Dotfiles for Bash

Dotfiles for Bash

On this page:

Overview

This document covers:

- what happens during X (window login) and terminal (tty) login sessions,
- what dotfiles are used to achieve specific tasks
- some common customizations that can be made
- troubleshooting dotfiles
- dotfiles created by programs or to customize programs

Introduction

The set of files used to describe session initialization procedures and store user customizations are commonly referred to as "dotfiles". These files can be used to customize screen appearance, shell behavior, program specifications, and aspects of your Athena session. Most dotfiles are text files, although some exist in other formats. Dotfiles generally contain one command per line and are stored in your home directory. Dotfiles usually have names that begin with a period, hence the name *dotfiles*. You are given some dotfiles that are necessary for you to be able to login when you get your account.

You may not have even noticed the dotfiles in your account because files that begin with a dot are not listed when the command **ls** is issued. To

view dotfiles use the command **ls -a**. You may have many more or less dotfiles in your account than this example shows. It all depends on what programs you have used and what, if any, customizations you have made to them. Many programs create dotfiles to store information and preferences.

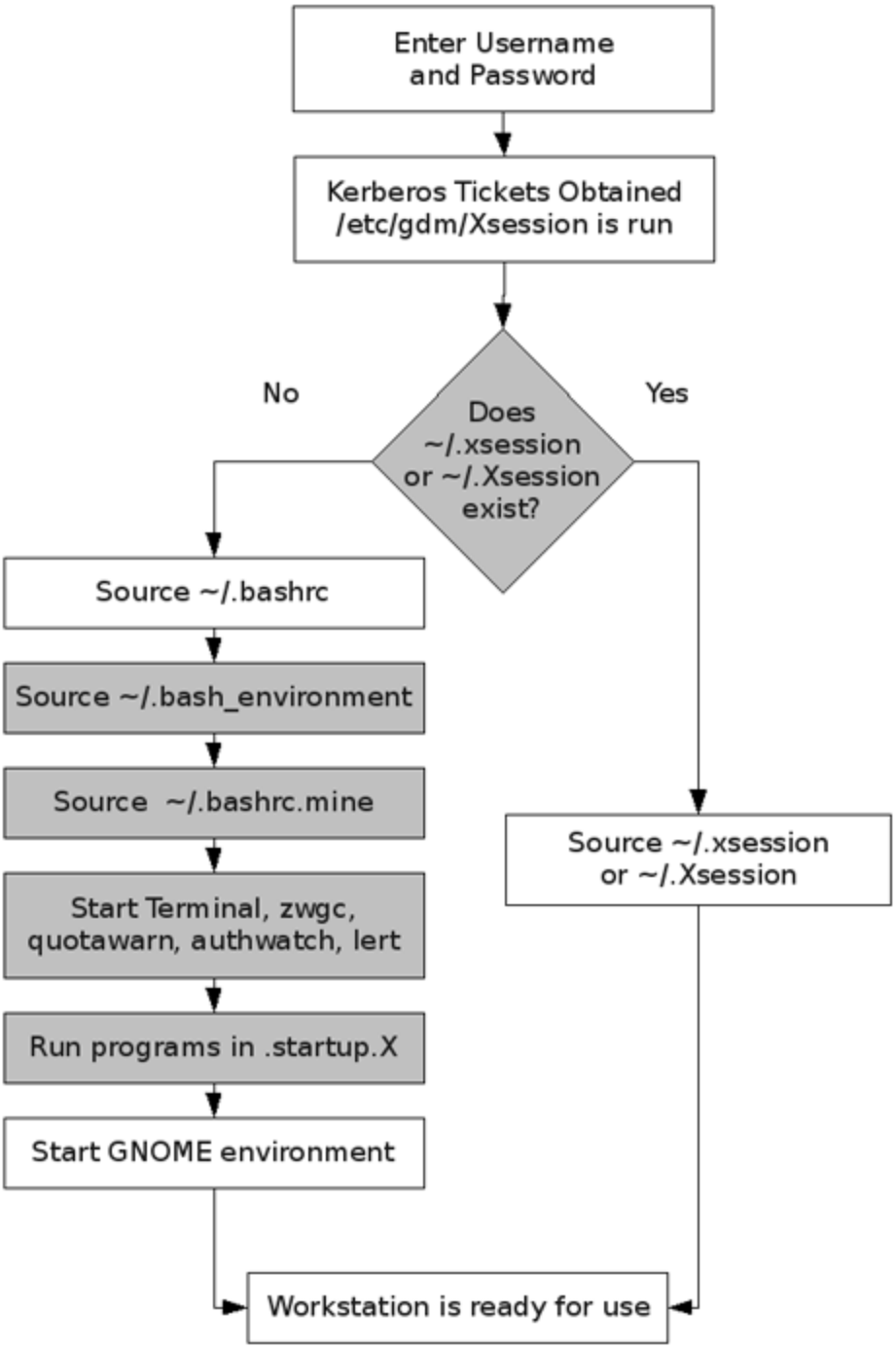
```
athena%
cd ~
athena%
ls -a

.
..
.bash_login
.bashrc
.cshrc
Desktop

.gconf
.gnome2
.login
.logout
.message_times
.mh_profile

.mozilla
.nautilus
OldFiles
Public
www
welcome
```

