weird; I would rather use branches.
  9. *Blank*
  10. *Blank*
  11. *Blank*
  12. Hmm, haven't used and likely won't. Seems interesting, though. Mostly the same as my current setup, but with the git wrapper already made for me. Encryption seems...interesting. Wouldn't trust it. Most importantly, I'm not willing to install 3rd party packages to every system if I can achieve the functionality without it.
  13. *Blank*
  14. *Blank*
  15. I have not used yadm
  16. *Blank*
  17. It looks quite interesting, I might give it a go eventually.
  18. *Blank*
- Which method(s) do you use for tracking changes and/or syncing?
  1. Github, GitLab, BitBucket, or other hosted Git;Local Git
  2. Github, GitLab, BitBucket, or other hosted Git

3. Github, GitLab, BitBucket, or other hosted Git;Local Git
4. Github, GitLab, BitBucket, or other hosted Git
5. Github, GitLab, BitBucket, or other hosted Git;SVN or other non-Git VCS (version control system);rsync
6. Github, GitLab, BitBucket, or other hosted Git;Local Git;Dropbox
7. Github, GitLab, BitBucket, or other hosted Git;Local Git
8. Github, GitLab, BitBucket, or other hosted Git;Local Git
9. Github, GitLab, BitBucket, or other hosted Git;Local Git
10. Github, GitLab, BitBucket, or other hosted Git;Other
11. Github, GitLab, BitBucket, or other hosted Git
12. Github, GitLab, BitBucket, or other hosted Git;Local Git
13. Github, GitLab, BitBucket, or other hosted Git
14. Github, GitLab, BitBucket, or other hosted Git;Local Git
15. Github, GitLab, BitBucket, or other hosted Git;Local Git
16. Github, GitLab, BitBucket, or other hosted Git;Local Git;Dropbox
17. Github, GitLab, BitBucket, or other hosted Git
18. Other

- If you selected Other, describe what you use here:
  1. *Blank*
  2. *Blank*
  3. n/a
  4. Ansible
  5. *Blank*
  6. *Blank*
  7. N/A
  8. *Blank*
  9. *Blank*
  10. Google Drive
  11. A variation of this: https://www.ackama.com/what-we-think/the-best-way-to-store-your-dotfiles-a-bare-git-repository-explained/
  12. *Blank*
  13. Tangled emacs org files
  14. *Blank*
  15. *Blank*
  16. *Blank*
  17. *Blank*

18. https://lib.rs/crates/meld-config-manager

- If you track changes, what do you use to stage changes and resolve conflicts?
  1. The git diff/merge tools
  2. *Blank*
  3. n/a
  4. Git
  5. Only my version so I just revert
  6. *Blank*
  7. It's a pretty fast and loose management of dotfiles if I'm being honest. Staging and resolving changes is far more elaborate then what I do. I basically copy-and-paste, whatever I want, and omit/delete what I don't.
  8. git and vim
  9. Git
  10. *Blank*
  11. *Blank*
  12. git, nvim
  13. Git
  14. git terminal
  15. gitlab/vi ;)
  16. GitMerge
  17. I manually resolve them. I don't find myself running into merge conflicts that often though.
  18. *Blank*
- What operating systems do you currently have dotfiles installed on?
  1. Mac;Linux
  2. Mac
  3. Mac
  4. Mac;Linux
  5. Mac;Linux
  6. Linux;Windows
  7. Mac;Linux
  8. Mac;Linux;Windows
  9. Mac;Linux;Windows
  10. Mac;Linux;Windows
  11. Mac;Linux;Sync or otherwise run on a Remote SSH Session
  12. Linux;Sync or otherwise run on a Remote SSH Session

13. Mac;Linux
14. Mac;Windows
15. Mac;Linux
16. Mac;Linux;Windows
17. Mac;Linux
18. Mac;Linux

- On how many computers (virtual or physical) do you currently have your dotfiles?
  1. 2
  2. 2
  3. 2
  4. 3
  5. 5+
  6. 5+
  7. 3
  8. 4
  9. 2
  10. 2
  11. 5+
  12. 5+
  13. 2
  14. 2
  15. 5+
  16. 4
  17. 3
  18. 5+
- If you sync dotfiles between computers, how do you manage configuration differences between machines?
  1. store everything in a remote git repo, then treat it like ay other code base.
  2. Just using yadm on work macs for now so no differences.
  3. don't have to, machines are nearly identical
  4. Ansible variables
  5. Test for release file specific to os
  6. *Blank*
  7. Pretty poorly. I mainly really on using similar machines for portability.
  8. each computer is on it's own branch

9. I don't
10. *Blank*
11. Exceptions in my configs based on the properties of the machine
12. I have a non-synced, but always sourced (if exists) .device.profile dotfile. It contains any needed differences. Should that not be sufficient, I'd just change dotfiles and perhaps create a new git branch for them.
13. Load untracked files from the config
14. Always pull and merge from the main branch on github
15. Ansible playbooks
16. *Blank*
17. *Blank*
18. https://lib.rs/crates/meld-config-manager

- How do you identify what dotfiles should be managed?
  1. Anything where I might have trouble getting it back to it's current state if I lost the file.
  2. For work: files that do not change per-person and otherwise would require individuals to set up.
  3. sub-folders
  4. Whatever apps I'm using
  5. I add to list what I need to remember
  6. in a separate Git directory, usually called /.dotfiles
  7. The ones I use most often, i.e. .zshrc and .vimrc
  8. manually
  9. I don't
  10. I have to remember
  11. Any tool I use in multiple locations
  12. manually, I add each file separately if I think it's needed. I usually only add files that I've changed manually.
  13. Software I use and/or customize a lot, if config was complicated, or if I intend to use it on a separate machine
  14. Whatever files are in the repository needs to be managed
  15. I have a base set of them defined by function in Ansible (e.g. git-related, shell-related, etc.)
  16. With their extension and type
  17. Foreseen reusability.
  18. https://lib.rs/crates/meld-config-manager

- How frequently do you use a terminal, such Windows Command Prompt or Mac Terminal?
  1. More than 5 times per day
  2. More than 5 times per day
  3. Between 1 and 4 times per day
  4. More than 5 times per day
  5. More than 5 times per day
  6. Between 1 and 4 times per day
  7. More than 5 times per day
  8. More than 5 times per day
  9. A couple of times a week
  10. Between 1 and 4 times per day
  11. More than 5 times per day
  12. More than 5 times per day
  13. More than 5 times per day
  14. More than 5 times per day
  15. More than 5 times per day
  16. Between 1 and 4 times per day
  17. More than 5 times per day
  18. More than 5 times per day
- Have you installed dotfiles from another user for work or personal computer? If so, what did you like or dislike about that process?
  1. Yes. I disliked the magic and the process of combining personal dot files with others.
  2. No.
  3. yes, have pulled from github repos
  4. No
  5. No
  6. *Blank*
  7. No, dotfiles are like your favorite hat or hoody. It's not about being the best, it's about having the setup that I want, deficiencies included.
  8. n/a
  9. no
  10. *Blank*
  11. *Blank*

12. Nope, I don't run random scripts and don't have the need to install anyone's dotfiles
13. *Blank*
14. Yes, I like the process of using git since it's straightforward and easy.
15. Nope
16. *Blank*
17. Yes. I often find myself forking their files and nitpicking what I need. I like time savings.
18. Yes, process is super smooth

- Do you share snippets intended for dotfiles or computer configuration at work? If so, are they shared manually or do you have a tool? Are they in version control and/or synced automatically? Roughly how many people are involved?
  1. They are synced through yadm for a team of 10
  2. Answered from the perspective of work! Not currently managing dotfiles on my personal machine but planning to use yadm.
  3. n/a
  4. Shared via GitHub repo and Gists manually
  5. Teams wiki
  6. *Blank*
  7. Occasionally I will share some dotfile configuration with a coworker if I think they'll benefit. It's a transfer directly between two people.
  8. via slack
  9. no
  10. shared manually
  11. Manually
  12. I'm not sure whether I understood the question. I have an installation script that sets up the 'dotfiles' wrapper and clones the git repo. Also installs relevant packages. My dotfile git repo is public.
  13. *Blank*
  14. *Blank*
  15. Typically I use https://onetimesecret.com/ for one-off things with potentially sensitive data. Otherwise, Ansible (via gitlab)
  16. *Blank*
  17. Yes, sometimes shared in Slack. Some are documented in Confluence. Some are version controlled; Sometimes in a generic repo for multiple dotfiles, other times the dotfiles are specific to a repo/service.