Login Sequence

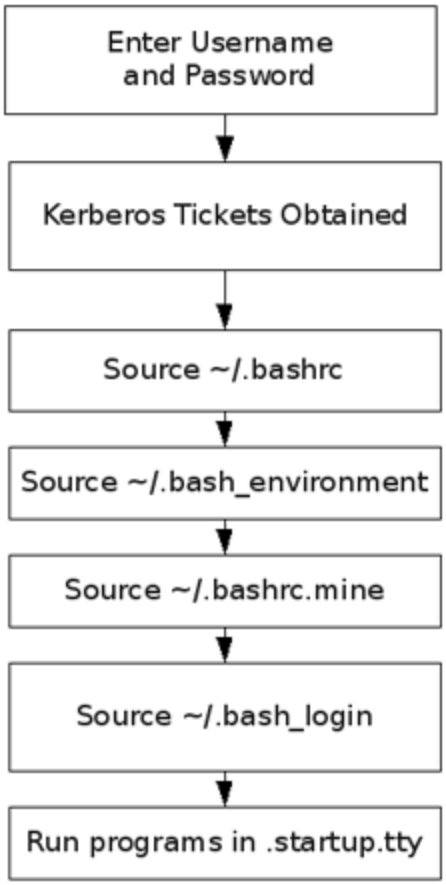
Below is an overview of what happens by default when you log in to a workstation. Some of the files mentioned are not in your home directory. Those files are located in a central location for updates at a future date without you having to make changes to your dotfiles each time there is a slight modification to the system.

Stages of the login process colored gray indicate that that stage is skipped if you choose to login without customizations.

Graphical Login



Terminal Login



Dotfiles Descriptions

The following section provides detailed descriptions of the files and programs associated with session initialization. This includes dotfiles and the programs that source them. In addition it contains links to sample customizations that appear in later sections.

Gdm

Although not a dotfile, **gdm** (GNOME Display Manager) is the program that is running on a workstation that is not in use. It is the program which allows you to enter your username and password, select a specific login session, and perform other actions.

For a standard session **gdm** acquires **Kerberos** tickets and **AFS** tokens, ensures that the user's home directory exists, and then runs the default Xsession script. After a user logs out, **gdm** restarts and waits for the next user to login. **Kerberos** tickets and **AFS** tokens are what authenticate identity and allow access to lockers and files.

Nonstandard sessions are possible from **gdm** by selecting them from the "Session" menu. You can specify *Failsafe GNOME* in which case user customizations files are not sourced and your session is configured as the default session. The files that are skipped in this case are:

- ~/.Xresources
- ~/.bashrc.mine
- ~/.startup.X
- ~/.hushlogin
- ~/.bash_environment
- ~/.xsession

It is also possible to login with terminal mode behavior. This means there are no X windows, just a shell, which is similar to logging in over dialup. More information on this kind of login is available in the section on terminal style logins.

/usr/athena/lib/init/xsession.bash

This file controls much of the login sequence:

1. The XSESSION environment variable is set to indicate an X session as opposed to a terminal style session. This variable is used by later files to determine the proper login procedures.

2. The ~/.bashrc file is sourced.
3. The X Windows system is configured through the ~/.Xresources file.
4. Then a windowmanager is started. The default windowmanager is **Metacity.**
5. Then /usr/athena/lib/init/xsession.bash starts a selection of programs, most of which can be avoided through alterations to the ~/.environment file.
    - Terminal: An initial Terminal window.
    - Start-up: **zwgc** (zephyr), motd (sends a zephyr announcing cluster closings and other info), **quotawarn** (warns users when they exceed or get close to their quota), **authwatch** (warns the user when their Kerberos tickets are about to expire)
    - Lert: Shows messages from accounts, if any, such as account deactivation messages.
6. The ~/.startup.X file is sourced. This file is where programs that are to be run at start-up should be executed.

~/.bashrc

This file simply calls the system-wide user initialization file, /usr/athena/lib/init/bashrc.

> **Warning:** DO NOT modify your ~/.bashrc file as you will not get updates made to the system-wide file. Any customizations you would normally put in your ~/.bashrc file on other systems should go in ~/.bashrc.mine on Athena.

/usr/athena/lib/init/bashrc

This file is called by your ~/.bashrc. It is located in a central place so it can be updated for system updates without requiring any user changes. It does two main things in the process of setup:

1. Environment Setup – sets UNIX environment variables, command search path, etc. This step is usually preformed only once a session.
2. Shell setup – sets shell variables/aliases, command prompt, etc. This step is preformed every time a new **shell** process is started (ex: whenever a **Terminal** window is started)

The file sets the following shell variables:

- **umask 077**
  set the default file permissions such that only the user can read and write files created in this session
- **ulimit -S -c 0**
  keeps programs from dumping core files, which can quickly use up your quota
- **HOME**
  set your home directory ~
- **PS1**
  set the prompt in your **Terminal** window
- **PATH**
  Where the shell should look for programs

Then the following environment variables are set:

- **MORE**
  Makes the more command work.
- **EMAIL**
  Sets your default email address to `username@MIT.EDU`, as opposed to `username@hostname.MIT.EDU`
- **EDITOR**
  Set the default editor to **emacs**
- **MANPATH**
  Set the default search path for man pages
- **ATHENA_SYS**
  Used by add when adding lockers to modify the path and manpath appropriately.
- **PRINTER**
  Sets the default printer for your workstation (if applicable)

Sources ~/.bash_environment to set user specified environment variables

Next, several aliases are set for your convenience:

- **renew**
  Renews your **Kerberos** tickets (that expire in 10 hours).
- **logout**
  logout from any shell, not just the initial one
- **add**
  attaches a locker and adds the locker to the path and manpath
- **setup**
  adds a class locker and runs appropriate related start-up scripts
- **setup_x**
  same as **setup** but for X windows sessions

- **setup_tty**
  same as **setup** but for terminal sessions
- **remove**
  undoes everything **setup** did

Finally your ~/.bashrc.mine is sourced to load user specified customizations

~/.bash_environment

The ~/.bash_environment file is a user created file. Commands that set shell and environment variables and that add frequently used lockers to your path are placed here. In addition this file can be used to skip some standard startup activities, customize printer options, and set default editor selections. All customizations made in this file are done only once a session and should be appropriate for both window and tty sessions.

~/.bashrc.mine

The ~/.bashrc.mine file is a user created file where custom variables for your shell are set, aliases are created and other customizations specified. This file was created on Athena to encourage users not to alter the .bashrc file. The only shell variables that should not go here are the ones that control the session (session controlling shell variables are located in ~/.bash_environment because they need to be sourced earlier in the login process).

~/.Xresources

The xsession runs **xrdb** on ~/.Xresources if the user has created the file and it exists in the home directory. This loads the preferences for X applications. ~/.Xresources can be used to change colors, specify locations, fonts, and other application specific settings. X resources are stored in a database, not as variables.

~/.startup.X

This user created file is sourced near the end of the session initialization by the xsession. ~/.startup.X is where you can specify programs you want to run upon start-up. This file will only be sourced in X window sessions – for terminal sessions, the equivalent file is ~/.startup.tty. Most commands run from .startup.X should have an ampersand (&) after them unless the command will finish quickly (**znol**, **fortune**, etc...). Otherwise the initialization will come to a halt until the program exits; you probably do not want this to happen. **emacs**, **xclock**, **gaim** and **firefox** are programs that will not exit without user interaction. This means your session initialization will not finish and logging out may be difficult.

~/.logout

This file is run at the end of your session and can be used to run programs or scripts just before the session is over. However, since user input is halted at logout, the scripts run should be self-contained with no stdin.

Common Customizations

Do **NOT** make changes to ~/.bashrc or ~/.login. **Any changes to these files are made at your own risk.** The session activity can not be guaranteed or supported as you will not receive changes to these files automatically. This may render you unable to login and unable to gain assistance to do so.

This section includes warnings about common errors and useful info about the dotfiles that can be used to customize an account. Read all the way through this section before making changes. It is important to know what kinds of customizations go in all the files before changes are made. This ensures customizations will be put in appropriate locations. Also be sure to read the appendices on testing dotfiles and how to recover from errors in dotfiles at the end of this section. It will save you a lot of time and effort. Printing out a copy of this document before trying to customize your dotfiles is recommended; it'll be handy to have a hard copy around if you break things such that you can't log in. There is also a chart in Appendix K that lists common customizations and what dotfiles you use to make those changes.

Many of these files do not already exist in your home directory. To create them use an editor, such as **emacs** or **gedit**, and save the file in your home directory with the appropriate name.

```
emacs ~/.dotfile
athena%
emacs ~/.bash_environment
```

Your Path

As mentioned above you can customize your path. The path is used when you try to run executable files (programs) to find the program. Things listed first in your path will be where the system looks first for the program you are trying to run. If there are multiple versions of the program, the first version found will be the one used. Be sure to take this into account if you decide to customize your own path. Your path is controlled by the environment variable **$PATH**. To view what is currently in **$PATH:**

```
athena%
echo $PATH
```

An example path:

```
. /usr/athena/bin /mit/games/arch/i386_deb40/bin
```

The dot suggests that the system look in the current directory first, then to the Athena default path (usr/athena/bin). Finally the system looks in the appropriate bin for your workstation in the games locker.

If you wish to add lockers or specific directories to your path permanently, you can use your ~/.environment file as described below to customize your path using the **add** command.