18. Yes

- If you answered the above question, what works or doesn't work about the current way configuration is shared?
  1. It works really well other than the "magic" aspect
  2. Currently no way to specify what software is required. We could do with some way to install software programmatically which would complement the dotfile sharing.
  3. n/a
  4. Public repo makes it work
  5. Ugly
  6. *Blank*
  7. If they don't get it, don't see the value, or don't want that configuration.
  8. already using slack so low friction
  9. N/A
  10. it's fine because it's rarely done
  11. *Blank*
  12. It's easy to install dotfiles on new machines.
  13. *Blank*
  14. *Blank*
  15. OTS is manual (bleh), Ansible is kind of a pain to manage secrets for (Vault)
  16. *Blank*
  17. Slack and Confluence can get outdated. Repo works fine but there's no discoverability i.e. you have to know it exists.
  18. *Blank*
- What is your job title or professional role?
  1. Senior Backend Engineer
  2. Senior Natural Language Processing Engineer
  3. analytics engineer
  4. Cloud Security Engineer
  5. Lead engineer
  6. analyst
  7. Software Engineer.
  8. Principal Data Science Consultant / Statistician
  9. associate software engineer
  10. Software engineer
  11. Principal Systems Engineer

12. Not relevant
13. Software engineer
14. Software Engineer
15. Lead Staff Software Engineer
16. Technical LEad
17. Software Engineer
18. Software Engineer

- Thank you for completing this survey! Please leave any general comments about managing dotfiles or this survey here.
  1. None
  2. Answered from the perspective of work! Not currently managing dotfiles on my personal machine but planning to use yadm.
  3. would be good to have ideas summarized in a post
  4. Should include Ansible in initial list
  5. Figure out a way for vim to be happy with different os's and vim version
  6. would like to learn more about them but don't have time
  7. Very good survey. Will have to look more into "professional solutions" to dotfile management.
  8. I did a talk on dotfiles at Kansas linux fest a few years back. If you would like to collaborate / have a professional statistician look at your data my email is nfultz@gmail.com
  9. N/A
  10. Very niche topic, I don't think I've put more than a few minutes thought ever into managing dot files. I learned something today
  11. Terminals are the best
  12. Git alone is a sufficient system, see Atlassian's guide for managing dotfiles using git with a bare repository! Also please post results of this survey to /r/dotfiles. Cheers!
  13. Good luck! Curious about other responses
  14. No comments
  15. Good luck!
  16. Good survey
  17. My naive git+bash setup works so I'm not dying to mess with it but I might look into yadm to see what can be improved.
  18. https://lib.rs/crates/meld-config-manager

A.2 Targeted Needfinding Results

Below are the raw survey results with two manual alterations. Firstly, the first participant saw both multiple choice questions as a single question. They provided the feedback that reading through so many items was difficult. The survey was modified to split the single question into two and results were updated retroactively below; however, for analysis, the options were re-combined. Secondly, the original second participant filled in the fields with only *"none"*, *"no"*, or *"n/a"* including the final question, so their response was excluded from analysis and the below raw results.