~/.bash_environment

The ~/.bash_environment file is used to add lockers, skip some start-up activities, and set environment variables. Remember all commands are case sensitive so be sure to copy capitalization exactly. Here are a few simple customizations you may wish to put in your ~/.bash_environment file:

skip_x_startup =t

skip_tty_startup=t

skip_initial_xterm=t

skip_quotawarn=t

skip_authwatch=t

export LPROPT="-h -z"

add *lockername1 lockername2 lockername3 /var/local/directory...*

add -f *lockername4*

skips startup commands that start zephyr, check quota, and send message of the day

same as skip_x_startup for terminal sessions.

should be used carefully; you most likely don't want to leave yourself without an **xterm** or the ability to get one at start-up, so only use this command if you plan to start another **xterm** in your ~/.startup.X file or you can get an **xterm** through the panel or your windowmanager.

disable the warning about being close to or exceeding your AFS quota. You are encouraged to reduce your quota usage or request an increase instead of using this variable.

disable notification of Kerberos ticket expiration.

will prevent the printing of the header page when you print (to conserve paper) and will enable the sending of zephyrs to you when your document is finished printing or if there is a printer error preventing printing.

is how to add lockers and directories to your path, manpath and get access to files in those lockers you have permission to read. It is much faster to list all the additions on one line so the script only runs once.

The **-f** option of **add** adds the locker or directory to the front of your path. Use this only if you want to run software out of a locker instead of a version installed locally.

There are many more environment settings than those listed above, but it is beyond the scope of this document to cover them all. To find out more about environment settings and other dotfile related topics, take a look at Appendix G.

Commands that you want run each time you login, for both window and tty sessions, should generally be put in ~/.bash_environment. However, this is run before ~/.bashrc. If what you want to run depends on these files being sourced, the command should be put in ~/.bashrc.mine.

~/.bashrc.mine

This file is typically used to set shell variables and aliases, but can be used for other customizations. Keep in mind that this file is executed AFTER ~/.bash_environment.

For example, if you wanted to alias the **zwrite** command to **z** to make it shorter to type, you could do:

```
alias z=zwrite
```

In general it is convenient to shorten commands you commonly use. If you have a lot of aliases, you may want to create a file ~/.aliases to put them in. Then all you need to do is put the following in your ~/.bashrc.mine file and the ~/.aliases file will be sourced.

```
athena%
source ~/.aliases
```

Note: Anything that runs in ~/.bashrc.mine should never generate output, as that can cause any number of problems, including the inability to use some secure file transfer software. For example, if you wanted to run `znol -l` to get a list of your friends who are logged in, you would put that in ~/.startup.tty or ~/.startup.X, not ~/.bashrc.mine.

~/.Xresources

There are more things that can be done with the ~/.Xresources file than can be covered in this document. The Xresources file contains information on how you want the programs you run to look and behave. The general form for entries in your ~/.Xresources file is:

*program*resource*subresource*...*subresource: value*

Capitalization is very important as all Xresources are case sensitive. Some capitalization in Xresources is not intuitive. You can find information about the Xresources for most programs by typing:

```
man programname
athena% man zwgc
```

There are a few common types of customizations you can make in your Xresources files. Some of them are changing the colors of applications, specifying locations where applications will appear, the fonts applications will use, and other program specific customizations. Here are some examples using **zwgc** (**zephyr**).

```
zwgc.style.message.personal*geometry: -0-0
zwgc.style.message.white-magic*geometry: -0+0
zwgc.style.help*geometry: +0+0
zwgc.style.login*geometry: +c+c
```

This set of commands will cause windowgrams (zephyrs) that are personal messages to appear in the lower right hand corner of the screen. Windowgrams sent to instance "white-magic" will appear in the upper right hand corner of the screen, and windowgrams to class "help" will appear in the upper left. Login messages (generated by znol) will be centered in the screen.

The geometry values can be set to numbers other than zero. The first number corresponds to the horizontal distance from the corner and the second number corresponds to the vertical distance from the corner. Thus if you set the geometry for personal zephyrs to -10-0, the windowgrams would appear flush against the bottom of the screen 10 pixels form the right hand side of the screen. Be careful when setting geometry because the measurements are all from the sides (except +c+c which is a special case for centering) and screen sizes vary.

Zephyr is not the only program for which geometry can be specified. Geometry can be specified for most any program. **Zephyr** is one of the most complicated because you can specify subresources of instance names, classnames and usernames. To use geometry for most other programs:

```
programname*geometry: 20x20+0+0
```

The 20x20 refers to the size of the window. It is important to be careful when setting the size of programs, such as zwgc, that vary in size since you could lose content. The 20 actually means 20 pixels, so this would be a very small window.

Screen location is not the only thing that can be specified in Xresources. Another popular customization is to change the color of programs on your screen:

```
zwgc.style.help*foreground: white
zwgc.style.help*background: green
XTerm*foreground: black
emacs*foreground: blue
XTerm*background: blanchedalmond
emacs*background: black
```

This will cause your **xterm** to have black text and a background color of blanched almond. **Emacs**, on the other hand, will have a black background and blue text. Windowgrams sent to class or instance help or from someone with the username help will be white on green. To find a list of colors available, consult the file `/usr/share/X11/rgb.txt`

There are a nearly infinite number of options with Xresources, here are a few more that will give you a thin scrollbar on your **xterm** that saves up to 500 lines of text, and an **xclock** with a different font.

```
XTerm*scrollBar: true
XTerm*saveLines: 500
XTerm*Scrollbar*thickness: 5
xclock*font: -*-helvetica-*-r-*-*-14-140-*-*-*-*-*-*
```

~/.startup.X

This file should contain whatever commands you would type at your prompt, one per line, for the programs you wish to be started upon initialization. Some common programs to start are **xclock** or **emacs**. Note that starting anything time consuming/computationally intensive, such as a web browser, can cause your initialization sequence to take a long time. If you only use a program occasionally it is generally better to start the program yourself when you want to use it.