- As an open-ended question, what change(s) would you make to yadm?
  1. Make 'yadm clone' work from anywhere. Stop 'yadm add .' from trying to add everything / requiring access to all dirs. Allow defaults in templates for when env vars do not exist + error if env var not found and no default.
  2. I would have yadm be more verbose about what it was doing under the hood. It's a great tool, but feels a little too magicky for me.
- What features would help interacting with git/syncing for yadm?
  1. Better descriptions over which diff is local vs. remote when pulling upstream changes;Better integration with Git applications, such as Gitkrakken, Github Desktop, etc.;Better listing of which files are currently managed by yadm;Easier conflict resolution if local changes conflict with upstream
  2. Better description of what changes to the diff would do to local files;Better descriptions over which diff is local vs. remote when pulling upstream changes;Better listing of which files are currently managed by yadm
- Where could yadm make managing dotfiles easier?
  1. Easier way to regenerate templates when you want to change a variable;Notification of new changes in upstream that should be synced
  2. Better way to discover code snippets that may be useful;Built-in identification of possible secrets in code before allowing a commit;Composability to sync partial snippets from coworkers and peers without syncing all of them;Easier setup of new computers
- If you selected Other(s), what are those?
  1. *Blank*
  2. *Blank*
- Could you explain more about how those changes might better meet your needs?

1. Can be confusing when first setting up to align shared dotfiles with existing local - ‘git reset –hard HEAD‘ seems too forceful. Templates break too easily / don’t error when they should.
2. While syncing dot files is fantastic among collaborators at any project, I would prefer if the entire files didn’t have to be synced. A selective sync/composability would be the killer feature for me.

- As a check that you know yadm, what is one library that can be used for encryption and decryption?
  1. transcrypt
  2. transcrypt

Below are the combined and tallied responses to the two multiple choice questions.

Table 1—Targeted Needfinding Multi-Choice Responses.

Question	Responses (out of 2)
Better descriptions over which diff is local vs. remote when pulling upstream changes	2
Better listing of which files are currently managed by yadm	2
Better description of what changes to the diff would do to local files	1
Better integration with Git applications, such as Gitkrakken, Github Desktop, etc.	1
Easier conflict resolution if local changes conflict with upstream	1
Better way to discover code snippets that may be useful	1
Built-in identification of possible secrets in code before allowing a commit	1
Composability to sync partial snippets from coworkers and peers without syncing all of them	1
Easier setup of new computers	1
Easier way to regenerate templates when you want to change a variable	1
Notification of new changes in upstream that should be synced	1
Difficulty with determining how to use the git CLI commands from yadm (addressed with better documentation, help, etc.)	0
Easier way to undo changes after a mistake	0
Built-in formatter or linter that keeps code consistent and can catch syntax errors on commit	0
Discovery of which configuration files in your local environment are good to track, like which files from VSCode should be synced (and which shouldn't)	0
Easier to make dotfiles operating-system specific	0
Easier way to edit templates where environment variables are used	0
If syncing with multiple computers, easier ways to commit changes and share	0
If using a single computer, automatic syncing and backup	0
Other(s)	0
None	0

A.3 yadm Heuristic Weakness: Gulf of Evaluation

Below captures how to reach a state through common user tasks that is difficult to evaluate and resolve for most users[1].

1 Unless you have this useful guide on hand

1. Configure yadm, add one or more local files (e.g. *yadm add ~/.viminfo*), and create and push to a new Github repository
2. Either uninstall *yadm* by removing the headless git repository (On Mac: *rm -rf ~/.local/share/yadm/repo.git*) or switch to a different computer
3. Start with the same file that was tracked and make any change (e.g. *nano ~/.viminfo*)
4. Clone the Github repo created in step 1 with: *yadm clone git-url*
5. Check the status with: *yadm status*

```
~  > cat ~/.viminfo
       | File: /Users/kyleking/.viminfo
-------------------------------------------------------------
   1   | # This viminfo file was generated by Vim 8.1.
   2   | # You may edit it if you're careful!
   4   |
~  > nano ~/.viminfo
~  > yadm clone https://github.com/KyleKing/dotfiles-yadm-demo
~  > yadm status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    .viminfo

Untracked files not listed (use -u option to show untracked files)
~  > cat ~/.viminfo
       | File: /Users/kyleking/.viminfo
-------------------------------------------------------------
   1   | # This viminfo file is auto-generated
   2   | # Documentation is available online
   3   | # Edited, but will need to be careful!
   4   |
```

Figure 13—Output after yadm clone when local changes differ