The session initialization sequence will come to a halt until the program has exited. If you are simply checking the weather, that will finish quickly and no ampersand is needed. However, if you are starting a web browser all initialization will halt until you exit the browser. Put the ampersand (&) after the command to solve this problem.

```
programname &
firefox &
```

~/.logout

This file runs after you type **logout** to end you session. A common customization is to:

```
cd /mit/$USER
(echo "*** Logged Out" `date` on `hostname` "***") > .plan

cat .plan
```

The ~/.logout file commands must apply to both X and terminal sessions. ~/.logout does not run within an **xterm** window so commands such as **clear**, **resize**, **exit**, etc will fail. Also, do not use commands that require user input since the ~/.logout file is run from the console window in X sessions and there is no method for user input.

~/.hushlogin

You do not need to put anything in this file. ~/.hushlogin exists only to supress progress messages in the course of your login. The easiest way to do this is to issue the command:

```
athena%
touch ~/.hushlogin
```

Please note that this command will create the file and you need do nothing more to your ~/.hushlogin file. A word of warning: it can be helpful, when working with your dotfiles, to have the progress messages so you can tell where you are having problems if your dotfiles break when you try to login. You might not want to create this file until you are done with other customizations and are sure your customizations work. Progress information can be helpful in determining where problems are in the initialization sequence. Also, the "message of the day" (motd) will not be sent with a quiet login. If you decide to create a ~/.hushlogin file you should restore the motd as the messages are important messages from accounts. Add this command to your ~/.startup.X and ~/.startup.tty files:

```
get_message -new -zephyr
```

Other Dotfiles

~/.plan

~/.aliases

~/.away

~/.emacs

~/.mailcap, ~/.maillock and ~/.mh_profile

~/.meetings

~/.mozilla

~/.newsrc, ~/.oldnewsrc and others

~/.gconf and ~/.gnome

~/.zephyr.subs

~/.zephyr.vars

~/.zwgc.desc

This file is generally used to put information you want people to see when you are fingered. If you want to make information avaliable this way the file must be publicly readable.

File commonly created if you use a lot of aliases to keep the ~/.cshrc.mine file more readable. If you make a .aliases file be sure to put this line in your ~/.cshrc.mine file:

```
source ~/.aliases
```

This file is used by the program **zaway**. ~/.away contains the message that is sent to people who zephyr you when you use the **zaway** program. For information about and how to customize **zaway** issue the following command:

```
athena%
man zaway
```

Used to customize the program **emacs**. For more information refer to the document Emacs on Athena.

These are not user customizable files. These files are used by some mail programs and should not be altered by hand.

This file is used to store information about **discuss** meetings. For more information about **discuss** refer to the document Discuss on Athena.

This directory holds configuration files for **Mozilla** and **Firefox**, including your browsing history, any extensions you may have installed, and your certificates.

These are files created by news readers that contain information about your preferences, what groups you read and other state information. These files are not user customizable in most cases. Read the documentation on your newsreader to find out what files are user customizable. Some news readers create directories that contain their customization files.

Customization directories for the GNOME environment. You should not edit these settings by hand.

Contains information about the **zephyr** classes and instances to which you subscribe. You do not have to edit this file by hand, the zctl add and zctl delete commands write information to this file automatically. Information is stored in the format

Class, instance, recipient

Your file might look a bit like this if you subscribe to instance help, instance b5, and class my-friends.

message,personal,%me%

message,urgent,%me%

message,help,*

message,b5,*

my-friends,**,**

See the Document Zephyr on Athena for more information about **zephyr, zctl** commands and **zephyr** related dotfiles.

Contains the information you provide with the **zctl set** command. This is zephyr's way of letting you set variables and customize its behavior to some extent. Signature and exposure information is kept here.

Describes how windowgrams will appear on your screen. The prototype of this file is located in /usr/athena/share/zephyr/zwgc.desc This can be copied into your home directory and modified. ~/.zwgc/desc is a **csh** script, unlike most of your other dotfiles.

Terminal Type Mode (ttymode) Login

Summary
A terminal style session is the type you get when connecting to the Athena dialups or connecting to an Athena machine via **ssh**.

Start of a Terminal Style Session
For a standard terminal (tty) session, the system starts the special "login shell" and goes directly to the Environment Setup phase. Environment Setup is exactly the same in window and tty sessions.

If you have a file named ~/.hushlogin (it can be an empty file), then the system will not echo status messages about initialization (i.e., messages such as "Setting up environment..." will not be echoed to your screen during start-up). This speeds up tty-session initialization, especially over slow dialup connections.

The "login shell" differs from other shells in that it runs the ~/.bash_login file first. This file calls the system file /usr/athena/lib/init/bash_login. Although this file simply calls another file, the ~/.login file is necessary because standard UNIX expects to find a login initialization file in your home directory.

The environment setup section of initialization is the part associated with the files ~/.bashrc, /usr/athena/lib/init/bashrc, ~/.bash_environment, and ~/.bashrc.mine. All of these files do the same thing in terminal sessions. Your environment and path will behave the same and lockers are attached normally. Anything you could do in an **xterm**, you can do normally in a terminal style session.

Warning: Do not make changes to ~/.login. Make changes to the tty initialization through the ~/.environment and ~/.startup.tty files.

Tty-oriented Start-up
Activity begins in earnest. Once the "tty" session start-up activities run, you can interact with the login shell. The start-up activities can be overridden through the ~/.startup.tty file; see below. Any text on a line after the # character will be ignored by the interpreter.

Run standard start-up activities (check mail, start **zephyr**, etc.)

In this step, the system runs standard activities that most users want to have run automatically:

set ignoreeof

zwgc

from -t -n

^D won't log you out

tries to start a **zephyr** client

checks for new mail

To override any one of these steps, you must override the whole set, then include the ones you want to run in your ~/.startup.tty file (see the next step). To override the default set of steps, include the following statement in your ~/.environment file: set skip_tty_startup

Run user-specified start-up activities (if file ~/.startup.tty exists)

~/.startup.tty
Start-up activities for tty sessions (e.g., **znol**, **zwgc -ttymode**, etc.). This is the equivalent file to ~/.startup.X.

In this step, the system session file runs all of the commands contained in the user file ~/.startup.tty. For example, ~/.startup.tty might contain:

znol -l

fortune

check who is online

display a random quotation or witticism

Note: In the ~/.startup.tty file, run in the background any command that will not exit quickly. To run a command in the background, put an ampersand, & after the command. In the examples above, this is not necessary because they will exit quickly.

Note: When you log in for your next Athena tty session, type the following command at the prompt:

```
athena%
zctl set fallback true
```

You need issue this command only once in your Athena tty-session career. This sets the fallback variable so that **zephyr** runs in tty mode, by default, when you log into a tty session.

Tty-oriented Logout
When you logout, a few tty-oriented system wrapup activities are conducted after the session file is completed (e.g., deactivating the workstation and detaching lockers), but these are not oriented toward the specific user's session. You can, however, specify your own wrapup activities in a ~/.logout file. To have the system run any special post-session activities, you can include commands in the file ~/.logout. Remember that the commands in this file will also be used for "window" session logout. See section on .logout.

Appendices

Appendix A: Shells

A shell, or command interpreter, tries to make sense of what you type and relay that to the operating system. A shell is automatically started in each **xterm** window that is created. Basically, whatever you type at an **xterm** is read and interpreted by the shell and then passed along to the operating system to be executed.

There are a lot of customizations that can be made to the shell; aliases to shorten commonly used commands are just the most common. The shell finds the locations of programs, interprets expansions (ex **ls *.exe**), substitutes values of shell and environment variables, and interprets aliases.

As you probably have already figured out, you can have multiple copies of the shell running at any given time. Some programs, such as xterms, start them. Every time a shell is started associated customization files are read. The default sell is **bash** but there are other shells that exist that you can use instead.

Appendix B: Bash Special Characters

Tcsh is the default Athena shell. Tcsh has some special characters that you should be aware of when making aliases or anything else that will be interpreted by the shell.

~

?

*****

" "

' '

[ ]

{ }

( )

;

>

<

|

****

An abbreviation for the home directory of the user foo is /mit/foo/

Any single character, kind of a fill in the blank thing

Any file name or part of a file name. * means all files, x* means all files starting with x, x.* means all files beginning x. and *.x means all files ending in .x. It even works for x*x where you get all files beginning and ending in x. This is a lot like ? but there's not a limit on the number of character to fill in with.

Used to group things. Anything within the "" will be treated as one word, except variables which are still expanded.

Same as above but won't expand variables.

Encloses list of character options: *.[ch] will list all files ending in .c or .h

Does it for words ls ~/1.00/{prob1, prob2, prob3}.c

Encloses commands to be executed in a sub shell.

Separates commands you want run on a separate line.

Takes output of a command and writes it to a file.

Supplies the file as input to a command ex: zwrite friend < file.

Takes the output of one command and makes it the input of the next command.

If you want one of the above characters as itself put a \ in front of it.

Appendix C: Variables

Variables are used to store values that can be used by other commands and programs. These values can be set and read from the shell or through dotfiles. There are 2 kinds of variables: environment and shell. Shell variables are set by

variablename=value
PS1="athena\$"

Environment variables are set by:

export variablename=value
export TERM=vt100

To display them:

```
athena%
echo $PS1
athena%
echo $TERM
```

Note that any shell variable can be turned into an environment variable simply by **exporting** it. For example, if you had previously set the shell variable FOO by typing `FOO=bar`, and then later decided you wanted that to be an environment variable, as long as you were in the same shell in which you set FOO, you could type `export FOO` to make it an environment variable.

The difference between environment and shell variables is that shell variables can only be read in the shell, whereas environment variables can be read by any subprocess, including other shells. However, if you open an **xterm** or **Terminal** window, and set an environment variable there, it will not be available in other **xterm** or **Terminal** windows, because the shells in those windows were not subprocesses of the shell in which you set the variable

There are some variables that programs expect to be able to find called standard variables. Some examples of these are **$PRINTER, $HOME**, **$EDITOR**, and so forth. It is possible to make your own variables and give them values just as you would a standard variable.

To view all the environment variables set by the shell, you can use the "printenv" command.

```
athena%
printenv
```

To view only shell variables, you can use the "set" command. You will probably want to pipe the output of "set" into a command like **more** or **less**, since it will also show you all shell functions.

You can customize your account temporarily by setting variables from a shell, but to customize permanently you must make these changes in the appropriate dotfiles, which is covered in the customizations section of this document.

Appendix D: Testing Dotfiles

The best way to avoid having to use the next section on trouble shooting dotfile errors is to test them ahead of time. Aliases and variables can be set from the command line. Before adding them to your dotfiles, try them out first on the command line. Note that some of these won't do anything if set after start-up. For example, setting **skip_lert** after you've already logged in will not cause any noticeable change. Setting an alias and then testing the alias, however, is very useful.

Source a new or changed login related dotfile from an xterm before logging out. This will make sure there are no errors that will keep the file from completing its tasks. Dotfiles may not do exactly what you think they will when you login, but if you test the dotfiles first and there are no failures, at least, you should be able to login successfully. The testing is done by:

```
source .dotfile
athena%
source ~/.bashrc .mine
```

Aliases can be tested individually at the prompt

```
athena%
alias dir='ls -l'
```

Then simply type dir at the prompt and see if the alias does the right thing. Setting variables can be done in much the same way. Here some examples for shell and environment variables respectively:

```
athena%
PS1="my new prompt "
athena%
export PRINTER=ajax
```

Whether the variable was set or not should be easy to test in this case. Some variables are harder to test whether they work or not without logging in again, but you can always check the value of a variable to make sure it was set.

```
athena%
echo $PS1
athena%
echo $PRINTER
```

For program related dotfiles, save the old dotfile under a new name before making changes. Then make your changes and try running the program with the new dotfile. This way you're sure to have the old version to go back to and look at to see where you made changes or use until you can fix the new version. Saving the old versions of all your dotfiles until you get the new ones working is generally a good idea.

To test changes to the ~/.Xresources file use the command below. After you have done this you must quit and restart programs to see any changes made to their settings. Otherwise your programs will continue to run under the old settings.

```
xrdb -load ~/.Xresources
```

You should change and test just one command at a time. This makes it much easier to locate and correct errors.

Appendix E: Troubleshooting Dotfiles

While making customizations it is possible to make mistakes that will make it difficult for you to login. Here are a few tips on how to deal with this, should it happen. Proper testing ahead of time can help avoid most mistakes that will make it hard to login, but there are always exceptions.

If you can't login normally, try logging in without customizations. As long as you stick to dotfiles mentioned in this document as user alterable, this should work. Another option is to login for a terminal style session. Also, it is best to test terminal mode customizations when you have access to an actual workstation, so that if a mistake is made you can login without customizations. There is no easy way to login without customizations using the Athena dialup servers.

Once you have logged in without customizations, the first thing to do is figure out where the problem is located. The easiest way to do this is to modify only one dotfile at a time and try logging in with the new one. This will help you determine which file contains the mistake. If you have already made changes to many files, try putting this at the top of each of your dotfiles (except ~/.Xresources and ~/.zwgc.desc) you made changes to:

```
echo "sourcing dotfile foo"
```

and this at the end

```
echo "done with dotfile foo"
```

Replace **foo** with the name of the dotfile. During session initialization the messages will appear in the console window so you can tell what dotfile is being sourced at any given moment. You'll know which one breaks the system by noting which one starts but doesn't finish. Looking at the flow charts of session initialization while this is happening will help it all make sense.

Once you know which dotfile is causing the problems, take a look at the dotfile and see if you mistyped anything. If not, see if you made a logic error or some other mistake. If you still can't figure out what is wrong you can copy a day old version of the file (if the dotfile existed before you made changes) from your OldFiles directory. Once you have a working file, try adding back in the changes one at a time.

Another option is to "comment out" sections of the dotfile that you changed to narrow down where the problem is located. This is done by putting the character # in front of each line you wish to be ignored. The # can simply be deleted at a later date to have the line sourced again. By commenting out changes and adding them back in one at a time the error can usually be located.

If all else fails, get the default user dotfiles and copy them into your account. See Appendix H.

Appendix F: Logging Out

Sometimes dotfile errors can make it difficult to logout. Here are some ways to logout.

- Type **logout** in any **Terminal** window
- Choose "Log Out" from the "System" menu
- If none of these options are available or fail due to changes you made, the last resort is to reboot the machine. Do not reboot a workstation unless there is no other option available.

Appendix G: Learning More

One of the best ways to learn about dotfiles is to look at other people's dotfiles and see what has been done. Do not copy someone else's dotfiles into your home directory. Some customizations are user specific and could cause your login to fail. The best way is to read other dotfiles and add things you like into your own. Make sure you understand what a command does before adding anything to your files.

Finding dotfiles is easy – there is a locker named dotfiles that contains dotfiles that people have made in the past. Take a look at them and see if anything there looks like it could help you. This is not a supported service and there are no guarantees, so be careful and only make changes to your own dotfiles that you understand. Read the file http://web.mit.edu/dotfiles/README for more information about the dotfiles locker and its uses.

You can learn a lot about X Resources by reading the X manual page and the manual pages for programs you wish to customize. To view the manual page for X windows, zephyr, ls, or other programs or commands use the following commands. For most manpages simply use the name of the program or command you would type at the prompt.

```
man programname
athena%
man x11
athena%
man zwgc
athena%
man ls
```

Appendix H: Default Dotfiles

If you make errors in your dotfiles, you may wish to revert to the default dotfiles. The directory /usr/prototype_user contains all the default dotfiles that come with new accounts. For example, if you wanted to restore your original .bashrc (which you should not be customizing anyway), you could use the following command:

```
athena%
cp -f /usr/prototype_user/.bashrc ~
```

The "-f" option to "cp" tells it to replace the existing file regardless, and the tilde character (~) is shorthand for your home directory.

Appendix I: Examples

Example dotfiles are available on Athena in the `/mit/dotfiles/Examples` directory. Please be sure to read the `README` file in that directory before attempting to use any of the example dotfiles.

The `dotfiles` locker also contains user-contributed dotfiles which demonstrate different ways people have customized their accounts.

Appendix J: Sourcing a File

The command to source a file is:

source filename
athena%
source ~/.bashrc.mine

When a file is sourced, the information in that file usually consists of environment changes to the shell. Instead of starting a new shell with the modifications, sourcing a file allows the modifications to be made to an already existing shell. Sourcing a file executes a shell script within the current shell. Environment modifications take effect immediately in that shell.

When a file is sourced during session initialization the modifications carry through to all shells you use in that session. However, if you source a file in an **xterm**, the modifications to the environment take place only in the shell in that xterm. It does not affect any of your other shells.

Appendix K: Customizations Chart

Customization	Associated Dotfiles
Specify Locations/Colors of Zephyrs	~/.Xresources
A Screen Background	~/.startup.X
Set Aliases	~/.bashrc.mine
Attach lockers	~/.bash_environment
Skip initial xterm or quota warning	~/.bash_environment
Set default editor	~/.bash_environment
Disable Printing header pages	~/.bash_environment
Set custom prompt	~/.bashrc.mine
Set screen location for programs	~/.Xresources
Set colors and fonts for programs	~/.Xresources
Set other program resources	~/.Xresources
Start a program upon login	~/.startup.X or ~/.startup.tty
Execute a coomand after logout is typed	~/.logout
Set size of core files	~/.bash_environment
Put a scrollbar on your xterm	~/.Xresources

Appendix L: Quick Stations

Quickstations are workstations designed to be used for only a few minutes to check mail, print out a copy of a paper, or anything else that can be done in a couple of minutes. When logging into a quickstation, you may not want to use all of your customizations, especially if they take a long time to load. At the same time you may not want to login without customizations. In order to make this possible there is an environment variable $ATHENA_QUICK. $ATHENA_QUICK is set only if you are logged into a quickstation. It is possible to test wether or not you are logging into a quickstation or a regular workstation during session initialization. This allows you to choose what startup activities to use in a quickstation login versus a regular login. In the example below, the "-n" is an operator that tells the bash shell to test if the length of the string is nonzero (ie: it has a value).

{{

```
if [ -n "$ATHENA_QUICK" ]; then
    # it's a quickstation, so do things differently
    command1
    command2
    ...
else
    # It's not a quickstation, so do my normal stuff
    commandA
    commandB
    ...
fi
```

}}

If you login to a quickstation commands one and two will be used, while if you login to any other kind of workstation commands A and B will be executed. Any commands to be run regardless of wether it is a quickstation should be left as they are in your dotfiles. Sections of customized dotfiles are included below.

~/.bash_environment:

{{

```
#always do this:
set skip_quotawarn

if [ -n "$ATHENA_QUICK" ]; then
    # If this is a quick station do the following:

    add consult
    export EDITOR=emacs
else
    #If it's not a quick station do this:
    add consult infoagents audio  tcl  graphics sipb calendar frame
    export EDITOR=gedit
fi
```

}}

Alternatively, you can test whether something is **not** a quickstation by changing the "-n" to a "-z" as in the example below. This tells bash to test whether the string has a zero length (ie: is empty or unset).

~/.startup.X

{{

```
#always do this:
xclock

if [ -z "$ATHENA_QUICK" ]; then
    # If this is NOT a quick station do the following:
    firefox &
    emacs &
fi
```

}}

This type of customization cannot be made in your .Xresources file because that information is loaded with the **xrdb** command. It can be used with almost any other user customizable files, although the most time saving customizations to make will be to avoid adding lockers you won't use during a quick login and to not start programs that take a long time to load, such as **Firefox**. Setting aliases and variables take a relatively short time and will not make as much difference in your session initialization time as starting programs and adding lockers.

Related Links

Dotfiles on Athena
Dotfiles for Tcsh
Athena Help
Athena at MIT